# An approach for online learning in the presence of concept changes

Ghazal Jaber

Université Paris Sud 11

Doctoral Thesis

# An approach for online learning in the presence of concept change

*Author:*
Ghazal Jaber

*Supervisor:*
Prof. Antoine Cornuéjols
Prof. Philippe Tarroux

*A thesis submitted in fulfilment of the requirements for the degree of Doctor of Computer Science*

*in the*

LIMSI-CNRS

October 2013

*Doctoral Committee:*
Antoine CORNUÉJOLS (*Co-Advisor*)
Philippe TARROUX (*Advisor*)
João GAMA (*Reader*)
Vincent LEMAIRE (*Reader*)
Michèle SÉBAG (*Examiner*)
Younès BENNANI (*Examiner*)

UNIVERSITÉ PARIS SUD 11

# *Abstract*

**An approach for online learning in the presence of concept change**

Learning from data streams is emerging as an important application area. When the environment changes, as is increasingly the case when considering unending streams and long-life learning, it is necessary to rely on on-line learning with the capability to adapt to changing conditions a.k.a. *concept drifts*. Adapting to concept drifts entails forgetting some or all of the old acquired knowledge when the concept changes while accumulating as much knowledge as possible when the underlying concept is supposed stationary. This tradeoff is called the *stability-plasticity dilemma*.

Ensemble methods have been among the most successful approaches. However, the management of the ensemble which ultimately controls how past data is forgotten has not been thoroughly investigated so far. Our work shows the importance of the forgetting strategy by comparing several approaches. The results thus obtained lead us to propose a new ensemble method with an enhanced forgetting strategy to adapt to concept drifts. Experimental comparisons show that our method compares favorably with the well-known state-of-the-art systems.

The majority of previous works focused only on means to *detect* changes and to *adapt* to them. In our work, we go one step further by introducing a meta-learning mechanism that is able to detect relevant states of the environment, to recognize recurring contexts and to anticipate likely concepts changes.

Hence, the method we suggest, deals with both the challenge of optimizing the stability-plasticity dilemma and with the anticipation and recognition of incoming concepts. This is accomplished through an ensemble method that controls an ensemble of incremental learners. The management of the ensemble of learners enables one to naturally adapt to the dynamics of the concept changes with very few parameters to set, while a learning mechanism managing the changes in the ensemble provides means for the anticipation of, and the quick adaptation to, the underlying modification of the context.

# *Acknowledgements*

First and foremost, I wish to thank professor Antoine Cornuéjols, who was not only my advisor but also my mentor. He has been supportive since my masters internship and throughout the long road to the end of my thesis. He gave me the freedom I needed in my research, gave me confidence in my work and was always present for guidance. Thank you Antoine for giving me the opportunity to work with you. It is with sadness that I leave your research team at AgroParisTech but I am hopeful that we will meet in conferences, and maybe, contribute in new publications. I also want to thank professor Philippe Tarroux, my advisor, for the many brainstorming discussions we had. It was a great honor to work with you at the LIMSI laboratory.

Other people helped me through my thesis. Thank you Laurent Orseau for your advices and guidance. As I already said: you will make a great doctoral advisor. I will miss our theological discussions and also our daily chocolate gathering with Antoine Cornuéjols and Christine Martin.

I want to thank my thesis readers, professor João Gama and Vincent Lemaire, who kindly accepted to review my work and took the time in reading and providing me with valuable comments despite their busy schedules. I also wish to thank the other members of the jury who kindly accepted to evaluate my thesis.

I am grateful to Leandro L. Minku who provided me with the empirical results published in his paper [70] on his online ensemble method DDD, and other state-of-the-art methods. I also want to thank him for his artificial datasets who helped me evaluate my algorithm.

This thesis is also the result of many people that supported me emotionally:

*Mom*, you are the source of endless love and kindness. My life would be unbearable and unmeaningful without you. Thank you for you unconditional love and support.

*Dad*, you never told me what to do in my academic life, but you raised me to challenge myself and overcome my fears. Thank you for being a role model to me.

*Nihad*, your love helped me overcome difficult times. Thank you for being my best friend, a supportive husband, and a caring lover.

*Nour* and *Fouad*, you are the best sister and brother anyone could ask for. I wish I were with you during the last three years. I thank you for believing in me and I hope I made you proud of your big sister.

Life can't be fun without friends. *Hana* and *Abir*, thank you for the special moments we shared together. You will always have a special place in my heart.

To my second family, my aunt, uncle and cousins. You were my rock during my masters and thesis. You supported me emotionally, cheered me up when I felt low and fulfilled my life in the absence of my parents. Thank you for your presence, warmth and love. A special thanks to my cousin *Aliaa Abbara* who listened to all my presentations' rehearsals throughout my thesis and who can now present my work even better than myself.

# Contents

# List of Figures

# List of Tables

# Resumé

Les années récentes ont connu l'émergence d'applications de flux de données où les données d'apprentissage sont reçues sous la forme d'un flux infini, et les prédictions sont faites à la volée. Des exemples de telles applications comprennent entre autres les systèmes de gestion de l'énergie électrique, l'analyse du panier de consommation, les systèmes de filtrage du spam et les prévisions météorologiques.

Les méthodes d'apprentissage automatique classiques ont été utilisées avec succès pour induire une fonction de classification (un concept) à partir des batchs statiques de données [14]. Cependant, les applications de flux de données ont introduit de nombreux défis que les méthodes classiques n'ont pas été conçues pour traiter. En premier lieu, les flux de données ne peuvent pas être stockés ou retraités, en raison de contraintes en terme de mémoire et de temps d'exécution. Deuxièmement, contrairement au cadre d'apprentissage classique, l'utilisation de la fonction de classification ne peut pas attendre jusqu'à ce que la phase d'apprentissage soit terminée, car le flux de données d'apprentissage pourrait être sans fin. Par conséquent, la fonction de classification induite devrait pouvoir être utilisée à tout instant, tout en étant mise à jour avec chaque nouvelle information reçue du flux. Enfin, la théorie classique de l'apprentissage automatique, basée sur l'hypothèse que les données d'apprentissage sont indépendantes et identiquement distribuées, n'est généralement pas valide dans les cas des flux de données. Les données d'apprentissage reçues peuvent dépendre du temps, et l'environnement sous-jacent peut évoluer, modifiant ainsi la distribution sous-jacente des données d'apprentissage, créant une *dérive de concept* [101]. Dans les systèmes de filtrage de spam, par exemple, le concept classe les emails en catégories : spam ou non-spam en fonction de leur contenu. Le concept est susceptible de devenir moins précis avec le temps vu que les spammeurs tentent constamment de tromper ces systèmes en modifiant les propriétés statistiques de la catégorie des spams.

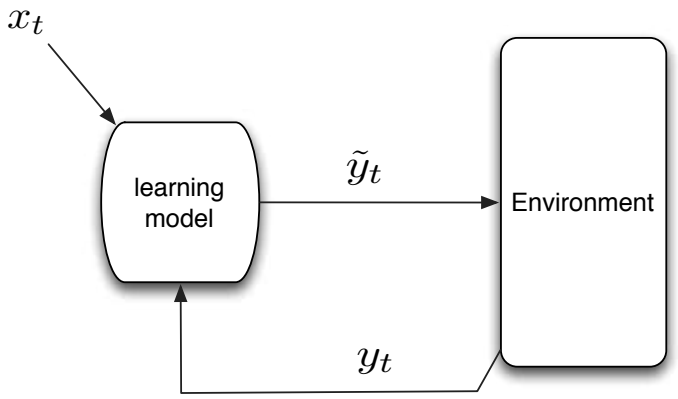


FIGURE 1: Le processus d'apprentissage en ligne. $\mathbf{x}_t$ représente un vecteur d'entrée, $\tilde{y}_t$ est la sortie prédite par le système d'apprentissage et $y_t$ la valeur réelle.

L'apprentissage à partir de flux de données est réalisé par des méthodes d'apprentissage en ligne (voir Figure 1). Lors de l'apprentissage en présence de dérive de concept, une préoccupation centrale est d'optimiser un compromis entre l'apprentissage à partir d'autant de données possibles, afin d'obtenir le modèle de classification le plus précis; et la reconnaissance des données obsolètes et potentiellement trompeuses, empêchant l'adaptation aux nouvelles tendances. Ce compromis est connu sous le nom de *dilemme stabilité-plasticité* [31, 64]. La stabilité requiert l'accumulation de connaissances sur le concept sous-jacent supposé stationnaire, alors que la plasticité nécessite d'oublier la totalité ou une partie des anciennes connaissances acquises afin d'apprendre le nouveau concept à venir.

Les dérives de concept sont communes considérant des flux de données infinis et des tâches d'apprentissage de longue durée. Les travaux récents se sont focalisé sur la *détection* des changements de concept ainsi que les meilleures approches permettant d'*adapter* le système d'apprentissage aux changements. Par conséquent, les approches courantes attendent passivement que les changements de concept surviennent plutôt que de prédire pro-activement ce qui est susceptible de se produire.

Nous soutenons que, dans ce contexte, l'adaptation passive aux changements peut ne pas être la meilleure stratégie d'apprentissage. Il peut être possible de profiter de l'information présente dans la séquence de données. Par exemple, les erreurs de classification coûteuses peuvent êtres évitées en reconnaissant une situation *récurrente* ou en *anticipant* l'évolution probable à venir. L'adaptation passive et l'anticipation pro-active sont des approches non contradictoires qui peuvent potentiellement fonctionner simultanément. Néanmoins, elles exigent différentes stratégies d'apprentissage. Alors que l'*adaptation* implique l'oubli d'informations périmées, l'*anticipation* nécessite de garder des traces du passé afin de prédire l'avenir.

Dans cette thèse, nous présentons DACC (Dynamic Adaptation to Concept Changes), une méthode d'ensemble conçue pour l'apprentissage à partir de flux de données avec *adaptation aux changements de concepts*. Notre méthode adaptative est ensuite couplée avec un mécanisme d'anticipation des concepts futurs ; le système résultant est nommé ADACC (Anticipative and Dynamic Adaptation to Concept Changes).

## DACC

Avant de concevoir une nouvelle méthode d'adaptation, nous étudions dans un premier temps les méthodes d'ensemble capables d'apprendre en présence de changements de concepts. Elles présentent deux stratégies principales pour oublier les informations obsolètes dans un environnement en évolution: **(a)** la suppression d'apprenants ayant une performance prédictive médiocre selon un seuil prédéfini, et **(b)** la suppression périodique de l'apprenant le moins performant de l'ensemble.

Dans les deux cas, les apprenants supprimés sont remplaçés par de nouveaux apprenants n'ayant aucune mémoire du passé. Alors que les anciens apprenants sont fiables dans un environnement stable, les plus jeunes sont plus efficaces confrontés aux périodes de changement.

Notre analyse des deux stratégies montre qu'une connaissance préalable de la dynamique de l'environnement est nécessaire afin de faire le choix du seuil de suppression pour la méthode **(a)** ou la fréquence de suppression pour la méthode **(b)**. Un choix inadapté des valeurs des paramètres peut engendrer des résultats indésirables, affectant la stabilité de l'ensemble ou sa plasticité vis a vis d'un changement de concept potentiel.

Ce travail de thèse a été motivé par la nécessité d'atténuer l'effet des connaissances a priori. Nous nous sommes concentrés sur la stratégie **(b)** afin d'éviter de trouver une valeur de seuil appropriée. Néanmoins, le choix de la fréquence de suppression $f_{del}$ pour la stratégie **(b)** ne va pas sans difficultés. Une valeur élevée de $f_{del}$ prône la suppression des apprenants jeunes si ces derniers n'ont pas eu le temps de capturer les régularités dans l'environnement. Ceci entrave la capacité du système à s'adapter aux changements de concept potentiels. La valeur de $f_{del}$ ne peut pas être relativement faible, sinon le système perd toute plasticité.

La question est alors de savoir s'il est possible de supprimer des apprenants de l'ensemble de manière fréquente, en prévision d'un changement de concept, tout en gardant les jeunes apprenants lors d'un changement de concept éventuel. Pour résoudre ce problème, nous proposons une stratégie de suppression qui, au lieu de supprimer l'apprenant le

moins performant de l'ensemble, supprime un apprenant au hasard parmi les *ds* apprenants les moins performants, et ceci, dans le but d'augmenter l'espérance de survie des jeunes apprenants lorsque la fréquence de suppression $f_{del}$ est élevée. Choisir la valeur de *ds* reste cependant un defi. De grandes valeurs de *ds* augmentent l'espérance des jeunes apprenants à survivre mais risquent également de supprimer les " bons apprenants" de l'ensemble, affectant ainsi la stabilité du système. De faibles valeurs de *ds* favorisent la suppression des jeunes apprenants, affectant par conséquent la plasticité du système. D'après une étude expérimentale, une valeur de *ds* qui est égale à la moitié de la taille de l'ensemble offre un bon compromis. La nouvelle stratégie de suppression constitue la base d'une nouvelle méthode d'ensemble en ligne appelée DACC.

### Propriétés

DACC s'adapte aux changements de concept en utilisant un comité dynamique et diversifié d'apprenants. Le comité est *dynamique* car il se met à jour en permanence en supprimant les apprenants les moins performants et en ajoutant de nouveaux apprenants. Le comité est aussi*diversifié* car constitué de différents types d'experts ou d'apprenants (réseaux de neurones, modèles de régression polynomiale, machines à vecteurs de support, etc ..). En ajoutant et supprimant des membres de l'ensemble, les apprenants sont entraînés sur des fenêtres de données d'apprentissage différentes. Ainsi, DACC gère indirectement la mémoire du passé des apprenants, ce qui lui permet de traiter *implicitement* le dilemme stabilité-plasticité. Cet avantage ne peut être obtenu avec un seul apprenant entraîné sur une fenêtre d'apprentissage de taille fixe.

Contrairement aux méthodes d'ensemble traditionnelles où le comité est composé d'experts faibles qui apprennent le même concept et coopèrent pour donner la prédiction finale [77], DACC est un mélange d'une stratégie *compétitive* et *coopérative* entre experts qui pourraient être formés sur des données provenant de concepts différents. D'une part, les experts sont en concurrence pour ne pas être supprimés de l'ensemble. D'autre part, les meilleurs apprenants coopèrent en votant pour la prédiction finale de l'ensemble. Deux fonctions de combinaison de votes sont utilisées: MAX et WVD. Dans les deux cas, les apprenants sont évalués selon leur performance prédictive sur les données récentes et des poids leur sont accordés. La fonction MAX applique un vote majoritaire entre les apprenants ayant la meilleure valeur de poids alors que la fonction WVD applique un vote pondéré entre les apprenants faisant partie de la meilleure moitié de l'ensemble. Alors que MAX ne prend en compte que les prédictions des meilleurs, excluant le vote des apprenants obsolètes en cas de dérive de concept, WVD conduit à une performance prédictive plus stable dans un environnement bruité.

Les résultats expérimentaux montrent que notre approche permet de surmonter beaucoup de difficultés communes rencontrées dans les méthodes d'ensemble courantes conçues pour s'adapter aux dérives de concept. Les principaux avantages de DACC comprennent:

**Une connaissance a priori minimale de la dynamique de l'environnement**

Contrairement à beaucoup d'approches, DACC ne s'appuie pas sur une valeur de seuil pour décider si un apprenant est adapté au concept courant ou s'il est obsolète et doit donc être supprimé. Ceci évite de choisir une valeur de seuil adaptée qui dépend évidemment des propriétés du changement de concept telles la vitesse du changement ou sa sévérité.

Nous montrons que DACC, étant moins sensible à ses paramètres prédéfinis, peut s'adapter à une large variété de changements de concept, tout en fixant les mêmes valeurs de paramètres.

**Le transfert de connaissances**

DACC ne supprime pas tous ses experts à la fois lorsqu'une dérive de concept se produit. Une période de $1/f_{del}$ sépare deux suppressions consécutives ce qui garantit que l'ensemble ne peut être réinitalisé avant au moins $N/f_{del}$ pas de temps, où $N$ représente le nombre d'apprenants dans l'ensemble. Ceci permet à l'ensemble de transférer ses connaissances acquises de l'ancien concept au nouveau lorsqu'une dérive de concept se produit, et si le changement de concept est lent et continu, DACC peut toujours faire des prédictions sur les instances de l'ancien concept durant la dérive.

**Adaptation rapide aux changements**

DACC met à jour son ensemble de manière fréquente, supprimant constamment des apprenants, indépendamment de l'état de l'environnement : stable ou en évolution. Dans certains cas, cela conduit à une adaptation rapide au changement de concept comparée aux autres approches classiques qui attendent que la performance d'un apprenant soit assez faible, ou que le système de détection de changement "ressente" la dérive de concept, avant de décider d'effectuer une mise à jour du système d'apprentissage.

### Des niveaux de diversité dynamiques

Les niveaux de diversité au sein du comité d'apprenants évoluent dynamiquement avec le temps. Lorsque le concept est stable, les meilleurs apprenants, n'étant pas supprimés de l'ensemble, affinent leur connaissance du concept cible sous-jacent, convergeant ainsi vers une performance prédictive presque maximale. Ceci se traduit par un niveau de diversité faible parmi les meilleurs apprenants, à l'opposé de la moitié la moins performante de l'ensemble qui, subissant des opérations de suppression fréquentes en vue d'un changement de concept, a un niveau de diversité élevé. Lorsque le concept évolue, la diversité augmente chez tous les apprenants. Les anciens apprenants, jadis performants, deviennent obsolètes et sont par conséquent remplacés par de nouveaux. Le groupe d'apprenants se restabilise et les niveaux de diversité reprennent la forme expliquée précédente (le cas de stabilité).

Les niveaux implicites de diversité dans DACC permettent aux apprenants ayant un niveau de diversité faible de faire des prédictions sur le concept stable courant, et aux apprenants ayant un niveau de diversité important d'être prêts à tout changement de concept. Le suivi de l'évolution des niveaux de diversité dans l'ensemble permettra au système de méta-apprentissage anticipatif ADACC de détecter les états de stabilité de l'environnement; nous l'expliquerons plus loin dans la section sur ADACC.

## Forces, faiblesses et perspectives

Un grand nombre de méthodes en ligne s'appuie sur une valeur de seuil pour s'adapter aux changements de concept. La valeur de seuil est généralement utilisée soit pour décider si un apprenant est obsolète et devrait donc être supprimé, ou pour détecter un changement de concept de manière explicite en surveillant, par exemple, la performance prédictive de l'ensemble. Dans tous les cas, la valeur de seuil joue un rôle important dans la capacité du système à s'adapter à la dynamique de l'environnement. En revanche, DACC supprime des apprenants fréquemment de l'ensemble, sans analyser si le concept cible sous-jacent est stable ou en évolution. Ainsi, contrairement aux autres méthodes d'ensemble, DACC n'est pas confronté au problème de non-adaptation aux changements ; il existe toujours des jeunes apprenants dans l'ensemble prêts à apprendre le nouveau concept.

Grâce à la suppression fréquente des apprenants, DACC s'adapte généralement plus rapidement aux changements de concept comparé à ses concurrents. Néanmoins, la réactivité de DACC n'est pas aussi rapide que celle des systèmes utilisant un mécanisme explicite de détection de changements de concept, et ceci dans le cas particulier où le changement est *sévère* et *soudain* ce qui est généralement plus facile à détecter. Une des

orientations futures possibles est de combiner notre ensemble adaptatif avec un ensemble utilisant un mécanisme de détection de changements, afin de tirer profit des avantages des deux approches tout en atténuant leurs inconvénients.

Lors de d'apprentissage à partir de flux de données, DACC donne la même importance aux exemples de chaque classe, faisant donc l'hypothèse que les classes sont équilibrées. Selon les résultats empiriques, ceci conduit à des difficultés lors de l'apprentissage en présence de classes déséquilibrées. Une solution possible consiste à échantillonner la(les) classe(s) minoritaire(s) et/ou de sous-échantillonner la(les) classe(s) majoritaire(s) afin de surmonter le déséquilibre. Ceci peut être réalisé, par exemple, en pondérant les exemples d'apprentissage de manière inversement proportionnelle à la fréquence observée de leur classe, donnant ainsi plus d'importance aux classes minoritaires.

Enfin, les résultats empiriques montrent que le temps d'exécution de DACC se compare favorablement à d'autres méthodes d'ensemble. Cependant, vu que les apprenants dans l'ensemble sont évalués et mis à jour indépendamment l'un de l'autre, il reste possible de paralléliser DACC afin d'accélérer son exécution.

## ADACC

Dans ADACC, nous allons plus loin en introduisant un mécanisme d'anticipation qui garde en mémoire la trace des différents concepts stables rencontrés lors du streaming continu des données. Notre objectif est de profiter des changements dans l'environnement afin de prévoir le futur, nous permettant ainsi d'agir pro-activement face aux changements, plutôt que de s'y adapter passivement comme le fait la majorité des méthodes d'apprentissage en ligne. Nous nous sommes intéressés à deux cas particuliers:

- *la récurrence:* la réutilisation de concepts appris précédemment quand ils réapparaissent
- *la prédictabilité:* la prédiction de concepts futurs probables par l'analyse de l'évolution du concept cible sous-jacent (voir Figure 2).

La récurrence et la prédictabilité nécessitent que l'histoire passée des concepts stables soit capturée. L'utilisation d'un mécanisme de détection de changements est un moyen possible d'identification des différents concepts. Toutefois, ces mécanismes sont connus pour avoir des difficultés à détecter des changements de concept lents tout en étant robustes aux fausses alarmes ; ils ont par conséquent été évités dans ce travail.

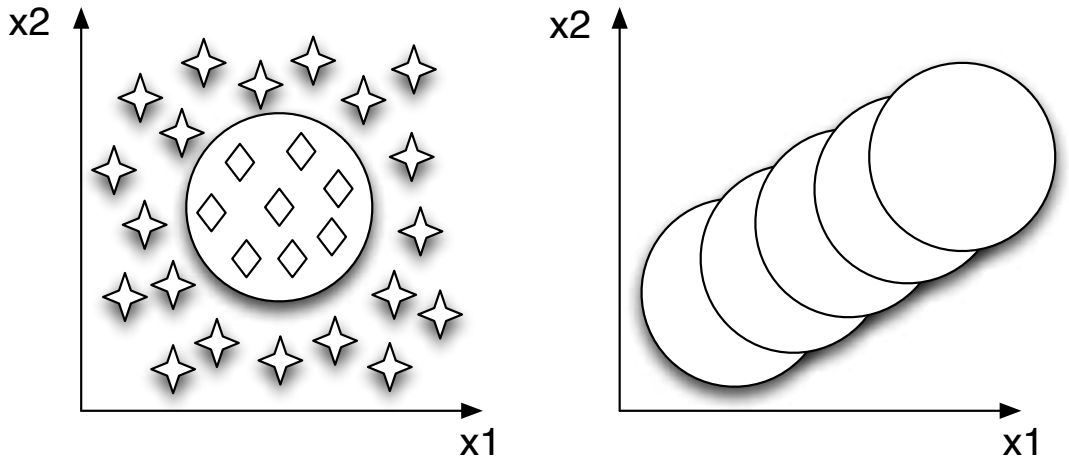


FIGURE 2: A gauche, un concept représenté par un cercle. Les exemples sont classifiés en deux classes: l'intérieur et l'extérieur du cercle. Droite: le cercle évoluant avec le temps, créant une dérive de concept.

Afin d'identifier les concepts stables, nous définissons une mesure de stabilité relative aux apprenants adaptatifs dans l'ensemble. Lorsque le concept a été stable pendant une période relativement longue, la performance prédictive des meilleurs apprenants adaptatifs converge vers une valeur presque maximale et leur diversité tend à diminuer. Lorsque la stabilité est suffisante, un " snapshot" du concept actuel est conservé en mémoire. Il s'agit d'une copie de l'apprenant ayant le poids le plus important dans l'ensemble, celui qui semble le mieux représenter le concept cible sous-jacent. Ainsi, la mesure de stabilité que l'on définit prend en compte à la fois la diversité des apprenants et leur performance prédictive.

La liste des snapshots stockée sert à deux fins. Tout d'abord, elle fournit une séquence de modèles successifs de l'environnement qui peut être analysée afin de prédire le concept suivant. Deuxièmement, elle garde en mémoire les anciens concepts dans le cas où l'un d'entre eux réapparaîtrait.

### Propriétés

La contribution principale de ADACC repose sur:

- l'utilisation d'une mesure de stabilité qui surveille l'ensemble adaptatif
- la mémoire des concepts significatifs appris par l'ensemble adaptatif

Le mécanisme anticipatif est complètement intégré dans le fonctionnement de la méthode d'ensemble adaptative. Ainsi, les snapshots sont évalués tout comme les apprenants

adaptatifs et leurs prédictions sont prises en compte suivant la fonction de combinaison utilisée. En utilisant la fonction MAX, un snapshot est utilisé si son poids est le meilleur parmi les autres apprenants adaptatifs et les autres snapshots.

Notre évaluation empirique explore diverses conditions d'évolution de concept dans des flux de données. Nous montrons que plus les changements de concept sont importants, plus les gains en performance prédictive sont grands en utilisant l'anticipation, comparée à l'utilisation d'un système adaptatif seul. En outre, le système anticipatif ne peut jamais détériorer les performances de prédiction. Dans le pire des cas, la performance prédictive de ADACC est égale à celle de DACC. Des expériences sur des jeux de données benchmark, réels et artificiels, ainsi que des comparaisons avec différents systèmes d'apprentissage en ligne montrent que ADACC améliore la performance prédictive de DACC, surperformant ses concurrents étudiés.

### Forces, faiblesses et perspectives

ADACC est un cadre général qui dote les méthodes d'ensemble adaptatives d'un mécanisme d'apprentissage de second ordre.

ADACC garde en mémoire la liste des snapshots des concepts stables rencontrés jusqu'à présent. Les snapshots mémorisés dépendent principalement de deux paramètres prédéfinis: un seuil de stabilité $\theta_i$ pour décider si l'environnement est "assez" stable, et un seuil d'équivalence de concept $\theta_d$ permettant d'éviter de stocker deux représentations consécutives redondantes du même concept cible. En choisissant une valeur élevée de $\theta_i$, le système risque de négliger un snapshot. Ainsi, au moment de choisir les valeurs des paramètres, nous nous assurons que la valeur de $\theta_i$ n'est pas très élevée. Le risque restant serait de stocker des snapshots redondants. Confrontés au cas de la récurrence de concepts, la présence de snapshots redondants ne nuit pas à la performance prédictive de ADACC. Toutefois, lorsqu'il s'agit de la notion de prédictabilité, les snapshots redondants seront considérés comme du bruit lors de l'analyse de la séquence des soi-disant concepts différents dans la liste. Cela signifie que les valeurs de seuils n'ont pas besoin d'être finement réglées pour le cas de la *récurrence* en contraste avec le cas de la *prédictabilité*, qui peut nécessiter une connaissance préalable de l'évolution de l'environnement afin d'éviter le bruit. Cependant, comme mentionné précédemment, ADACC peut seulement apporter un gain à la stratégie d'adaptation. Par conséquent, même lorsque les valeurs des paramètres ne sont pas adaptées, le pire scénario pour ADACC est d'avoir la même performance prédictive que l'ensemble adaptatif de DACC.

Toutes nos expériences concernant la prédictabilité des concepts futurs ont été appliquées sur des jeux de données artificielles, afin de simuler des changements de concepts réguliers.

Il serait d'autant plus intéressant d'étudier des scénarios de la vie réelle où le concept cible sous-jacent évolue de façon régulière selon une fonction d'évolution particulière, bénéficiant ainsi de l'approche de prédiction anticipative. Une amélioration importante de ADACC serait de trouver des moyens de garder constante la taille de mémoire des snapshots mémorisés. Une voie prometteuse est de stocker les prototypes des snapshots à la place des snapshots originaux, en utilisant une technique de clustering hiérarchique. Enfin, une étude théorique de la performance prédictive du système confronté à différents types de dérives de concepts sont des extensions possibles de ce travail.

# Chapter 1

# Introduction

In recent years, the domain of machine learning witnessed the emergence of data stream applications where the training data are received in an infinite stream and predictions are made on the fly. Examples of such applications include electricity management, market basket analysis, spam filtering systems, weather prediction and news filtering systems, among others.

Classical machine learning methods were successfully used to induce a classification function (a concept) out of static batches of data [14]. However, faced with data streaming applications, the classical methods were confronted with many challenges they were not designed to handle. First, streaming data cannot not be stored or reprocessed, due to memory and time constraints to allow real-time processing. Secondly, unlike classical learning framework, the use of the classification function cannot wait until the learning phase is over, as the stream of training data might be endless. Hence, the induced function should give answers in an any time fashion, while updating itself with each received information from the stream. Finally, the theory of classical machine learning, based on the assumption that the training data are independent and identically distributed, does not generally hold in data streams. The streaming data can be time-dependent and the underlying environment can evolve with time, changing the underlying distribution of the training data.

Learning from streams [36] is usually treated using online machine learning techniques. Aside from the challenges presented above, online learning methods are faced with the problem of *concept drift* where the underlying target concept (represented by the distribution of the streaming data) evolves time [101]. When learning under concept drift, one central concern is to optimize a tradeoff between learning from as much data as possible, in order to get the most precise classification model, while at the same time recognizing when data points become obsolete and potentially misleading, impeding the adaptation

to new trends. This is known as the *stability-plasticity dilemma* [31, 64]. While stability entails accumulating knowledge regarding the supposedly stationary underlying concept, plasticity, however, requires forgetting some or all of the old acquired knowledge in order to learn the new upcoming concept.

Concept drifts are common considering learning unending streams and long-life learning tasks. Recent works focused on evolving concepts in case of non-stationary environments, specially on how to *detect* concept changes and how to best *adapt* to them. Hence, they passively wait for the changes to occur and then try to follow them as best as possible rather than proactively predict what is likely to happen. In this context, passive adaptation to concept changes may not be the best learning strategy. Indeed, we may profit from the information possibly conveyed by the very sequence of data. For instance, one can gain precious time and avoid costly incorrect classifications by being able to recognize a *recurring* situation or to *anticipate* the likely evolution to come along. Passive adaptation and pro-active anticipation are non-contradictory approaches that can potentially operate simultaneously. Nevertheless, they require different learning strategies. While *adaptation* entails forgetting outdated information, *anticipation* entails keeping traces of the past in order to predict the future.

In this thesis, we first present an analysis of two main forgetting strategies used by online ensemble methods to adapt to concept drifts: (a) deleting learners with poor predictive performance, according to a preset threshold value, and (b) deleting periodically the relatively worst learner in the ensemble. The analysis shows that strategy (a) requires prior knowledge on the dynamics of the environment in order to choose an adapted threshold value, while strategy (b) may result in rather unwanted behavior, affecting the ability of the ensemble to adapt to new trends.

The analysis provides a base for a new forgetting strategy which deletes periodically one learner chosen randomly from the worst half of the ensemble. According to our study, this strategy corrects unexpected behaviors of (b). Empirical comparisons with a representative method based on (a), shows that the new forgetting strategy overcomes the difficulty of finding the appropriate threshold, and this on a large variety of concept drifts, with several levels of severity and speed.

We then go one step further by introducing an anticipative mechanism that keeps in memory the trace of the different stable concepts encountered during the data streaming. This is realized by monitoring a stability measure relative to the adaptive ensemble learners. The idea is that when the concept has been stable for a relatively long time, the classification performance of the best adaptive learners will converge towards the near maximal value and their diversity will tend to decrease. When stable enough, a "snapshot" (copy) of the current concept is kept in memory. The list of stored snapshots

serves two purposes. First, it provides a sequence of successive models of the environment that can be analysed to predict the upcoming concept. Second, it stores a memory of old concepts in case a recurring concept can be recognized.

The thesis is organised as follows.

**Chapter 2** describes the learning problem. The classical machine learning paradigm is presented and the new challenges introduced by the data streaming applications are discussed. This serves to introduce the framework of online machine learning along with the different benchmark datasets used to evaluate online learning systems.

**Chapter 3** describes the state-of-the-art in the area of online machine learning in the presence of concept drifts. We provide a categorization of the drift handling strategies and present briefly a number of online systems that will be used in our experimental studies.

**Chapter 4** starts with the description of the particular framework of online ensemble methods that adapt to concept drifts without explictly using a drift detection system. We then present an analysis of the main forgetting strategies used in this domain to deal with the stability-plasticity dilemma. The analysis provides a base for a new adaptive approach called DACC (*Dynamic Adaptation to Concept Changes*) with an enhanced forgetting strategy to handle concept drifts. Experimental results on benchmark datasets and extensive comparisons with state-of-the-art systems show that DACC adapts to a variety of concept drifts with different levels of drift severity and speed.

**Chapter 5** introduces the anticipative meta-learning mechanism that builds upon the adaptive ensemble. The anticipative and adaptive approach are combined in a new approach called ADACC (*Anticipative and Dynamic Adaptation to Concept Changes*). Empirical results show the advantage of the meta-learning mechanism compared to a mere adaptive strategy. Moreover, second order learning can never be detrimental to the overall classification performance as compared to the adaptive only policy.

**Chapter 6** summarizes our main contributions in the area of machine learning and presents possible directions for future work.

# Chapter 2

# The Problem

## 2.1 Classical Supervised Machine Learning

The aim of machine learning methods is to extract valuable information out of a (large) set of observations [14]. In case of supervised machine learning, an observation is represented by a pair consisting of an input vector and an output (target) value. The induced information has the form of a function that identifies the relationship between the inputs and their outputs.

Take for instance the task of classifying individuals into men and women. In this case, two possible output values exist: *man* and *woman*, while the input space describes the observed individuals. The input space can contain both real values (height, weight) and discrete or categorical values (eye color).

The importance of the induced function lies in its ability of generalization i.e. its capacity of predicting the outputs of new inputs variables. In our classification example, the induced function can be used to predict whether a new individual is a man or a woman.

### 2.1.1 Scenario

In classical supervised machine learning, it is supposed that a stationary environment $E$ generates inputs $\mathbf{x} \in X$ that are independently and identically distributed (*i.i.d.*) according to a static probability distribution $P(x)$. An *input* $\mathbf{x}$ is a vector in a $d$-dimensional feature space $X$; it is assigned a *target* value $y \in Y$ according to an unknown probability distribution function $P(y|\mathbf{x})$. The target value $y$, also called the corresponding *output*, *class* or *label*, can be represented as a multidimensional vector, in a more general formulation. In a classification problem, $y$ is a discrete value whereas in a regression problem

$y$ is a real value. The pair consisting of the feature vector and its target value $(\mathbf{x}, y)$ is called an *observation*, an *example* or an *instance*.

Learning systems aim at identifying the relationship between the inputs $\mathbf{x}$ and their corresponding target values $y$, and this by inducing:

1. the joint probability function $P(\mathbf{x}, y)$, or
2. a functional relationship $h : \mathbf{x} \rightarrow y$ which maps each input to a target value.

The first category of learning systems are called *generative* models [50] because they can generate new observations according to the induced probability distribution function. The second class of learning systems, which is our concern in this thesis, represents *discriminative* models [50, 72].

Discriminative models suppose the existence of an unknown function $f$ generating the observations. Their goal is to induce a function $h$ out of the observations that is as "close" as possible to the unknown function $f$. The induced function $h$ is usually called a *hypothesis*, while the unknown function $f$ is referred to as the real target function or the underlying target function. In case of a binary classification task where the target takes one of two possible values ($y \in \{0, 1\}$), the hypothesis is also called a *concept*. Please note that, in this thesis, we use the term concept to refer to any induced function, regardless of whether it was learnt from a generative or a discriminative model.

Classical learning systems undergo two separate phases: the learning and the predictive phase. In the *learning phase*, the system induces a hypothesis out of the observations. The observations, stored in a batch, can be processed multiple times by the learning system. Then, once the hypothesis is learnt, comes the *predictive phase* during which the system is used to predict the target values of new input variables. The observations used in the learning and predictive phase are called training and testing examples, respectively.

### 2.1.2 Performance criterion

A learning system learns a hypothesis $h \in H$ which given an input $\mathbf{x}_i$ returns a predicted target value $\tilde{y}_i$. The hope is to find a hypothesis that gives correct predictions, not only for the training examples used in the inductive learning process but also for testing examples. The question is how to quantify the quality of an induced function $h$.

A function $h$ giving correct predictions should get a high quality score. This score should penalize wrong predictions. Thus, if $\tilde{y}_i \neq y_i$, the system pays the price of giving a wrong prediction. The cost is a positive value defined by a loss function which describes the

loss encountered by predicting $\tilde{y}_i = h(\mathbf{x}_i)$ instead of $y_i = f(\mathbf{x}_i)$. The expectation of loss on all the possible pairs $(\mathbf{x}, y) \in X \times Y$, also known as the *real risk* [94], is defined as:

$$R_{Real}(h) = \int_{Z=X\times Y} l\big(y_i, h(\mathbf{x}_i)\big) dP(\mathbf{x}, y) \tag{2.1}$$

This value can be seen as the risk of choosing the induced hypothesis $h$ over the real (unknown) target function $f$. It can be used as a measure of distance between the induced function $h$ and the reality.

### 2.1.3 Learning systems as optimization tools

Inducing a hypothesis $h$ can be turned into an optimization problem that minimizes the real risk defined in Equation 2.1. However, the dependency $P(\mathbf{x}, y)$ between the entries and their labels is unknown. The only available information regarding $P$ comes from a limited set of observations, the training examples $S = \big\{(x_i, y_i)\big\}_{i=1}^{m}$. Thus, instead of minimizing the loss on the whole space $X \times Y$, the loss is minimized on the training data. The *empirical risk* [93] computed on the training data $S$ is defined as follows:

$$R_{Emp}(h) = \frac{1}{m} \sum_{i=1}^{m} l\big(y_i, h(\mathbf{x}_i)\big) \tag{2.2}$$

If the training data are representative enough to infer a general claim describing the world, then minimizing the loss on the empirical observations would also minimize the real loss, but this, under three conditions:

- the training data are *i.i.d.*
- the size of the training data tends to infinity
- the hypothesis space is constrained[1].

A large number of learning models induce a hypothesis by minimizing the empirical risk. Examples of such learning models are decision trees, neural networks and support vector machines.

### 2.1.4 The bias-variance tradeoff

Discriminative induction learning aims at approaching an unknown target function $f \in F$ by means of induction from a training set. In this section, we present the different factors

[1] This condition will be explained in the next section.

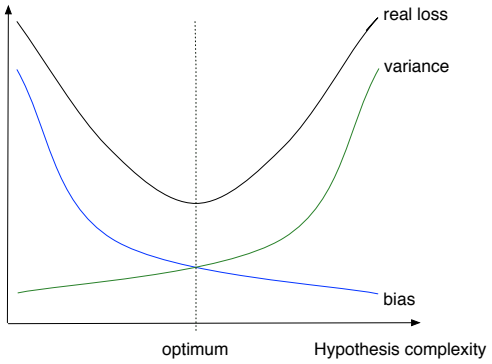


FIGURE 2.1: The bias-variance tradeoff

that play into the distance between the induced function $h$ and the real target function $f$, regarded from the real risk perspective.

### Bias

Since the target function is unknown, the space of functions $H$ that is considered by the learner may be different from the target function space $F$. As a result, even if the learner finds $h^*$, the optimal hypothesis in $H$ (the closest to $f$ using the real risk as a distance measure), $h^*$ can be far from the real target function $f$. This error which results from the difference between $H$ and $F$ is known as the *inductive bias*. For instance, suppose the target function $f$ is a quadratic polynomial while the learner space is the set of all the linear polynomials. The learner induces the hypothesis $h^* \in H$. Even if $h^*$ is the closest function to $f$ belonging to the space of linear polynomials, the choice of linear polynomials instead of quadratic polynomials introduces a bias error between $h*$ and $f$.

### Variance

In the above discussion, we supposed that the learner is able to find the optimal hypothesis $h^* \in H$. This however is not always possible. The induced hypothesis $h$ is related to the training set used in the induction process. Therefore, different training sets can produce different hypotheses. The distance between the optimal hypothesis $h^* \in H$ and the selected function $h \in H$ is known as the *variance error*. Generally, the richer the space of hypothesis $H$, the higher the variance. High variance means that minor changes in the training data can lead to major changes in the induced hypothesis.

It is clear that in order to reduce the bias error, the learner should choose a richer hypothesis space. However, a richer space $H$ leads to a higher variance. Thus, reducing the bias increases the variance and vice-versa [34, 57].

### 2.1.5 Overfitting

As we saw in the previous section, the total distance between the induced function $h$ and the real target function $f$ is a function of two errors: the bias and the variance. The bias-variance tradeoff implies finding a compromise: either decrease the bias and increase the variance or the opposite. We present here the effect of each choice.

Overfitting results from a high variance and a small bias, when the hypothesis space $H$ is very rich, often having too many parameters to adjust compared to the size of the training data. In such case, the learning system "learns by heart" the training data with a very poor capacity of generalization. Thus, the empirical loss on the training data is low while the real loss on unseen data is high. Since, in this case, the empirical loss is not representative of the real loss, the learning algorithms try to avoid overfitting by constraining their hypothesis space.

Constraining the hypothesis space increases the empirical loss. However, it insures that the empirical loss observed on the training data is representative of the real loss on test examples, leading to a better estimation capacity of the real loss.

We show in Figure 2.1 the effect of the complexity of the hypothesis space on the bias and variance errors, and as a result on the distance between the induced hypothesis $h$ and the real target function $f$. While a richer hypothesis space decreases the bias, it also increases the variance. The optimal complexity of the hypothesis space is the compromise of having a space constrained enough to allow for a good generalization, but also rich enough to have a good predictive accuracy [51, 60].

### 2.1.6 Practical evaluation measures

It is important to estimate how accurately a predictive function $h$, induced by a learning model, will perform in practice, that is to have an estimation of the real risk (see Equation 2.1). Generally, this is done using an empirical estimation of the expected error on available observations. Accordingly, the original set of observations $S = \left\{(\mathbf{x}_i, y_i)\right\}_{i=1}^{m}$ is divided into two sets:

- a training set $D$, used by the learning model to learn the function $h$
- a testing set $T$, used to validate the capacity of $h$ to generalize on test examples. The average predictive errors on the test set can be used for instance as an evaluation measure.

This validation technique is the simplest one. Nevertheless, it still requires deciding of the size of the training and testing sets. Minimizing the real loss (see Equation 2.1) requires a large number of training data to learn the predictive hypothesis. However, a large training set entails a small testing set, and thus a small number of observations that can be used to estimate the real risk in practice. In order to solve this issue, cross-validation is used as an alternative technique for learning and validation. Cross-validation repeats the training and testing procedure multiple times, each time on a different partitioning of the original set into training and testing sets. The estimated errors of each round are then averaged to produce a single estimation, which is generally more representative of the real risk than this of a single round validation procedure.

The most well-known cross-validation techniques is the *k-fold* cross-validation [56] which divides the original dataset $S$ is divided into $k$ equal size subsamples. The validation process repeats $k$ times (folds). At each round $i \leq k$ of cross-validation, the subset $i$ is kept for validation while the remaining $k-1$ sets are used for training. The advantage of this method is that all the observations are used for both training and validating. The *leave one out* cross-validation [53] is a special case of *k-fold* cross-validation with $k = m$, the size of the original dataset $S$. Thus, training and validating repeats $m$ times. This is generally used when the number of observations is very small and thus a maximum number of training data is required to induce a hypothesis. There exists other validation techniques such as the $5 \times 2$ cross-validation test of Dietterich [27], bootstrapping [71], etc...

## 2.2 Data Streaming

In many applications, the training examples are supplied continuously in time, in the form of an infinite stream. Examples of such applications are electricity management, market basket analysis, weather prediction and others. With the emergence of data streaming applications, classical learning methods were confronted with many challenges they weren't designed to handle.

### 2.2.1 Practical challenges

Data streams can be endless, making it impossible to keep the data in the working memory as with a classical learning system. Even with an endless memory, waiting for the whole training data to be collected is not possible. Thus, unlike classical systems, a data stream learning system should be able to learn incrementally with each new piece

of information received, while at the same time, being ready to predict the target label of an input at any moment, in an anytime fashion.

Data streams are also generally rapid, supplying data at a relatively high speed. With time constraints, a training example can be processed at most once, and is then discarded from the processing unit, making room to new examples.

### 2.2.2 Theoretical challenges

Classical machine learning systems not only suffer from time and memory constraints, their theoretical foundation is also shaken. Most supervised learning systems rely on the assumption that the training examples are independently and identically distributed. This condition however cannot be ensured in *real* data streams, as there is no control over the source generating the examples. In fact, in many streaming systems, subsequent examples are expected to be related and their order of appearance is not arbitrary. For instance, the images observed by a mobile agent during a navigation process are space related. In case of a system predicting flight delays, it is expected that many (timely) close flights will be delayed, for instance, due to bad weather.

Classical learning systems also consider that the world is stationary. Hence, once a classical system learns a concept, the concept remains unchanged. Generally, if the training examples are generated over a long period of time, as is the case in data streams, the underlying concept is expected to evolve, creating what is known as a concept change. For instance, user preferences evolve depending on fashion trends. Market demands evolve depending on economy conditions, and so on. As a result, if the environment changes, the predictions of the static concept will be wrong and as time goes by, more and more mistakes will be made.

### 2.2.3 Concept change

In non-stationary environments, the underlying target concept is expected to evolve with time, depending on some hidden context.

Kubat [61] gives the example of a system that learns to control the load redistribution in computer clusters where the overloaded units send part of their load to underloaded units. The rules describing the overload depend on many variables, as the CPU and memory requirements, the frequency of disk operations, and others. However, the only observed variables to the system are the lengths of the CPUs and disk queues. Thus, the workload structure is the hidden context controlling the generation of the visible

variables. The workload structure is expected to evolve with time and the same context might also reappear with time.

Consider also the example of a marketing system that learns customers preferences by observing their transactions on a website. Having a knowledge of the costumer's preferences enables suggesting specific products and promotions. Customer's buying profiles might evolve with time, depending on fashion trends, economical conditions, and the like. In a rising economy for instance, nouveaux-riches customers will buy goods or luxuries they were unable to buy before. In this case, fashion trends, economy conditions and other latent variables controlling the observed transactions are hidden to the learning system. The system can only see the transactions, with no additional knowledge of the hidden context behind the market evolution.

The changes in the hidden context induce more or less radical changes in the target concept, creating what is known as a concept change. In the general case, the learning system has no a priori knowledge about the time at which a concept changes or starts changing, nor about the severity and speed of change. It is also possible for the same context to reappear, either in a cyclic manner (seasonal variations) or in an irregular manner (inflation, market mood). In case of recurrence, the learning system should take advantage of previous experience in the learning of the current concept.

### 2.2.4 Types of concept change

If we consider that the concept is represented by the distribution of the training examples $p(\mathbf{x}, y) = p(y|\mathbf{x}) * p(\mathbf{x})$, a *concept change* happens in three cases:

- The conditional distribution $p(y|\mathbf{x})$ changes.
- The unconditional distribution $p(\mathbf{x})$ changes.
- The change involves both distributions: $p(y|\mathbf{x})$ and $p(\mathbf{x})$.

The change in the conditional distribution $p(y|\mathbf{x})$ is generally referred to as a *concept drift* [22, 101]. An example of concept drift happens in spam filtering system where the concept classifies emails into spam or non-spam depending on their content. The concept is likely to become less accurate with time since spammers try constantly to mislead the filtering systems by changing the statistical properties of the target value $p(y|\mathbf{x})$.

The change can also come from the unconditional distribution $p(\mathbf{x})$. Different terms are used to refer to this type of change: a *pseudo concept drift* [81], a *virtual concept drift* [88], a *sample selection bias* [32] or a *covariate shift* [8]. This type of change

doesn't necessarily reflect a non-stationary environment. The world can be stationary but the change in the unconditional distribution happens because the order of the received examples depends on the part of the world currently explored. In a navigation task for instance, the distribution of the images perceived by a mobile agent depends on the (timely) local part of the world currently visited. Thus, the agent encounters a change in the unconditional distribution of the images during navigation, even though the world is stable.

In most cases, the change involves both distributions: $p(y|\mathbf{x})$ and $p(\mathbf{x})$. According to [101], the source of concept change is not important from a practical point of view. The learning model capturing the target concept should be revisited anyway.

### 2.2.5 Properties of concept change

Concept changes have been described according to different criteria, mainly in terms of severity and speed. We first give an insight of the various terms used in the litterature to describe the differents types of concept change. We then report the heterogenous and mutually exclusive categories proposed by Minku et al. [69] to describe concept changes.

#### Ambiguity in litterature

Concept changes have been categorized depending on the *severity* of the change. In case of a concept drift and if the change in the conditional probability function $p(y|\mathbf{x})$ affects all the feature space, the change is said to be *global* [90]. If parts of the feature space remain stable, the concept change is called *local* [25, 90] (see Figure 2.2). In a spam filtering system for instance, new kinds of spams emerge with time. Thus, an email that would be classified as non-spam by an outdated filtering system, might be considered as a spam by an up-to-date system. This kind of change however is not global since in most cases, emails marked as spam will still be marked as spam after the change. For instance, emails containing the word "viagra".

Concept changes have also been categorized depending on their *speed*, into *gradual* or *abrupt* drifts. A sudden concept drift is called a concept shift [96, 99] in some studies, while a gradual concept drift is sometimes called a drift. As we can see, the terms used in the litterature are somehow ambiguous. While the term *drift* can be used to refer to a change in the conditional distribution $p(y|\mathbf{x})$, it is also used to refer to a gradual change. The terms are also vague. Let's take for example a concept represented by a hyperplane. The hyperplane divides the feature space into two parts and the examples in the feature space are labeled into one of two classes, according to the part they belong

to. The hyperplane moves gradually from its initial position to a final destination in the feature space, affecting the labels of the examples. It is unclear if such case represents a single gradual concept change or a series of abrupt concept changes.

### Minku's categorization

Minku et al. [69] suggested heterogenous and mutually exclusive categories to describe concept changes. The categories describe the properties of a single concept change in isolation as well as the properties of a sequence of concept changes. The properties of a concept change in isolation are: *severity* and *speed*; and the properties regarding a sequence of concept change are: *predictability*, *recurrence* and *frequency*.

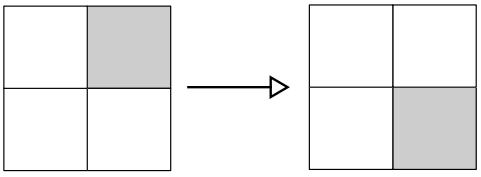

FIGURE 2.2: Example of a local drift.

### Severity

Severity can be regarded as the "amount" of change. In the litterature, no unified measure exist to quantify this amount. In [69], in case of a concept drift (the concept change is related to the conditional distribution $p(y|\mathbf{x})$), severity can be computed as the percentage of the feature space whose label is different after the drift. For example, in the drift represented by Figure 2.2, $2/4 = 50\%$ of the feature space has its target class changed. In case of a virtual concept drift (the concept change is related to the unconditional distribution $p(\mathbf{x})$), severity is the amount of change in the distribution of the feature attributes. Generally, the amount of change is estimated by the drop in the predictive performance of a learning system just after the change. If we suppose that a learning system $L$ learnt the old concept with a perfect generalization (100% classification accuracy on any test example), the amount of change can be seen as the drop in the predictive performance of $L$ when tested on examples of the new concept.

The severity has been referred to as the "extent of drift" by Widmer and Kubat [96]. The extent, as they defined it, is the dissimilarity between two concepts $A$ and $B$. It is quantified as the relative error between the two concepts i.e. the probability that $B$ will misclassify a randomly drawn example that is labeled according to $A$ (and vice-versa). This can be viewed as the probability of drawing an example from $A \oplus B$ of the two concepts, where $\oplus$ represents the symmetric difference of the sets $A$ and $B$.

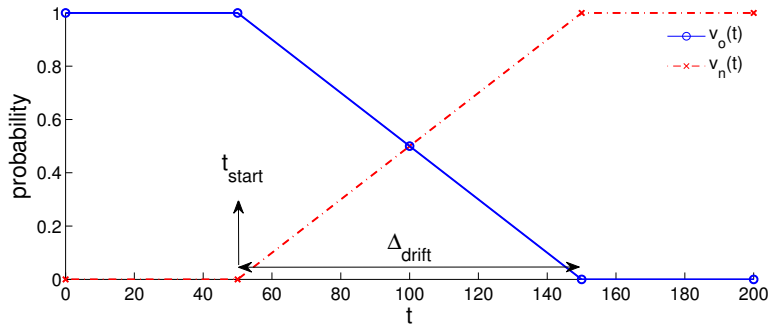


FIGURE 2.3: Example of a gradual drift with drifting time of 100 time steps. The functions $v_o(t)$ and $v_n(t)$ represent the probability that an example from the old and new concepts, respectively, will be presented.

For instance, let's consider an artificial domain defined by 6 boolean attributes $\{a_1, a_2, \ldots, a_6\}$ and the following concepts: $A \Leftrightarrow a_1 \wedge \bar{a_2}$, $B_1 \Leftrightarrow \bar{a_1} \wedge a_2$ and $B_2 \Leftrightarrow [a_1 \wedge \bar{a_2} \wedge (a_3 \vee a_4)] \vee [\bar{a_1} \wedge a_2 \wedge a_3 \wedge a_4]$. Assuming a uniform probability distribution over the feature space, the extents are: $ext(A, B_1) = 32/64 = 0.5$, $ext(A, B_2) = 8/64 = 0.125$.

### Speed

Speed is the inverse of the time required for the new concept to completly replace the old one [55]. Let's consider a data stream where at each time step, a new example is received. One concept change event occurs at time step $t_{start}$ and lasts for a period of $\Delta_{drift}$ time steps. Thus, from time step 1 to $t_{start}$, the examples are labeled according to the old concept, and starting from time step $t_{start} + \Delta_{drift}$, the examples are labeled according to the new concept. The speed of change from time step $t_{start} + 1$ and $t_{start} + \Delta_{drift}$ can be modeled by the following linear degree of dominance functions:

$$\begin{aligned} v_n(t) &= \frac{t - t_{start}}{\Delta_{drift}}, t_{start} < t \leq t_{start} + \Delta_{drift} \\ &\text{and} \\ v_o(t) &= 1 - v_n(t), t_{start} < t \leq t_{start} + \Delta_{drift} \end{aligned} \tag{2.3}$$

where $v_n(t)$ and $v_o(t)$ represent the probability that an example of the old or the new concept will be presented to the system, respectively and $t$ is the current time step. During the change, the examples are labeled according to $v_o(t)$ and $v_n(t)$ (see Figure 2.3). When $\Delta_{drift} = 1$, the change is said to be sudden otherwise it is continuous or gradual. The speed of change can be modelled by functions other than the linear function. Nevertheless, it is always assumed that during the transition between consecutive concepts, the probability that an example is labeled according to the new concept will increase, until the new concept takes completely over the old one.

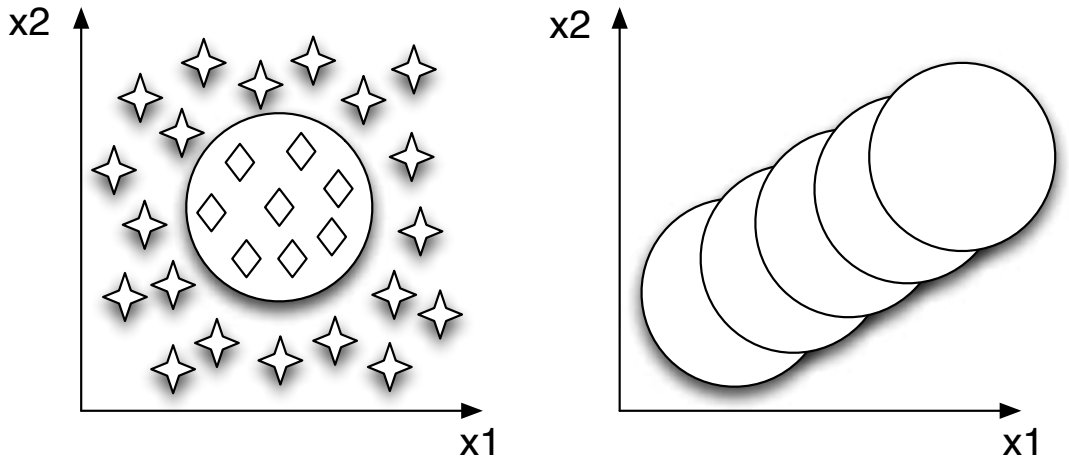


FIGURE 2.4: Left, a concept represented by a circle. The examples are classified into one of two classes: inside, outside the circle. Right, the circle moving with time, creating concept drifts.

### Predictability

The first property regarding a sequence of concept changes is predictability. Predictability refers to the ability to predict how a concept evolves with time [69]. Consider for instance a concept represented by a circle in a 2-dimensional space (see Figure 2.4). The examples are labeled into two classes: inside and outside the circle. The radius is fixed but the circle center moves, creating a concept drift. If the magnitude and direction of change are always the same, then the sequence of concept changes is predictable. There is no formal definition of predictability in case of concept changes. We consider that "predictable" refers to an underlying prediction system, that is a learning system that, taking as input information about the past history of the concept evolution, can predict its (near) future. According to Minku et al. [69], a sequence of concept changes is either predictable or random.

### Recurrence

Another property is concept recurrence [3, 37, 41, 69, 99]. Recurrence means that the same concepts might reappear with time, either in a cyclic manner (seasonal variations) or in an irregular manner (inflation, market mood). Recognizing a recurring situation allows a learning system to act pro-actively, gaining precious time and avoiding costly incorrect predictions. Therefore, it is important that the learning system takes advantage of its previous experience in its predictions of target values when an old concept reappears.

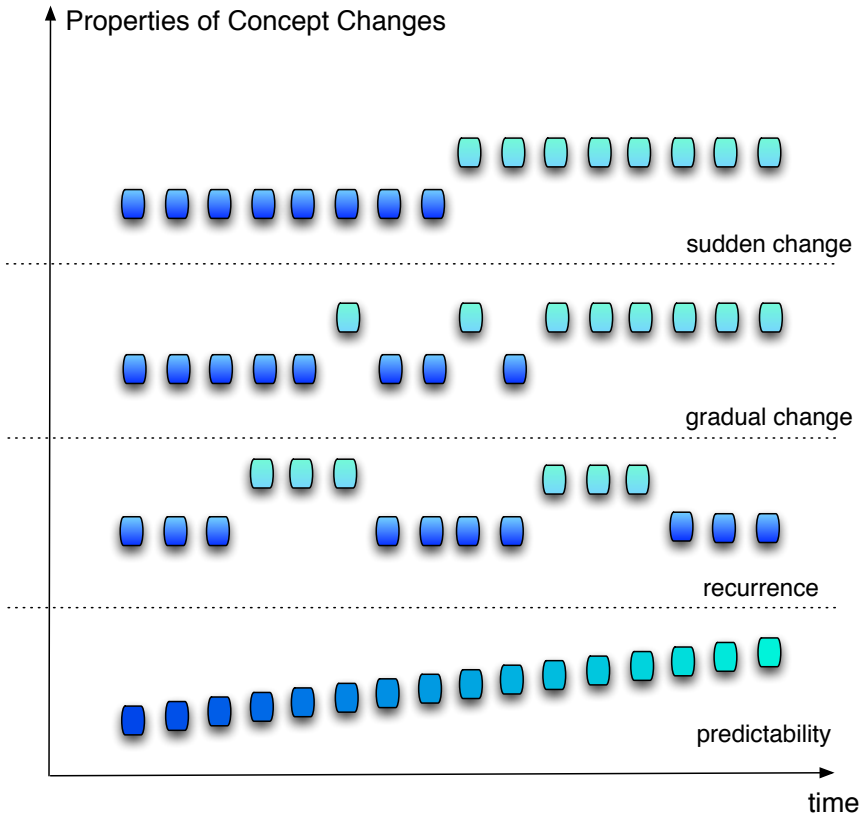


FIGURE 2.5: Different properties of concept changes. The x -axis represents time while the y-axis shows differents concepts (according to the concept color).

#### Frequency

The third and last property regarding sequence of changes is frequency [62, 69]. Frequency can be measured as the inverse of the number of time steps between two consecutive concept changes. According to this criterion, concept changes can either be periodic or non-periodic. It is important to note that frequency and recurrence are not the same. Concepts can change every $t$ time steps but without repeating.

Different properties of concept changes are illustrated in Figure 2.5.

### 2.2.6 The stability-plasticity dilemma

It is not possible from a practical point of view to store all the examples received in a data stream. Only a summary of examples can be kept. And under the assumption of possible concept changes, it is important to ensure that the summary reflects the underlying target concept. In other words, the summary shouldn't include examples belonging to an old concept, as obsolete examples can be misleading and even harmful to the learning system. A simple solution is to trust the recently received examples.

Thus, at each time step, a window containing the $n$ most recent examples can be used to learn the current concept. However, the size of the window, $n$, remains to be defined. In fact, if the concept is *stable*, a large window of training examples allows the system to learn more precisely the current target concept. However, if the concept is changing, the window should be small, excluding outdated data. This is known as the *stability-plasticity dilemma* [31, 64]. While a plastic learning system with a small window size adapts rapidly to changes, a stable learner with a large window size is more reliable in periods of stability.

### 2.2.7 Adaptation and anticipation

Predicting in an environment with possible evolving states can be handled with two non-contradictory and potentially simultaneous approaches: *adaptation* and *anticipation*.

*Adaptation* follows new trends, with no insight to the future. An adaptative approach would incorporate new training examples into the learning memory when the concept is stable, and would reset the learning memory when the concept changes.

*Anticipation*'s concern, on the other side, is to understand and characterize the evolution of the environment in order to predict upcoming trends. While adaptative approaches adapt passively to changes, anticipation acts pro-actively, trying to be aware of what might happen in the near future, preparing prediction strategies in advance. Studying changes in the environment requires keeping memory of the past. From a practical point of view, it is not possible to retain all the received training examples in the memory. Hence, a compact representation should be considered. For instance, in the work we will present, we suggest an anticipative approach that keeps the history of all encountered concepts during the data streaming. Given the sequence of encountered concepts $C_{seq} = \big(C_1, C_2, \ldots, C_k\big)$, a pure anticipative approch may operate as follows:

- The upcoming concept $\tilde{C}_{k+1}$ is predicted by analyzing the sequence of previously encountered concepts. The predicted concept $\tilde{C}_{k+1}$ is then used to predict the labels of data stream examples when $C_k$ changes.
- The upcoming concept $\tilde{C}_{k+1}$ is assumed to be a recurring concept i.e. $\tilde{C}_{k+1} \subset C_{seq}$. Thus, when $C_k$ changes, $C_{seq}$ is scanned and the concept that fits the recent examples most is used for prediction.

Adaptation and Anticipation can either operate independently, or simultaneously. A disadvantage of pure adaptative approaches is their unability to take advantage of their

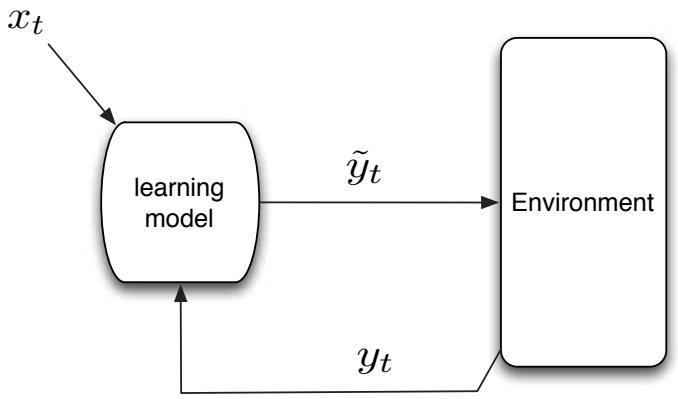


FIGURE 2.6: The workflow of an online learning system

previous learning experience. If for instance, the same concept reappears, a pure adaptative approach would learn the concept from scratch. By contrast, a pure anticipative approach would fail to predict correctly the target values of data stream examples if the predicted concept turns out to be different from the real underlying target concept.

Generally, anticipation doesn't come without adaptation. Anticipation can be seen as a second-order learning that makes use of adaptative learning systems to get the list of the encountered concepts. In other words, it is by means of adaptation that the anticipation mechanism has knowledge of the different concepts. When combining both approaches, anticipation is used first to predict the upcoming concept. If the anticipation fails in its prediction, the adaptative approach takes over, adapting to the new concept.

## 2.3 Online Machine Learning

Online learning systems have been suggested to deal with the challenges introduced by data stream applications [36]. Unlike classical learning models, an online learning model is trained incrementally with each available new piece of information. Not only is the induced hypothesis constantly updated, it is also constantly used to predict the target values of the stream examples. In stock markets for instance, forecasters predict the answer to each new incoming incentive, possibly in order to regulate the stocks, even before the outcome is known. Managers of electrical grids also try to predict the continuous power demand in order to optimize the functioning of electrical plants. Stream learning applications include ATM transactions analysis, the prediction of web search patterns, weather prediction, fraud detection, and others.

### 2.3.1 Scenario

Online learning proceeds in a sequence of trials or time steps [11–13, 85] (see Figure 2.6). At each time step $t$, the online learning system repeats the following steps:

1. The learning system receives an input $\mathbf{x}_t$.
2. The learning system predicts the target value $\tilde{y}_t$ of $\mathbf{x}_t$.
3. The environment reveals the real target value $y_t$ to the learning system.
4. The learning system updates its hypothesis based on the training example $(\mathbf{x}_t, y_t)$.

### 2.3.2 Practical evaluation measures

Classical machine learning methods divide the available data into two sets: the training and the test set. The training set is used to learn the concept while the testing set is used to evaluate the accuracy of the learning system. In online learning scenarios, there are no such distinctions as data are continuously generated. We present here three main evaluation measures used to assess the predictive performance of online learning methods: the online performance, the time sliding windows and the fading factors.

#### Online performance

Most online approaches use the *online predictive performance* [38] as an evaluation measure for a learning system. The online performance is a continuous function representing the accuracy of the online learning method at each time step.

More formally, the learning system predicts at each time step $t$ the label $\tilde{y}_t$ of the training example $\mathbf{x}_t$ before the example is learnt. The prediction error is between the prediction $\tilde{y}_t$ and the real value $y_t$ is calculated and the average prediction error from time step 1 to $t$ is updated. The average prediction error, also known as the *online predictive error* is computed iteratively as follows:

$$oe(t) = oe(t-1) + \frac{err(\mathbf{x_t}) - oe(t-1)}{t} \tag{2.4}$$

where

$$err(\mathbf{x_t}) = \begin{cases} 0 & \text{if } \tilde{y}_t = y_t \\ 1 & \text{if } \tilde{y}_t \neq y_t \end{cases} \tag{2.5}$$

The online predictive performance or accuracy ($acc$) and the online error ($oe$) are related by the following equation: $acc(t) = 1 - oe(t)$.

The online performance is also called the *progressive evaluation* [90], the *prequential accuracy* [10, 40] or the *interleaved-test-then-train* [13] method.

### Time windows

In the presence of concept drift, it is also possible to evaluate the predictive accuracy of the online learning method on time sliding windows over the data stream. Accordingly, the error $we(t)$ at time step $t$ over a sliding window of size $w$ is computed as follows [40]:

$$we(t) = \frac{\sum_{i=0}^{w-1} err(\mathbf{x_{t-i}})}{w} \tag{2.6}$$

The accuracy ($acc$) of the predictive system and the computed error ($we$) are related by the following equation: $acc(t) = 1 - we(t)$.

This method reflects the recent performance of the predictive system. This is advantageous in case of an evolving environment since it allows to observe the reactivity of the system towards a concept change. Nevertheless, time sliding windows require to keep in memory the errors over the recent examples.

### Fading factors

Another technique that exhibits the recent predictive performance weights previous errors using a decay factor [40]. Accordingly, the error $fe(t)$ at time step $t$ using fading factors can be computed as:

$$fe(t) = \frac{S(t)}{N} \tag{2.7}$$

where $S$ represents a cumulative loss function that can be computed recursively as follows: $S(1) = l(1)$ and $S(t) = l(t) + \alpha \times S(t-1)$, with $\alpha < 1$ and $l(t)$ representing the loss at time step $t$. The error value $fe(t)$ converges to 0 when $N$ tends to $\infty$. Gama et al. [40] suggested a correction factor that turns Equation 2.7 into:

$$fe(t) = \frac{S(t)}{B(t)} = \frac{l(t) + \alpha \times S_{t-1}}{n(t) + \alpha \times B_{t-1}}$$

where $n_t$ is the number of examples used to compute $l(t)$. For instance, when the loss function $l$ is the the zero-one loss (see Equation 2.5), the loss is computed for every single example and $n(t) = 1$.

Like time sliding windows, this evaluation technique gives more importance to the recent predictive performance of the online learning method. In addition, it is memoryless as it doesn't need to keep old traces of the system's errors.

### 2.3.3 The theory of online learning

In online learning, the order of the examples is generally not random as consecutive training examples can be related. For instance, the spatio-temporal correlations of video data, the time-dependent electricity consumption, the time-correlated rate of subway rides (peak, off-peak), and others. Thus, training data are *not* assumed to be independent. And since the environment generating the streaming data may change over time, allowing for concept changes, the training data are also *not* identically distributed. Hence, the *i.i.d.* property does not hold. As a result, online learning methods cannot use the real loss measure (see Equation 2.1) as a performance criterion. Since the world is not stationary, there are no stable target function $f$ that the learning system tries to approach. Using the empirical loss (see Equation 2.2) as an optimization criterion is also not possible since the training data are not indenpendently and identically distributed.

Since there exist no stable target function $f$, the quality of an induced hypothesis $h$ cannot be measured by its distance from $f$ as in Equation 2.1. Thus, instead of comparing the induced hypothesis $h$ with a target function $f$, the hypothesis is compared with any other candidate hypothesis. The notion of loss used in classical learning theory is replaced by the notion of *regret* which in this context denotes the regret of choosing a function $h$ instead of any other possible function $h'$ when predicting a target value. Hence, the theoretical framework assumes the existence of an ensemble of hypotheses, and a forecaster which, after getting the individual predictions of the hypotheses, gives its final prediction. The goal is to minimize the cumulative regret of the forecaster with respect to each hypothesis in the ensemble. The theoretical framework is presented more formally in the next section.

#### The theoretical framework

Cesa-Bianchi [20] studied the theory of online machine learning, using the model of prediction with expert advice. This model provides the foundation to the theory of prediction of individual sequences.

It is supposed that a forecaster predicts an unknown sequence of bits i.e. $\tilde{y}_t \in \{0, 1\}$. To compute $\tilde{y}_t$, the forecaster listens to the advice of $N$ experts. Experts are seen as reference forecasters, black boxes of unknown computational power, possibly different

learning models (neural networks, decision trees, and others). No a priori assumptions are made regarding the experts. They can be anything, from induced hypotheses learnt by means of learning models, to random forecasters providing random predictions. Take for instance an advisory committee on Immunization Practices. The committee is formed of experts that vote on proposed influenza vaccine recommendations. The forecaster in such case combines the votes of the different experts to make a final recommendation regarding the vaccine. While the experts in our example are specialized members, most likely Professors of Medicine, the forecaster however is not required to hold a scientific position, as its function here is to simply count the votes.

In the theoretical framework, the forecaster computes his predictions in an online fashion and his predictive performance is compared to that of experts. The forecaster's goal is to keep as small as possible the cumulative regret with respect to each expert[2]. This quantity is defined for expert $E$, by the sum:

$$R_{E,n} = \sum_{t=1}^{n} \big(l(\tilde{y}_t - y_t) - l(f_{E,t} - y_t)\big) = L_n - L_{E,n}$$

where $n$ is the length of the sequence of predictions, $\tilde{y}_t$ and $f_{E,t}$ are the predictions of the forecaster and the expert $E$ at time step $t$, respectively. The $l$ function represents a loss function which computes the "score" of a particular prediction. For instance, the zero-one loss function returns one if the prediction is wrong and zero, otherwise. In the above equation, $L_n = \sum_{t=1}^{n} l(\tilde{y}_t - y_t)$ is the forecaster's cumulative loss while $L_{E,n} = \sum_{t=1}^{n} l(f_{E,t} - y_t)$ is the expert's cumulative loss. Hence, $R_{E,n}$ is the difference between the forecaster's total loss and that of the expert $E$ after $n$ prediction rounds.

The interest of the theoretical research is to bound the cumulative regret of a forecaster with respect to the expert with the lower number of mistakes. To give an insight to the theoretical study, we report two basic examples introduced by Cesa-Bianchi [20].

### A simple case

Cesa-Bianchi [20] starts with a simple case, where the forecaster knows in advance that there is some expert $i$ that makes no mistakes. The index $i$ of this expert however is unknown. The forecaster assigns a weight to each expert, a value reflecting the possibility of this expert to be the best expert $i$, yet unknown to the forecaster. At the beginning, all of the experts are potential candidates and are all assigned a weight value of 1. When an expert makes a mistake, its weight is lowered to zero. At every time step $t$, the forecaster

[2] or to a combination of the experts' predictions.

predicts $\tilde{y}_t$ using a weighted vote. Thus, $\tilde{y}_t = 1$ if and only if the number of experts $j$ with a weight value of one and with $f_{j,t} = 1$ is bigger than those with a weight value of one and with $f_{j,t} = 0$. The weighted vote has the effect of eliminating the experts that made at least one mistake from the forecaster's final prediction $\tilde{y}_t$. Only experts with a 100% predictive accuracy so far participate in the prediction process. In the presented scenario, the number of mistakes made by the forecaster is at most $\lfloor \log_2 N \rfloor$. To prove this upper bound, let $W_m$ be the sum of the weights of the experts when the forecaster made his $m$-th mistake. Initially, $m = 0$ and $W_0 = N$, the number of experts. When the forecaster makes a mistake, this means that at least half of the experts with a weight value of one were mistaken, and as a result, their weights have been lowered to zero. Thus $W_m \leq W_{m-1}/2$. By recursive calculation, we get $W_m \leq W_0/2^m$. Since expert $i$ makes no mistake, this implies that $W_m \geq 1$ at all times. Solving the equation $1 \leq N/2^m$ for $m$ gives the bound $m \leq \lfloor \log_2 N \rfloor$

**A more general case**

The same spirit of analysis can be transfered to a more general case where we don't suppose the presence of an expert with perfect predictions. The goal here is to bound the number of mistakes made by the forecaster compared to this of the expert with the minimum number of errors a posteriori. The forecaster is the same as the previous one with one difference. Since all the experts might make mistakes, a mistaken expert shouldn't be eliminated. Thus, if an expert makes a mistake, its weight is not set to zero. In a multiplicative update framework, the expert's weight is decreased by multiplying its value by a constant $\beta$, where $0 < \beta < 1$. When the forecaster makes its $m$-th mistake, at least half of the total weight is multiplied by $\beta$ and the other half remain as is. Thus, $W_m \leq W_{m-1}/2 + \beta W_{m-1}/2$. Since this relation holds for all $m \geq 1$, we get $W_m \leq W_0(1+\beta)^m/2^m$. Let $E_k$ be the expert with the fewest mistakes when the forecaster made its $m$-th mistake. The weight of $E_k$ is $w_k = \beta^{m^*}$ where $m^*$ is $E_k$'s number of mistakes. Thus we have:

$$\begin{aligned} \beta^{m^*} &\leq W_m \\ &\leq W_0(1+\beta)^m/2^m \end{aligned} \tag{2.8}$$

We then get the final bound,

$$m \leq \left\lfloor \frac{\log_2 N + m^* \log_2(1/\beta)}{\log_2 \frac{2}{1+\beta}} \right\rfloor$$

This bound shows a dependency between the number of mistakes made by the forecaster compared to this made by the best expert, after a number of predictions.

#### Critical point of view

Other bounds were shown in the work of Cesa-Bianchi [20], and this on various types of forecasters with different combinations of the experts predictions (polynomially weighted average forecaster, exponentially weighted average forecaster, etc...). The advantage of the extracted bounds is that they apply on any sequence of predictions and regardless of the type of experts (learning models). However, by considering experts as black-box components, the theory fails to provide proofs on how *intelligent* the prediction system is, relative to its ability of capturing regularities in the environment. The experts on which the forecaster has to rely, may be dumb, for instance, always predicting the last observed class regardless of the input $\mathbf{x}_t$, that is, $\tilde{y}_t = y_{t-1}$. In case of a time-correlated environment where subsequent examples are likely to have the same class labels, the former experts will have a high predictive accuracy. A question is then whether it is possible to minimize the forecaster's predictive error relative to the best expert but without omitting the intelligence aspects of the prediction system. Another limitation of this work is that, since the research has been conducted on very general cases, theoretical proofs cannot be used to evaluate and compare the performance of different algorithms in specific situations, for instance, in case of severe drifts, gradual evolutions of the environment, recurring contexts, and the like. According to [86], the extracted bounds are rather loose and uninteresting in practice.

The task of *learning drifting concepts* has been studied from a theoretical point of view. Generally, some restrictions are imposed on the type of admissible concept change, such as the rate of change [63] or the severity of change [46]. The main disadvantage is that the special cases studied don't usually occur. The theoretical bounds can also include large sizes of training data that would be impractical to employ [91].

### 2.3.4 Online learning in practice

Finding theoretical garanties of an online learning algorithm is always an advantage. However, without practical applicability, an online learning system is useless. We discuss the practical requirements that should be taken into consideration when evaluating online learning systems. We then present the main categories of existing systems.

#### 2.3.4.1 Practical requirements

**Time and memory constraints** Online learning systems should take into consideration time and memory constraints. Thus, both prediction and training time should be bounded, allowing for a real-time processing.

**Adapting to changes** Online learning systems should also handle drifting concepts by adapting to new trends. One central concern here is to optimize a tradeoff between learning from as much data as possible, in order to get the most precise prediction model, while at the same time recognizing when data points become obsolete and potentially misleading, impeding the adaptation to new trends (the stability-plasticity dilemma). Note that the detection of concept changes should be *fast*. Thus, the learning system should be highly sensitive to possible changes in the environment while at the same time being robust to noise and false alarms.

**Knowledge transfer** The variety of concept speed and severity adds more challenges, specially the need for *knowledge transfer* i.e. the ability to classify examples belonging to the old concept when the concept evolves. Knowledge transfer is useful in two cases. First, during the drifting time in case of a gradual drift since examples are generated for both the old and the new concept. Secondly, in case of a non severe concept change where knowledge acquired from the learning of the old concept can help classify examples from the new one.

**Anticipating changes** Being able to recognize changes in the environment and to anticipate them requires some kind of second-order or meta learning. The learning system needs to be able to analyze and reflect upon its past experiences and responses and to decide what is the best course of action or decision given the past. Very few learning systems are anticipative. Most approaches are purely adaptative, adapting passively to changes with no insight of the future.

#### 2.3.4.2 Existing approaches

Online machine learning in the presence of concept drifts is rather a recent research area. Nevertheless, a large number of online learning systems have been proposed to deal with evolving environments. The first systems capable of handling concept drifts are STAGGER (1986) [83], IB3 (1991) [1] and FLORA (1996) [96]. To the best of our knowledge, the existing systems have all empirical evidence, and few have theoretical foundation. Depending on their way of handling drifting concepts, existing systems can be divided into three main categories: *instance selection*, *instance weighting* and *ensembles of classifiers*.

**Instance selection** The earliest techniques used instance selection to deal with the stability-plasticity dilemma. Instance selection methods aim at selecting the examples that are relevant to the current concept.

The most common instance selection techniques are *windowing techniques* which use time sliding windows over the data stream. At each time step, the concept is learnt based on the last $n$ observed training examples, where the window size $n$ can either be fixed or adjusted with time using heuristics [96]. When the window size is fixed, selecting a "good" window size requires an apriori knowledge about the change. An algorithm that fixes the window size also assumes that the environment behaves the same way all the time, a condition that is not met in the real world. Adaptative window sizes were a solution to this problem. The idea behind dynamic window sizes, is to decide to when a concept is changing. Once the change is detected, the algorithm can then change the size of the window to more accurately represent the current concept. The FLORA methods by Widmer and Kubat [96] are probably the most known amongst the time sliding techniques. The FLORA methods with adaptative windows enlarge the window size as long as the concept is stable. When a change is detected, the window size is shrinked accordingly. The FLORA systems are limited to handle symbolic concepts represented using an attribute-value logic language. For instance, a concept with the following description: (color = white and temperature = low).

Instance-based learning has also been suggested to deal with concept drifts. Instance-based learning predicts the label of an incoming example based on its similarity to one (or more) stored training example(s). Similar examples are assumed to have similar labels or target values. IB3 [1] is the first instance-based learning technique capable of handling concept drifts. IB3 describes a concept by a set of instances. For each instance, IB3 calculates the percentage of its correct classification attempts and compares it with its class's frequency to determine which training examples to keep and which examples are outdated and should be discarded [91].

**Instance weighting** Example weighting make use of a weighting scheme over past examples, taking advantage of the ability of some learning algorithms such as Support Vector Machines (SVMs) [81] to process weighted examples. Examples can be weighted depending on their age or relevance regarding the current concept. Generally, weighting functions give less weight to past examples as it is supposed that recent data are more representative of the current concept than old examples. Windowing and weighting techniques imply the choice of thresholds, decay factors and so on [54]. The adaptation to changes in the former two approaches requires an explicit change detection mechanism which should be highly sensitive to changes while at the same time being able to distinguish between real concept changes and noise. Sensitivity and robustness generally require opposite strategies. While robustness entail analyzing large amount of data to detect outliers, sensitivity to change requires a fast response to any unusual event.

**Ensemble of classifiers** Several online ensembles methods have been proposed for tackling changing concepts [12, 13, 59, 70, 84, 85, 90]. Ensemble methods have the advantage of holding diverse learners (i.e. concept descriptions) in the ensemble. The predictions of the learners are generally combined using simple voting, weighted voting or the most relevant concept description is selected for prediction. For instance, the first concept drift handling system STAGGER [83] maintains a set of concept descriptions, and more complicated concept descriptions are then produced iteratively, the best of which are selected according to their relevance to the current data.

Keeping a diverse ensemble of learners has advantages over the use of one single learner. According to [69], the diversity helps reduce the initial drop in accuracy that happens just after the change. When the concept is stable, however, low diversity in the ensemble gives more accurate results. In many ensemble approaches, learners are removed from the ensemble when their performance drops under a threshold and are then replaced by new learners with a small window size. Expulsing an learner from the ensemble can be the result of a concept change which makes its training window unadapted to the current situation. Thus, ensembles of classifiers are not necessarly embedded with an explicit concept change detection system. Adapting to concept changes can happen *implicitly* by removing and adding new members to the ensemble. Another advantage of ensemble methods is that the adaptation process may result in smoother changes in the ensemble (addition and/or removal of some members) compared to the rigid changes of approaches that use one single learner.

### 2.3.5 Online learning datasets

In order to evaluate online learning systems in the presence of drifting concepts, researchers usually generate artificial datasets with controlled concept drifts or make use of private business data. The advantage of artificial datasets is the ability of simulating various types of concept changes, with different speeds and severities. This allows one to study the behavior of online learning algorithms in different circumstances. The real datasets, by contrast, represent real-life scenarios. The main problem is that, with real datasets, it is often not possible to know for sure whether a drift occured or not. As a result, explaining the behavior of a learning algorithm during the data streaming is not as easy as with controlled datasets. Nevertheless, it is expected that, if the examples are collected over a long period of time, concept drifts occur, due to seasonal changes, economic conditions, market evolution and others. Virtual concept drifts can also happen as the result of a sampling bias.

In this section, we cover the majority of the datasets that were used to evaluate the predictive accuracy of online learning systems. Three types of datasets are presented: artificial, real, and semi-artificial. Semi-artificial datasets are real datasets that were modified to simulate concept drifts. Hence, the examples come from real-world applications but the labels were modified to simulate drifting concepts.

It is important to note that some of the real data streams presented here are not expected to be undergoing concept drifts. They were nevertheless used in our experimental evaluations to assess the predictive ability of drift handling methods under stationary conditions.

#### 2.3.5.1 Artificial datasets

**STAGGER** Originally introduced by Schlimmer and Granger in 1986, the STAGGER problem [83] represents the sequence of the following three concepts: $A \Leftrightarrow size = small \wedge color = red$, $B \Leftrightarrow color = green \wedge shape = circular$ and $C \Leftrightarrow size = medium \vee large$. The dataset contains 120 training examples chosen uniformly from the feature space. The first 40 are labeled according to concept $A$, the second 40 according to concept $B$, and the last 40 according to concept $C$.

**FLORA** Widmer and Kubat [96] introduced in 1996 two datasets, with moderate and slow speeds of change, in order to evaluate the different versions of their online learning system FLORA. Two concepts are defined over 6 boolean attributes $\{a_1, ..., a_6\}$: $A \Leftrightarrow a_1 \wedge a_2$ and $B \Leftrightarrow (a_3 \wedge a_4) \vee (a_5 \wedge a_6)$. Concept $A$ gradually drifts towards concept $B$ at time step 100 with a drifting period of 100 and 200 time steps for the moderate and slow speeds, respectively. Each dataset has a total of 500 training examples.

**SEA** The SEA dataset was proposed by Street and Kim in 2001 [87]. It represents a sequence of four different concepts, with 15,000 examples each, for a total of 60,000 examples. Each example has three attributes. Attributes are numeric between 0 and 10, and only the first two are relevant. The classification is done using $x_1 + x_2 \leq \theta$, where $x_1$ and $x_2$ are the first two attributes and $\theta$ is a threshold value between the two classes. The value of $\theta$ is set to 8, 9, 7 and 9.5 for the four blocks of examples. In each block, 10% class noise was inserted (10% of the examples had their real class value inverted).

TABLE 2.1: Minku's artificial problems

| Problem | Fixed Values | Before→After Drift | Severity | 2N |
|---|---|---|---|---|
| Circle | $a = b = 0.5$ | $r = 0.2 \rightarrow 0.3$ | $\approx 16\%$ | 2000 |
| | $\leq$ | $r = 0.2 \rightarrow 0.4$ | $\approx 38\%$ | |
| | | $r = 0.2 \rightarrow 0.5$ | $\approx 66\%$ | |
| SineV | $a = b = 1$ | $d = -2 \rightarrow 1$ | 15% | 2000 |
| | $c = 0$ | $d = -5 \rightarrow 4$ | 45% | |
| | $\leq$ | $d = -8 \rightarrow 7$ | 75% | |
| SineH | $a = d = 5$ | $c = 0 \rightarrow -\pi/4$ | $\approx 36\%$ | 2000 |
| | $b = 1$ | $c = 0 \rightarrow -\pi/2$ | $\approx 57\%$ | |
| | $\leq$ | $c = 0 \rightarrow -\pi$ | $\approx 80\%$ | |
| Line | $a_1 = 0.1$ | $a_0 = -0.4 \rightarrow 0.55$ | 15% | 2000 |
| | $\leq$ | $a_0 = -0.25 \rightarrow -0.7$ | 45% | |
| | | $a_0 = -0.1 \rightarrow -0.8$ | 70% | |
| Plane | $a_1 = a_2 = 0.1$ | $a_0 = -2 \rightarrow -2.7$ | 14% | 1000 |
| | $\leq$ | $a_0 = -1 \rightarrow -3.2$ | 44% | |
| | | $a_0 = -0.7 \rightarrow -4.4$ | 74% | |
| Bool | $c = S \vee M \vee L$ | $a = R, \wedge_1$ $b = R \rightarrow R \vee T$ | $\approx 11\%$ | 1000 |
| | $\wedge_2$ | $a = R, b = R,$ $\vee_1 \rightarrow \wedge_1$ | $\approx 44\%$ | |
| | $=_1 =_2 =_3$ | $a = R \rightarrow R \vee G,$ $b = R \rightarrow R \vee T,$ $\vee_1 \rightarrow \wedge_1$ | $\approx 67\%$ | |

**Minku's artificial problems** A total of 54 datasets were generated by Minku et al. in 2010, simulating different types of concept changes on the following problems [69]:

- Circle: $(x - a)^2 + (y - b)^2 \leq r^2$
- Sine wave: $y \leq a\ sin(bx + c) + d.$
- Moving plane: $y \leq -a_0 + a_1 x_1 + a_2 x_2$
- Moving line: $y \leq -a_0 + a_1 x_1$
- Boolean: $y = \big(color\ eq_1\ a\ op_1\ shape\ eq_2\ b\big)\ op_2\ size\ eq_3\ c$

The parameters, $a$, $b$, $c$, $d$, $r$, $a_0$, $a_1$, $a_2$, $eq$ and $op$ can assume different values to define different concepts; $eq$ represents $=$ or $\neq$ and $op$ represents the logical connective $\vee$ or $\wedge$.

A concept change event is simulated by changing one or more of the problems' parameters, creating a change in the conditional probability function $p(y|\mathbf{x})$ (see Table 2.1). The Sine wave problem is divided into two subproblems: *SineH* and *SineV* depending on whether the sine wave moves horizontally or vertically after the concept change, respectively.

For each problem, different datasets were generated. A dataset contains one concept change event and different types of changes are simulated by varying the amount of change severity and speed. The severity here represents the percentage of the feature space which has its target class change after the concept change. Speed is the inverse of the time required for the new concept to completly replace the old one. The speed is modeled by the linear degree of dominance functions described in Equation 2.3. A concept change starts at time step $N$ where $2N$ is the total number of training examples and the concept change lasts for $\Delta_{drift}$ time steps. The drifting time varies among 1, $0.25N$, and $0.50N$ time steps and the degree of severity in the datasets vary from low to medium to high.

With three levels of severity and three speeds, nine datasets are created for each problem (for more details, refer to [69]). For *Plane* and *Boolean*, $N = 500$ and the examples are normally distributed through the whole feature space. For the other problems, $N = 1000$ and the number of examples belonging to class 1 and 0 is always the same, having the effect of changing the unconditional probability distribution function $p(\mathbf{x})$ when the drift occurs. Eight irrelevant attributes and 10% class noise were introduced in the *Plane* datasets.

#### 2.3.5.2 Semi-artificial datasets

**CAR** Minku et al. [69] created a partially artificial dataset based on the real-world database CAR, from the UCI Machine Learning Repository [5], by introducing simulated drifts inspired by Scholz and Klinkenberg [84]. The CAR database was randomized and divided into three partitions in order to simulate two drifts. Each partition has size 576, but only 75% of the examples were retained for training, giving a total of 1,296 examples. The drifting time was 1 and 288 for the first and second drift, respectively and the speed of change was modeled by linear degree of dominance functions (see Equation 2.3). The different concepts were created by changing the labels of the target classes of the examples according to Table 2.2. As it can be observed from the table, all of the class labels are different after the first drift. The second drift presents a partial return to the first concept, with the first two classes getting their original labels back.

**IRIS** Minku et al. [69] also simulated artificial drifts using the real-world database IRIS from the UCI Machine Learning Repository [5]. The IRIS database was replicated 3 times in order to create three partitions and each partition was randomized. Each partition has size 150, but only 75% of the examples were retained as training data, the remaining examples being kept for testing purposes. Accordingly, the data stream has a total of 339 examples. The drifting time was 1 and 75 for the first and second

| Probl. | Original class | $C_1$ | $C_2$ | $C_3$ |
|---|---|---|---|---|
| IRIS | Setosa 33.33 % | 1 | 4 | 4 |
| | Versicolour 33.33 % | 2 | 1 | 2 |
| | Virginica 33.33 % | 3 | 3 | 3 |
| | 0 % | 4 | 2 | 1 |
| CAR | Unacc 70.02% | 0 | 1 | 0 |
| | Acc 22.22% | 1 | 0 | 1 |
| | Good 3.99% | 1 | 0 | 0 |
| | Very good 3.76% | 1 | 0 | 0 |

TABLE 2.2: IRIS and CAR semi-artificial datasets. Rounded percentage of examples of each class in the original database and concepts ($C_i$) used to create the drifting datasets. For instance, the class *Versicolour* has its label changed from 2 to 1 then back to 2.

drift, respectively and the speed of change was modeled by linear degree of dominance functions (see Equation 2.3). The different concepts were created by changing the labels of the target classes of the examples according to Table 2.2.

**USENET** This dataset simulates a news filtering system with the presence of concept drifts relative to the change of interest of a user over time [74]. The dataset contains 5,931 examples representing documents collected from the 20 Newsgroups [5]. A document is represented as a bag of 658 words. Attributes are binary values indicating the absence or the presence of the respective word, while the document's label indicates whether the document interests a virtual user or not. Virtual concept drifts are simulated by changing the interest of the virtual user as follows. Six different topics exist, each with a corresponding mailing list. The simulated user is subscribed to four mailing lists. Over time the virtual user decides to unsubscribe from two mailing lists and subscribes to two new ones, and so on.

**ELIST** The emailing list dataset [52] simulates a stream of email messages from different topics that are sequentially presented to a user who then labels them as interesting or junk according to his/her personal interests. The email stream are collected messages from usenet posts that exist in the 20 Newsgroup collection [5]. The stream contains 1,500 examples with 913 attributes (boolean bag-of-words representation), and represents two distincts contexts. In the first context, the user is only interested in messages related to medicine. In the second one, the user's interest switches to space and baseball. The stream is the sequence $C_1, C_2, C_1, C_2, C_1$ where $C_1$ and $C_2$ are sequences of 300 email messages, labeled according to the first and second context respectively.

#### 2.3.5.3 Real datasets

**Forest-Covertype** This dataset contains the forest covertype for 30*30 meter cells obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. The dataset contains 581,012 examples described by 54 attributes. The task consists of predicting cartographic observations into one of 7 possible cover type designations. The Forest-Covertype has been used in several papers on data stream classification [39, 76].

**PAKDD** The PAKDD 2009 dataset comprises data for a credit assessment application [24]. The dataset comes from a private label credit card operation of a major Brazilian retail chain, along stable inflation condition. The training data, which contains 50,000 examples, corresponds to a 1 year period. Each example corresponds to a client and contains 27 input attributes, such as sex, age, marital status, profession, income, etc. The target class identifies if the client is *fraudulent* or *good*, assigning a class label of 1 (positive class) and 0 (negative class), respectively. The class *fraudulent* is a minority class, composing 19.7% of the data. This dataset is supposed to contain no drift. However, since *fraudulent* is a minority class, this raises the challenge of obtaining a low false negative rate (a low number of times a *fraudulent* client is considered as a *good* client) in order to avoid fraud.

**KDD** The KDD'99 cup dataset is a network intrusion detection dataset [89]. The task consists of learning a network intrusion detector capable of distinguishing between bad connections or attacks and good or normal connections. The training data consists of 494,021 examples. Each example represents a connection described by a vector of 41 features which contain both categorical (ex: the type of protocol, the network service) and continuous values (ex: the length of the connection, its duration). The class label is either 0 or 1 for normal and bad connection, respectively.

**Electricity** The electricity dataset was collected from the Australian New South Wales Electricity Market [44]. In this market, the prices are not fixed and may be affected by demand and supply. The dataset contains 45,312 examples, dated from May 1996 to December 1998, which record electricity prices at 30 minutes interval. Each example contains 8 input attributes (the date, time stamp, day of the week and 2 electricity demand values, 2 electricity supply values, and scheduled electricity transfer between states.) and the target class identifies the change of the price (UP or DOWN) relative to a moving average of the last 24 hours.

**Airlines** The Airlines task [23] is to predict whether a given flight will be delayed, given the information of the scheduled departure. The dataset contains 539,383 examples. Each example is represented by 7 feature values describing the flight (airline, flight number, source, destination, day of week, time of departure and length of flight) and a target value which is either 1 or 0, depending on whether the flight is delayed or not. This dataset is an example of time-correlated stream of data. It is expected that when a flight is delayed (due to weather condition for instance), other timely close flights will be delayed as well.

**Ozone** The Ozone dataset [5] is used to learn an ozone alert forecasting system, a necessary application that would issue warnings to the public before the ozone reaches a dangerous level. The dataset was collected for the Houston, Galveston, and Brazoria area. It consists of 2,536 examples with a feature space of 72 dimensions. The feature space contains different measures of air pollutant and meteorological information, while the target value classifies a day into an "ozone day" or a "normal day". The examples are collected over a period of 7 years and thus the data is subject to concept changes. The physical laws describing the conditional probability $p(y|\mathbf{x})$ is not expected to change. However, due to the limited number of training examples, the unconditional distribution $p(\mathbf{x})$ might evolve with time creating virtual concept drifts. The data exhibit temporal dependence. If ozone levels rise, they don't decrease immediatly.

**SPAM** The SPAM dataset [52] consists of 9,324 examples and was built from the email messages of the Spam Assassin Collection using the boolean bag-of-words representation with 500 attributes. As mentionned in [52], the characteristics of spam messages in this dataset gradually change as time passes (gradual concept drift).

We show in Table 2.3 the properties of the different data streams, including the size of the stream, the size of the feature space and the number of classes. We also show the frequency of each class (%). If more than 3 classes exist, only the frequency of the most three frequent classes are reported. The class frequencies of STAGGER, SEA, and the FLORAs datasets are calculated over one instantiation of the artificial datasets. For *Plane* and *Boolean* problems, the input attributes are normally distributed through the whole input space. Thus, the class frequencies will depend on the concept drift properties in each of the 9 datasets of each problem.

TABLE 2.3: Stream datasets

| Data stream | size | #features | #classes | class frequency % | artificial | real | semi |
|---|---|---|---|---|---|---|---|
| STAGGER | 120 | 3 | 2 | 44.77, 55.23 | ✔ | | |
| FLORA-M | 500 | 6 | 2 | 57.6 42.40 | ✔ | | |
| FLORA-S | 500 | 6 | 2 | 63.80, 36.2 | ✔ | | |
| SEA | 600,000 | 3 | 2 | 62.69, 37.31 | ✔ | | |
| *Boolean* | 1,000 | 3 | 2 | - | ✔ | | |
| *Plane* | 1,000 | 11 | 2 | - | ✔ | | |
| *Circle, Line* | 2,000 | 2 | 2 | 50, 50 | ✔ | | |
| *SineV, SineH* | 2,000 | 2 | 2 | 50, 50 | ✔ | | |
| CAR | 1,296 | 6 | 2 | 57.87, 42.13 | | | ✔ |
| IRIS | 339 | 4 | 4 | 32.74, 25.96, 23.6 | | | ✔ |
| USENET | 5,931 | 658 | 2 | 50.4, 49.6 | | | ✔ |
| ELIST | 1,500 | 913 | 2 | 53.33, 46.67 | | | ✔ |
| Forest | 581,012 | 54 | 7 | 36.5, 48.8, 6.2 | | ✔ | |
| PAKDD | 50,000 | 27 | 2 | 80.3, 19.7 | | ✔ | |
| KDD | 494,020 | 41 | 2 | 19.7, 80.3 | | ✔ | |
| Electricity | 45,312 | 8 | 2 | 42.5, 57.5 | | ✔ | |
| Airlines | 539,383 | 7 | 2 | 55.5, 44.5 | | ✔ | |
| Ozone | 2,536 | 72 | 2 | 97.1, 2.9 | | ✔ | |
| SPAM | 9,324 | 500 | 2 | 25.6, 74.39 | | ✔ | |

## 2.4 Summary

Learning from streams is usually treated using online machine learning techniques which differ from classical batch learning methods in three major points. First, streaming data are not stored or reprocessed, due to memory constraints. Secondly, the prediction model should give answers in an any time fashion, while updating itself with each received information from the stream. Finally, online learning does not presuppose that the training data be independent and identically distributed. It is ready to adapt to changing conditions. When the environment changes, a central concern is to optimize the *stability-plasticity dilemma*. That is, to learn from as much data as possible when the concept is stable while also recognizing when data points become obsolete when the concept evolves.

Dealing with the challenges of online learning presented above is the focus of this work. Our goal is twofold. First, to present a learning method able to *adapt* to concept changes and secondly, to analyze how learning models (capturing the underlying target concept) evolve with time in order to *anticipate* future scenarios and act pro-actively. In the next chapter, we give an overview of the state-of-the-art methods designed to learn in non-stationary environments. We give special attention to *online ensemble methods* which are the base of the learning method we suggest.

# Chapter 3

# State of Art

In this section, we provide an overview of existing online learning systems designed to operate under evolving environments. We go through a number of systems that either use a single learning model or an ensemble of learning models. The list of presented systems is non-exhaustive. However, it gives an insight of the different methodologies used to handle real life learning scenarios, where the training data is received in an online fashion and the underlying environment can evolve with time, creating concept changes.

When evaluating online learning systems, several criteria are taken into account. A "perfect" online learning system meets the following requirements:

1. Fast detection of concept changes (sensitivity to changes)
2. Robustness to false detections
3. Ability to adapt to new concepts
4. Ability of knowledge transfer between consecutive concepts
5. Few parameters to set
6. Low sensitivity to preset parameters
7. Respect of time and memory constraints

These requirements can be contradictory. For instance, sensitivity and robustness are often antinomic (requirement 1 and 2). Methods designed to react quickly to the first signs of concept drift may be misled into overreacting to noise. Some approaches can also have few parameters to be set but the preset values may have a great impact on the type of concept drifts that can be detected (requirement 5 and 6). For instance, a

learning system that detect concept changes using a threshold value on the predictive accuracy may behave differently depending on the threshold value. A low prediction threshold value will detect highly severe concept changes while a high threshold value will detect low severity concept changes, and will react also to noise. In the design of online learning systems, compromises are generally made. As a result, the weakness of some approaches are the strengths of others, and vice-versa.

We first explain in Section 3.1 how online methods generally adapt to concept changes. We then present a number of online learning systems that either use a single learning model (Section 3.2) or an ensemble of learning models (Section 3.3) in order to learn from the data stream and adapt to evolving environments. The different systems are presented chronologically.

## 3.1 Adapting to the Change

The online methods can be divided into two categories [70] depending on whether or not they use an explicit concept drift detector in the process of adaptation to evolving environments. We show here the differences between the two methodologies and present three well-known state-of-the-art drift detection systems.

### 3.1.1 Explicit detection

The first category includes the methods that *explicitly* detect a concept change by monitoring either variations in the distribution of the incoming data or the variations in the classification performance of the ensemble [38].

In these approaches, the learning system is generally reset when a change is detected and the new system starts learning from scratch, with no knowledge of the past. This makes these approaches highly responsive to concept changes. However, their performance relies mostly on the detection mechanism. A false negative detection makes the learning system unable to adapt to the concept change, while a false positive detection resets the learning system when the concept is still stable. By resetting the learning system, these methods cannot transfer acquired knowledge from the old concept to the new one after a concept change. Examples of drift detection mechanisms include: DDM [38], EDDM [6] and ADWIN [9] drift detectors.

### DDM

The basic idea in DDM is that, when the environment is stable, the learning algorithm is expected to make less errors with time. If however, the online error increases, the underlying target concept is assumed to be changing and the learning algorithm should be reset. DDM defines two error levels:

1. the warning level: beyond this level, the examples are stored in anticipation for change.
2. the drift level: beyond this level, a concept drift is assumed to be happening. The learning algorithm is reset and is trained on the examples stored since the warning level.

### EDDM

The Early Drift Detection Method (EDDM) [6] is an enhancement of the DDM algorithm [38] which detects a concept change by monitoring the online predictive error of the learning algorithm.

Instead of taking in consideration the number of errors only, EDDM considers *the distance* between two classification errors. The basic idea in EDDM is that, when the environment is stable, the distance between two consecutive errors increases with time. A decrease in the distance is therefore the evidence of a concept drift. Thus, the distance is monitored and, if the distance falls below a predefined threshold $\alpha$, the warning level is triggered, indicating that a concept drift might have happened. From this moment, all the training examples presented to the system are used for learning and then stored. If the distance goes below another predefined threshold value $\beta$, where $\beta < \alpha$, the concept drift is confirmed and the system is reset. It was shown that EDDM performs better than DDM on very slow drifts.

### ADWIN

ADWIN detects a concept change using an adaptive sliding window model. ADWIN looks at all possible subwindows partitions in the window of training data. Whenever two large enough subwindows have distinct enough averages, a concept change is detected and the older partition of the window is dropped. ADWIN's only parameter is a confidence bound $\gamma$, indicating how confident we want to be in the algorithm's output.

### 3.1.2 Implicit adaptation

In the second category of online methods, the learning system adapts *implicitly* to a concept change by holding an ensemble of experts[1]. Each expert is given a weight that reflects its classification record; if the weight value drops under a predefined threshold, the expert is removed and replaced by a new one. The new experts starts learning from scratch with no knowledge of the past, allowing them to adapt to a potential concept change. Advantages of implicit methods include their capacity of achieving knowledge transfer between consecutive concepts, and this by keeping old experts in the ensemble after the concept change. These methods do not rely on a concept change detection system, and thus, unlike their rivals, are not sensitive to false detection alarms. Their main drawbacks include slow reactivity to a concept change if it takes long to delete experts from the ensemble.

In both categories, the adaptation ability of the learning system depends mainly on the value of the predefined parameters as the threshold value used to remove an expert from the ensemble or the parameters used to explicitly detect a concept drift. Choosing the parameters values requires generally an a priori knowledge of the concept drift properties. As a result, even if a set of parameters is adapted to a specific concept change, it might not work for others.

## 3.2 Online Classifiers

We present here online systems that use a single learning model (a classifier). The presented systems comprise two pioneer drift handling systems: IB3 and FLORA, and two more recent systems: RePro and PreDet.

### 3.2.1 IB3 (1991)

IB3 [1] extends the nearest neighbor algorithm, which saves and uses selected instances from the training data to generate classification predictions. IB3 describes a concept by a set of instances, having each a classification record i.e. a number of correct and incorrect classification attempts. A classification record summarizes an instance's classification performance on past successive training instances and suggests how it will perform in the future.

[1] We use the same terminology used by Cesa-Bianchi in its book on online learning [20].

IB3 employs a significance test to determine which instances are good classifiers and which ones are believed to be noisy. The former are used to classify subsequently presented instances. The latter are discarded from the concept description. If an instance gets wrongly classified, the instance is added to the concept description. IB3 was criticized for being able to adapt to gradual concept drifts only, and its adaptation is relatively slow [96].

### 3.2.2 FLORA (1996)

In FLORA [96], a concept is described using a simple representation language based on attribute-value logic without negation. For instance, (color=*white* and temperature=*low*). A conjunction of attribute-value pairs is called a *description item* and a concept is represented by a set of description items. FLORA keeps description items that are consistent with a sliding window of examples.

Four versions of FLORA exist. The first version, *FLORA1*, uses a fixed size sliding window. Selecting a window size remains a difficult problem. A narrow window will not accommodate a sufficient number of examples for a stable concept description. A wide window; on the other hand, will slow down the learner's reaction to a concept drift. The second version, *FLORA2*, uses a dynamic window size. The idea is to shrink the window when a concept drift seems to occur, and keep the window size fixed when the concept seems stable. Otherwise, the window should gradually grow until a stable concept description can be formed. In order to adjust the window size, *FLORA2* relies on an explicit concept change detection system. Changes are detected by monitoring two indicators: the predictive accuracy on the recently classified instances and the properties of the evolving concept. The basic assumption is that a significant drop in the predictive accuracy or an explosion of the number of description items is a sign of a possible concept change. The third version, *FLORA3* extends *FLORA2* by dealing with recurring contexts. It stores concepts in stable situations and recall them in similar contexts. The last version, *FLORA4*, handles noisy environments, on the expense of a slower reaction to concept changes.

### 3.2.3 RePro (2005)

The RePro method [99] combines *proactive* and *reactive* predictions. In a proactive mode, RePro anticipates the future concept when a concept change occurs and prepares prediction strategies in advance. If the anticipation turns right, the predicted concept is used to classify the upcoming examples. Otherwise, the reactive mode takes over, adapting a new learning model to the new data.

The proactive mode creates a history of the different concepts encountered during the data streaming, in the form of a Markov chain [75]. The Markov chain is built incrementally where the states represent the stable concepts and the arcs the transitions between consecutive concepts.

Before adding a new concept to the history, two measures should be defined:

- *a stability measure* which is used to decide whether the learning model has learnt enough training examples and thus can be trusted to represent the underlying target concept. In RePro, the learning model should be trained on $m$ examples at least to be considered as stable. The value of $m$ needs to be predefined.
- *a concept equivalence measure* which evaluates the resemblance between two concepts. This measure ensures that the same concept is not considered twice in a row in the Markov chain in construction. The equivalence between two concepts $C_1$ and $C_2$ is computed in RePro as the mean number of examples on which both $C_1$ and $C_2$ agree on the predicted label.

A concept change is detected using a fixed size sliding window on the training data. Whenever the classification error exceeds a predefined threshold, a drift is detected and the system checks the most likely concept to come according to the Markov chain. If no potential candidates exist, the whole list of possible concepts is examined and the concept with the lowest classification error on the sliding window is selected to classify upcoming instances. The error of the selected concept should nevertheless be smaller than a predefined threshold. Otherwise, the reactive mode takes over the proactive one, resetting the learning algorithm which learns from scratch starting with the examples in the sliding window.

In total, the system's parameters include: an error threshold $\theta_1$ and a window size $w_1$ for the drift detection system; the parameter $m$ for the stability measure; a threshold $\theta_2$ for the concept equivalence measure; a probability threshold $\theta_3$ for the Markov chain and a classification accuracy threshold $\theta_4$ to decide whether the reactive mode should take over the proactive mode.

### 3.2.4 PreDet (2008)

The PreDet [15] algorithm analyzes the change to anticipate future scenarios. PreDet uses decision trees as classifiers and anticipates future trees by predicting for each decision node the evaluation measure of each attribute, this value being used to determine which

attribute will split the node. In case of a leaf node, PreDet predicts its class label distribution.

The anticipation mechanism operates as follows. The data stream is split into $m$ consecutive batches of examples where each batch is used as a time reference to estimate the current evaluation measures along with the class label distributions. The future evaluation measures are predicted using a linear regression model trained on the last $r$ estimations of the predicted parameters.

The classification process operates as follows. The sequence consisting of $r$ consecutive batches $(S_i, ..., S_{i+r-1})$ is used to predict the decision tree for the batch $S_{i+r}$ that chronologically follows the sequence using the anticipation mechanism. The predicted tree is then used to classify samples in $S_{i+r}$. This process is repeated for each value of $i$ where $i = 0, \ldots, m - r$.

### Summary

Single learning models are among the first systems suggested to adapt to concept changes. An important challenge is to decide when past data become obsolete and should therefore be discarded from the learner's memory. Several directions have been proposed to dynamically adapt the memory of the past data. One is to select *instances* relevant to the current concept. For instance, in FLORA [96], time sliding windows on the past data are used, either with a fixed or a varying window size. This technique necessitates either a priori knowledge about the dynamics of the environment, or a good heuristic that guesses when to shrink or expand the sliding window. In IB3 [1], the most relevant instances do not necessarly represent the most recent data as with sliding windows. Instances from the stream can be stored in memory (or removed) according to their classification record, which is deemed to reflect how relevant they are to the current context. Another approach relies on *weighting past data* with respect to their relevance to the current situation [54]. Again, this is all dependent on the design of an appropriate weighting mechanism.

More recent techniques take advantage of the change in order to *anticipate* the near-future characteristics. RePro [99] and PreDet [15] are examples of such systems, where the sequence of concept changes is analyzed in order to predict the most probable upcoming concept. The main difficulty is to identify the different concepts encountered during learning. This is either realized using a drift detection system [99] or by making a priori assumption on *when* concepts are expected to change [15]. While the former strategy requires choosing adapted parameter values for the detection system, the latter needs a priori knowledge about the dynamics of the environment.

## 3.3 Online Ensembles of Classifiers

By contrast with the single learning models approaches that explicitly control the memory of past data, ensemble methods rely instead on the control of past hypotheses. Ensemble-based approaches maintain an ensemble of online learners that each maintain their own memory of the past. By managing the population of these base learners, one is implicitly controlling the use of past data.

A large variety of ensembles learning methods have been proposed for tackling changing concepts [12, 13, 59, 70, 84, 85, 90]. In addition to the implicit control of past memory, ensemble methods have the advantage of holding diverse online learners (experts), and therefore multiple concept descriptions. They generally adapt to a variety of changing conditions in the environment and do not require fine tuning of their parameters.

The ensemble methods can be grouped depending on many criteria such as:

**Data receipt** The data is either received and processed one instance at a time or in sequential batches. A main problem when learning on sequential batches is that a drift might occur inside a batch. An expert trained on this batch will learn misleading and sometimes contradicting information. Choosing the batch size might also be a problem when we have no a priori knowledge of the training data. If, for instance, we know that what we learn depends on the season of the year then we may set the batch size such that it covers each season separately.

**Data pre-processing** The data are either used as is or can be pre-processed by changing the data distribution using sampling and/or weighting methods. The latter strategies use the idea of offline bagging and boosting methods [77] where the static batch of training data is learnt by each expert after being resampled with replacement. As a result, each expert is trained on a different data distribution which increases the diversity of the experts in the ensemble.

**Experts' learning extent** In most cases, when the data is received in batches, an expert learns on a block of data and its learning stops at this point. In other scenarios, however, experts don't stop learning: they constantly update their knowledge with new observed training data.

**Experts deletion** Some approaches delete an expert from the ensemble when its performance drops under a predefined threshold. Some other approaches always remove

the worst expert each $\tau$ time steps. In the latter case, and if the data is processed one instance at a time, promising experts might be removed if $\tau$ examples are not enough for the expert to learn a stable concept and get a relatively good predictive performance. In order to avoid random and unmeaningful deletion operations, approaches may set a maturity age, a minimum number of training examples that an expert should observe before being evaluated for deletion. The deletion problem is not encountered when the data is processed one block at a time since the block size is generally large enough to learn a stable concept description.

**Ensemble's final prediction** The ensemble classifies a test instance using the experts' predictions and weights. In case of unweighted ensembles, each expert's prediction carries equal weight and the final prediction is the result of a majority vote. In case of weighted ensembles, experts are not considered equal in the voting process. Each expert is assigned a weight that reflects the importance of the expert's prediction. The higher the weight, the more the impact of its vote on the final prediction.

Different methods have been proposed for weights computing. Most methods evaluate an expert based on its classification performance on recently observed data. Other methods use local accuracy to weight an expert: the expert's classification performance is evaluated on the neighborhood of the test instance and the weight is set accordingly.

The experts predictions are then integrated to give the ensemble's final prediction. Classical integration methods include: majority voting, weighted majority voting, selecting the prediction of the expert with the highest weight, using a roulette-wheel selection on the experts weights, and others [28, 47, 66, 78].

**Concept change handling** Ensemble learning methods can be divided into two categories depending on whether they adapt implicitly or explicitly to concept changes (see Section 3.1).

In *explicit* scenarios, a drift detection mechanism is used to adapt to concept changes. After a change is detected the ensemble is generally reset. These approaches are highly responsive to concept changes. Their main drawbacks include sensitivity to false alarms and unability to achieve knowledge transfer between consecutive concepts.

In *implicit* scenarios, adapting to concept changes is achieved by the perpetual renewal of the population of experts using removal and addition operations. These approaches can achieving knowledge transfer between consecutive concepts by keeping old experts in the ensemble after the concept change. Their main drawbacks include slow reactivity to a concept change if it takes long to delete experts from the ensemble.

In the following, we present several online ensemble approaches that have been designed to learn under evolving environments. The methods are categorized according to the different criteria presented above (data receipt, data pre-processing, learning extent, etc...) in Table 3.1.

### 3.3.1 DWM (2003)

The Dynamic Weighted Majority algorithm [59] does not use a drift detection method. It handles drifting concept using an ensemble of classifiers that are weighted according to their accumulated predictive performance observed during their lifetime. Each classifier in the ensemble is initially assigned a weight of one. The weight is decreased by a multiplier constant $\rho$ when the classifier misclassifies an instance, when the current time step is a multiple of $p$. This ensemble method copes with concept drift by dynamically adding and/or removing classifiers every $p$ time steps. A new classifier is added if the ensemble misclassifies a test sample and a classifier is removed if its weight is lower than a predefined threshold $\theta$. This strategy makes the ensemble size variable. As for the ensemble final prediction, it corresponds to the class label with the highest accumulated weight (weighted majority vote).

### 3.3.2 CDC (2003)

CDC uses a weighted committee of decision trees to deal with concept changes [85]. When an example is received, each committee member predicts its class value. The predictions are then combined by a weighted vote. The weight of each member reflects its classification performance on the recently received data. Thus, committee members that have been doing well recently have more to say. When a member's performance drops too low, it is replaced by a completely new member. It is expected that when a concept changes, many members retire from the committee, allowing new members to learn the new concept. While old members are reliable in stable environments, youngest members tend to be valuable during changing times.

### 3.3.3 KBS-stream (2005)

The KBS-stream algorithm [84] is a boosting-like method [33, 82] that trains a classifier ensemble from data streams. In KBS, the data are received in a sequence of batches. With each received batch, either a new classifier is added or the latest added classifier is updated. Before learning, the data distribution in the recent batch is changed using sample weighting and sampling strategy: the data are passed through the existing classifiers

and the distribution is modified such that the last classifier learns something different than the remaining ones in the ensemble. The data are assumed to be independent and identically distributed in each batch and a drift is allowed to occur between two consecutive batches but never inside a batch. These conditions are not met in many real-world online problems.

### 3.3.4 DIC (2008)

The Dynamic Integration of Classifiers [90] uses an ensemble integration technique to handle concept drift. In this approach, the data is received as a sequence of fixed-size batches. The data is then divided into overlapping or non-overlapping blocks. With each data block, a new expert is trained and added to the ensemble. When the ensemble size reaches its maximum size, the *replace the loser* pruning strategy is used: the expert with the worst classification record on the last data block is removed and replaced by a new one, trained on the most recent data block. In dynamic integration of classifiers, each classifier is given a weight proportionnal to its local accuracy with regard to the instance tested. The best classifier is selected to predict on the behalf of the ensemble or the classifiers are integrated using weighted voting. It was shown that dynamic integration gives better accuracy than the most commonly used integration techniques, specially in the case of local drifts [90].

### 3.3.5 Adwin Bagging (2009)

Bagging has been used in classical offline machine learning as a combining technique to improve the accuracy of weak classifiers. Offline bagging methods [16] create $M$ base learners (classifiers) trained on $M$ training sets of size $N$, each generated by drawing random samples with replacement from the original training set. As a result, each base learner is trained on $K$ copies of a training example $(\mathbf{x}, y)$ where $K$ follows a binomial distribution. For large values of $N$, the binomial distribution tends to a Poisson distribution [2] with $\lambda = 1$, where

$$\text{Poisson}(\lambda; k) = \Pr(K = k) = \frac{\lambda^k \exp^{-\lambda}}{k!}$$

and Pr is the probability mass function of $K$.

The predicted class value $\tilde{y}$ of an instance $\mathbf{x}$ is the result of a majority vote among the base learners.

Bagging methods have also been suggested for streaming data [77]. In online mode, the whole training set is not available. Hence, instead of resampling, instances are weighted according to a Poisson distribution with $\lambda = 1$.

ADWIN Bagging [13] is the online bagging method with the addition of the ADWIN algorithm as a change detector [9]. When a concept change is detected, ADWIN Bagging removes the worst classifier of the ensemble and replaces it with a new classifier.

### 3.3.6 ASHT-Bagging (2009)

The ASHT-bagging algorithm [13] is a variant of online bagging designed to tackle with non-stationary concepts. The approach builds an ensemble of Hoeffding trees [30] with different tree sizes: small trees to adapt more quickly to changes and large trees which are better for stationary concepts. In this approach, the data is received one instance at a time. A tree in the ensemble is updated with each new observed training data. When a tree reaches its maximum size, it is pruned, even for stationary concepts.

There are two different pruning options:

- delete the oldest node, the root, and all of its children except the one where the split has been made. The root of the child not deleted becomes the new root.
- delete all the nodes of the tree, restart from a new root.

The maximum allowed size for the $n$-th ASHT tree is twice the maximum allowed size for the $(n-1)$-th tree. The size of the first tree is predefined by the user.

The diversity of the ensemble improves bagging and allows the adaptation to concept drifts. The predictions of the trees in the ensemble are combined either by a weighted or a simple vote, depending on the user's predefined choice. In case of a weighted ensemble, each tree has a weight proportional to the inverse of the square of its error. The error is monitored with an exponential weighted moving average (EWMA) which is defined as follows:

$$X_t = \alpha z_t + (1-\alpha)X_{t-1} = X_{t-1} + \alpha(z_t - X_{t-1})$$

where $X_t$ is the moving average, $z_t$ is the latest measurement, and $\alpha$ is the weight given to the latest measurement (between 0 and 1). The $\alpha$ value is set to 0.01 in the ASHT-Bagging algorithm.

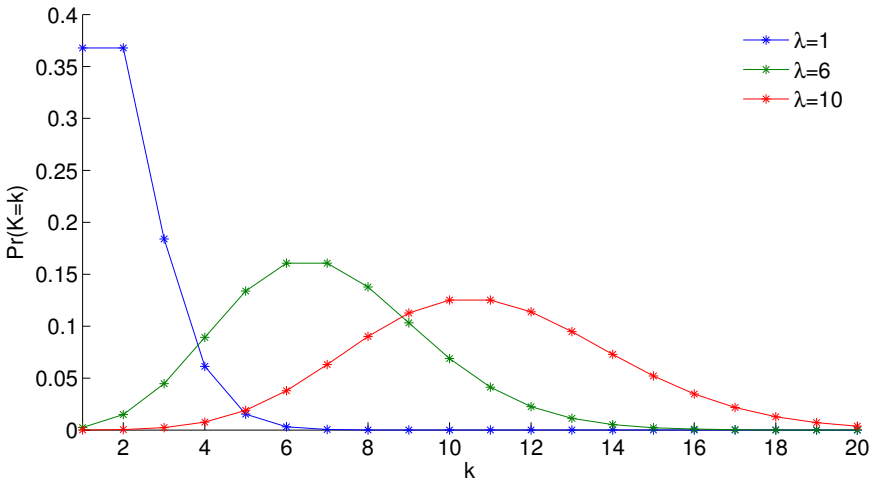


FIGURE 3.1: The poisson distribution using different $\lambda$ values.

### 3.3.7 CCP (2010)

The Conceptual Clustering and Prediction method [52] is an ensemble method that uses clustering to build its list of classifiers and is able to handle recurring concepts. Each batch of $m$ instances is mapped into a conceptual vector (descriptor). The vector is either assigned to an existing cluster according to a distance threshold $\theta$ or a new cluster is created. In the latter case, a classifier trained on the batch is assigned to the cluster. In the former case, the classifier of the corresponding cluster is updated with the batch instances. The classifier of the new or existing cluster is then used to classify the next $m$ instances. The number of clusters cannot be larger than $c_{max}$ otherwise the new item is incorporated into the nearest cluster.

### 3.3.8 Leveraging Bagging (2010)

The Leveraging Bagging method [12] is an online bagging method designed to handle drifting concepts. Leverage bagging is different from online bagging in three points. First, the instances are weighted according to a Poisson distribution with a $\lambda$ value greater than one in order to achieve more diversity in the generated weight values, as shown in Figure 3.1. Secondly, in order to add more diversity to the classifiers, error-correcting output codes can also be used [29]. With correcting codes, each classifier $h$ maps a class value $c$ to a binary value $\mu_h(c)$ in a uniform, random way. This forces the classifiers to learn different functions. Finally, Leverage Bagging handles drifting concepts using the ADWIN method to detect concept changes [9]. If ADWIN detects a change in the error of one of the classifiers, the classifier with the highest error is replaced with a new one.

### 3.3.9 DDD (2012)

There also exist algorithms that use multiple ensembles to handle concept changes. For instance, DDD algorithm by Minku [70] uses four ensembles with different diversity levels to cope with evolving environments. According to Minku, diversity in the ensemble helps reduce the initial drop in the classification accuracy that happens just after the change. When the concept is stable, however, low diversity in the ensemble gives more accurate results. The diversity of an ensemble is achieved by online bagging. Whenever a training example is available, it is presented $K$ times for each learner, where $K$ is drawn from a $Poisson(\lambda)$ distribution. Different levels of diversity are realised using different $K$ values. The higher the $K$ value, the lower the diversity level.

In DDD, any drift detection system can be used to detect possible concept changes. In the experimental setup, the same system as EDDM was used. Before a drift is detected, the learning system is composed of two ensembles: an ensemble with low diversity and an ensemble with high diversity. The low and high diversity levels are controlled by the parameters: $\lambda_l$ and $\lambda_h$, respectively, where $\lambda_l > \lambda_h$. Both ensembles are trained with incoming examples but only the low diversity ensemble is used for system predictions. After a drift is detected, new low diversity and high diversity ensembles are created. The ensembles corresponding to the low and high diversity ensembles before the drift detection are kept and denominated old low and old high diversity ensembles. The old high diversity ensemble starts to learn with low diversity (with a small $\lambda$ value) in order to improve its convergence to the new concept.

The system's predictions are determined by the weighted majority vote of the output of the old high diversity, the new low diversity and the old low diversity ensembles. The new high diversity ensemble is not considered because it is likely to have low accuracy on the new concept. The weight of an ensemble is proportional to its online classification accuracy since the last drift detection. The weight of the old low diversity ensemble is multiplied by a constant $W$ which allows controlling the trade-off between robustness to false alarms and accuracy in the presence of concept drifts, and all the weights are normalized. It is important to note that while the predictions of the ensembles are weighted, the predictions of the classifiers inside an ensemble are combined using a simple majority vote.

TABLE 3.1: Properties of online ensemble methods

| Algorithm | Prepro- | Data Receipt | | Learn Extent | | Deletion | | vote | | drift handling | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | cessing | batch | inst. | limited | cont. | thr. | no thr. | simple | wgt | expl. | impl. |
| AdwinBag | ✔ | | ✔ | | ✔ | | ✔ | ✔ | | ✔ | |
| LBag | ✔ | | ✔ | | ✔ | | ✔ | ✔ | | ✔ | |
| CDC | | | ✔ | | ✔ | ✔ | | | ✔ | | ✔ |
| DDD | ✔ | | ✔ | | ✔ | | ✔ | ✔ | ✔ | ✔ | |
| DWM | | | ✔ | | ✔ | ✔ | | | ✔ | | ✔ |
| ASHT | ✔ | | ✔ | | ✔ | | ✔ | ✔ | ✔ | | ✔ |
| DIC | | ✔ | | ✔ | | | ✔ | | ✔ | | ✔ |
| KBS | ✔ | ✔ | | ✔ | | | ✔ | | ✔ | | ✔ |
| CCP | | ✔ | | ✔ | | | ✔ | | | | ✔ |

## 3.4 Summary

In the recent years, a large number of online learning systems have been suggested to learn from data streams, dealing with the challenges of learning under changing conditions while also meeting memory and time constraints.

While some systems were designed to *adapt passively* to concept changes, other systems took advantage of past learning experiences to *act proactively* and anticipate future scenarios. Most proactive systems worked on *recurrence*, keeping traces of old and outdated concepts in case they become useful in the future. In 2008, anticipation went beyond the case of recurring contexts. The notion of *concept change mining* was introduced with the PreDet system [15] where the learning models were analyzed in order to predict the near-future characteristics of the evolving domain.

In the rest of this thesis, we suggest a novel online learning method that combines passive adaptation to concept changes with a proactive anticipative mechanism that predicts (if possible) the upcoming concept, treating the cases of *recurrence* and *concept change mining*. We present the adaptive strategy in Chapter 4 followed by its combination with the anticipative mechanism in Chapter 5. We finally conclude our main contributions and suggest future research directions in Chapter 6.

# Chapter 4

# Adaptation to Concept Changes

## 4.1 Motivation

In this Chapter, we present an online learning algorithm that relies on an ensemble of experts in order to adapt to drifting concepts. The method we propose adapts implicitly to concept changes without the use of an explicit change detection mechanism. Our choice was motivated by the advantages of implicit methods over explicit ones, as detailed in Chapter 3.

Our main goal in the design of the online ensemble algorithm is to make its behavior minimally sensitive to its parameter values. Explicit adapting systems rely on preset parameters in their detection of changing concepts. Abrupt changes are easier to detect by explicit detection methods, but difficulties arise with the slow gradual changes. The parameters should be tuned such that the detection system gets highly sensitive to any change while at the same time, being robust to noise. These two requirements generally require opposite strategies. In addition, when a concept change is detected, explicit methods take harsh decisions, generally resetting the ensemble of experts in the pool of learners. In case of a false alarm, this has the effect of losing all acquired knowledge and learning from scratch, despite the fact that no real change in the underlying target concept occured. The sensitivity of explicitly adaptative systems to the detection mechanism led us to consider implicit adaptation methods.

Implicit methods adapt implicitly to changes by adding and removing experts from the ensemble. Experts are generally removed if their predictive performance falls under a predefined threshold, suggesting that the expert is outdated and does not represent the current underlying concept anymore. Even when the concept is stable, a wrong deletion of one or more experts doesn't have a major impact as with explicit systems which react

severely in case of suspicion of change, resetting all the experts in the committee. Implicit adaptation also rely on predefined parameters in the adaptation to change. For instance, if the system uses a threshold parameter to decide if an expert should be deleted, then depending on the threshold value, an expert might be deleted even though the concept is stable, or might not be deleted, even though the concept is changing. Thus, challenges are always present, in a form or another.

In Section 4.2, we present the framework of ensemble methods adapting implicitly to concept drifts. We analyse its main components and the main factors that contribute to the sensitivity of an implicit system to its preset parameters. We then present in Section 4.3 our ensemble algorithm which adapts implicitly to concept changes. The ensemble learning algorithm was designed with the goal of overcoming the sensitivity of its rivals to the values of the preset parameters. In other words, the presented ensemble method adapts to a large variety of concept changes, with different speeds and severities, and this, without the need of fine tuning its parameters.

## 4.2 Framework

Ensemble methods with implicit adaptation to changes maintain an ensemble of experts $\{h_t^i\}_{1\leq i\leq N}$, each of them adapting to the new input data. They administer this ensemble or committtee thanks to a deleting strategy and an insertion one. The main principles of these methods are the following issues:

- Prediction: for each new incoming instance $\mathbf{x}_t$, the prediction $\tilde{y}_t = H(\mathbf{x}_t)$ results from a *combination* of the prediction of the individual experts
- Learning: each expert in the ensemble continuously learns from the received training data $(\mathbf{x}_t, y_t)$ until it is removed from the ensemble.
- Deletion: every $\tau$ time steps, the experts are *evaluated* on a window of size $\tau_{eval}$. Based on the results of this evaluation, the deletion procedure chooses one or more expert(s) to be removed.
- Insertion: every $\tau$ time steps, a new expert can be *created* and inserted in the ensemble. A new expert starts learning from scratch with no knowledge of the past. It is protected from possible deletion for a duration $\tau_{mat}$.

The remainder of this section addresses the following:

- The type of experts used in the ensemble.
- The prediction process, on how the individual predictions can be combined to form the ensemble's final prediction.
- The weighting functions used to evaluate experts in the ensemble.
- The sensitivity of the implicit framework to the deletion strategies.

### 4.2.1 Experts

Ensemble methods rely on a set of experts i.e. forecasters that predict the target value $\tilde{y}_t$ of an instance $\mathbf{x}_t$. Generally, an expert is a base learner. For instance, a support vector machine [45], a decision tree [80], a neural network [43], a naive bayes classifier [100], among others. The base learner constantly updates a predictive hypothesis $h$ induced via the training instances observed during its lifetime.

In our work, we aim at extending the type of experts in the ensemble. Instead of using learning models only, we would also have experts that don't "learn" from the training data, providing predictions of a target value $\tilde{y}$ regardless of the corresponding input $\mathbf{x}_t$. For instance, a predictor that would always predicts the class $A$ or a predictor that predicts the same class value as the last received example i.e. $\tilde{y}_t = y_{t-1}$.

Some ensemble algorithms have been designed to deal with concept drifts using a specific type of learning models. For instance, the ASHT-bagging algorithm by Bifet et al. [13] which uses Hoeffding decision trees only [30]. One of our goals in the design of the adaptative ensemble is the ability to use any type of learning model. We even wish to push this idea further by mixing different types of base learners in the ensemble at a time. For instance, different decision trees, support vector machines and neural networks, altogether in one ensemble. The diversity in the types of learners allows one to capture various types of regularities in the underlying target concept. Generally, ensemble algorithms avoid mixing different base learners since one would also have to implement control policies to determine which base learner to add by the insertion strategy [59].

### 4.2.2 Prediction

In classical ensemble learning methods, such as Adaboost, it was shown that combining experts improves in many cases their predictive performance [33]. Classical ensemble

learning methods allow the *cooperation* of a set of experts by combining their predictions, usually by a majority vote or a weighted vote.

In the presence of changing concepts, an online ensemble method adapting implicitly to concept changes may contain two categories of experts at the same time: experts trained on the current new concept, and expert trained on the previous and old concepts. In such case, the cooperation with the old and outdated experts is not the best strategy. Ideally, we would allow the cooperation between the "good" experts, that are trained on instances from the new concept, and exempt the "bad" outdated experts from voting. Assessing the quality of an expert is generally done by computing a weight, reflecting the expert's quality in terms of its expected predictive accuracy on future instances. In weighted ensembles, the experts with the highest weights, seen as the most promising ones, contribute more than others to the ensemble's final prediction. The least promising candidates, with the smallest weights, can be the subject of expulsion from the ensemble by the deletion strategy.

Seen from this perspective, the weighted experts are rather in *competition* than cooperation. Weighted experts compete for they life in the ensemble, and also compete to be heard in the voting process. If more than one promising experts exist in the ensemble, they *cooperate* in the voting process. Thus, experts are in a *competition* from a predictive accuracy perspective, and good experts *cooperate* when forming the ensemble's final prediction. The predictions of the experts are generally combined using one of the following combination functions:

- **V**: a simple vote. The simple vote doesn't take into account the goodness/badness of the predictor when combining predictions.
- **WV**: weighted vote. The main issue in WV is that in case of a concept drift, if the ensemble contains a relatively large number of outdated experts trained on an old concept, their weights will sum up and will be high enough to win the vote. This delays the adaptation to the new concept.
- **WVD**: weighted vote after suppressing the predictions of the worst experts i.e. the experts with the performances that fall into the lower half of the performance interval. This overcomes the problem of the weighted vote, explained above.
- **MAX**: the prediction of the best expert is selected i.e. the expert with the largest weight is selected for the final prediction.

### 4.2.3 Weighting functions

In weighted ensembles, each expert is assigned a weight that reflects its predictive record. The weight value translates *how well* the expert is expected to perform on future instances. We give here an insight into the main methods proposed for weights computing.

In DWM [59] each classifier in the ensemble is initially assigned a weight of one. The weight is decreased by a multiplier constant $\rho$ when the classifier misclassifies an instance. The main issue in this scenario is that the prediction records are computed over the experts' lifetime, and not over their recent predictions. Therefore, if a concept drift recently occured, an expert with a high predictive performance on the old concept will still have a relatively high weight after the drift even though its knowledge of the underlying target concept became outdated. This misleads the ensemble into considering that outdated experts are adapted to the current context when in fact, they are not. This type of weight assignement is also used in other systems such as the systems described in [20] and [58].

Other methods use local accuracy to weight an expert: the expert's classification performance is evaluated on the neighborhood of the test instance and the weight is set accordingly. It was shown that this weighting function leads to better accuracy, specially in the case of local drifts [90]. Generally, this method is only beneficial when training data are processed as a sequence of batches in order to ensure that enough data are available to assess the neighborhood of test instances. In case of a pure online learning scenario where instances are processed one at a time, two values should be defined: (a) the size of the past data $w$ from which neighborhood instances are considered and (b) the neighborhood size $k$. The value of $w$ should be small enough to avoid the presence of outdated data and since $k$ should be smaller than $w$, this can lead to relatively small neighborhood sizes and consequently unstable weight values.

Finally, weighting methods can evaluate an expert based on its classification performance on recently observed data [85]. It is common practise to use only the most recent data available because their characteristics are very likely to be best reflecting those of (unknown) near future data [15]. Therefore, it is expected that experts with a good predictive performance on the recent data will also perform well in the near future. For instance, in CDC [85], the weight of an expert is simply its predictive accuracy (the mean number of correct predictions) on its last $\tau_{eval}$ predictions. Using the predictive accuracy for weight computing makes weight values more readable and interpretable.

### 4.2.4 Deletion strategies

Ensemble methods adapt implicitly to concept drifts by deleting experts with bad predictive performance. A deleted expert is considered unadequate to the underlying target concept and is replaced by a new expert that starts learning from scratch.

The deletion strategy plays the key role in the adaptation process since it allows experts to forget the acquired memory of outdated training data. Generally, an expert is removed according to the *replace the loser* pruning strategy which deletes the worst expert [12, 13, 59, 85, 90] in the ensemble. In some approaches, the "loser" is removed only if its evaluation record is below a predefined threshold.

We study here the sensitivity of the adaptation process to the parameters of the deletion strategy, which triggers the deletion of experts and generally, also the addition of new experts. We address the following questions:

Q1 Does the use of a threshold enhance the "replace the loser" strategy? And how can we find the optimal threshold value?

Q2 Is the "replace the loser" strategy the best heuristic?

Q3 How can the deletion strategy achieve both plasticity and stability, without being biased towards one or the other?

#### 4.2.4.1 Deletion strategies using a threshold value

A deletion strategy based on a threshold replaces experts in the ensemble when their *prediction record*, evaluated on the window size $\tau_{eval}$, is below a predefined threshold $\theta_d$. When the performance is computed as the percentage of correctly classified instances on the evaluation window, only the experts with a classification accuracy of at least $\theta_d$% thus remain in the ensemble.

This strategy leads to different behaviors depending on the characteristics of the environment. For the sake of the analysis, let us suppose that a concept drift occurs corresponding to a change in the label of $sev$% of the input space. This is called the *severity* of the concept drift [69]. Let's suppose further that all experts in the ensemble have a perfect classification accuracy of 100% before the drift. We are then faced with a difficult conundrum.

If $sev << 1 - \theta_d$, that is if the severity of the concept change is below the detection capability induced by the threshold, we may very well end up with an unchanged ensemble

of experts. Charged with their memory of outdated data they will be slow, if ever, to adapt to the new concept. For instance, if $sev = 15\%$ and $\theta_d = 75\%$, then after the change, the performance record of the experts will drop to 85% and thus won't be removed from the ensemble according to the relatively small threshold value.

However, the choice of a higher threshold is loaded with two potential pitfalls. First, in case of a noisy environment, the classification accuracy of many experts may drop under the $\theta_d$ value resulting in a severely impoverished committee even though no real concept drift did happen. Second, new experts may not be able to reach the exacting threshold before they reach the maturity age ($\tau_{mat}$) and are therefore no longer protected from deletion.

Overall, it is difficult to set a value for a threshold without well-informed prior knowledge on the dynamics of the environment. Too low a threshold threatens the plasticity of the system, while a high one may cause havoc in the ensemble and prevent stability and good prediction performance. For these reasons, ensemble methods that do not rely on explicit threshold have been promoted.

#### 4.2.4.2 Deletion strategies that delete the worst expert

Rather than having to set a threshold for deciding which expert to eliminate, one can encourage the diversity in the ensemble while preserving the best current expert by periodically removing the worst ones. One such strategy removes the worst expert in the ensemble every $\tau$ time steps. This seems a sensible strategy since it should discard experts that no longer correspond to the current state of the environment and introduce at the same time new experts. However, a potentially vicious interaction involving the parameters $\tau$ and $\tau_{mat}$ may ruin this hope.

Let us first suppose that the period of time during which a new base learner is protected from deletion: $\tau_{mat}$ is less than $\tau$. At each new deletion time, the newest expert is prone to be deleted and will be if it did not have time to learn enough of the regularities in the environment. But $\tau$ cannot be too large lest the system looses any plasticity.

Suppose then that $\tau \leq \tau_{mat}$. Again the risk exists that deletion will affect only the newest experts in the ensemble effectively dividing the ensemble into a protected subset of the best and oldest experts and a subset of the newest ones that are never able to catch up with the other ones except when the overall performance of the system has so declined that even a low prediction performance may allow an expert to avoid elimination. A question is then whether it is possible to break this poor behavior which impedes plasticity by never allowing new experts to enter the top subset.

More formally, let's assume that the experts are indexed from 1 to $N$, where $N$ is the size of the ensemble. Every $\tau$ time steps, the experts are evaluated for deletion. At the time of the first deletion, the worst expert $h_{k_1}$ is replaced by a new expert, with the same index. The new expert $h_{k_1}$ is now protected from deletion for $\tau_{mat}$ time steps.

if $\tau_{mat} < \tau$, then at the time of the second deletion, $h_{k_1}$ is not protected from deletion anymore. It is possible that $\tau_{mat}$ time steps were not enough for the young expert to exceed the performance of the others experts in the ensemble. As a result, $h_{k_1}$ is still the worst in the ensemble. It is replaced by a new expert and so on.

if $\tau \leq \tau_{mat}$, then at the time of the second deletion, $h_{k_1}$ is still protected from deletion. The next worst expert is $h_{k_2}$, where $k_2 \neq k_1$. Once $h_{k_1}$ becomes unprotected from deletion, it is removed from the ensemble. Then $h_{k_2}$ is removed again and so on. The deletion cycle contains $d = \tau_{mat}/\tau$ experts with different indexes.

Depending on the values of $\tau_{mat}$, $\tau$ and the size $N$ of the ensemble, we get the following special cases for the size $d$ of the deletion cycle (the number of continuously deleted experts):

- If $\tau_{mat} = N$ and $\tau = 1 \implies d = N$. All the experts are removed, the ensemble is reset all the time.
- If $1 \leq \tau_{mat} < N$ and $\tau = 1 \implies 1 \leq d < N$. A subset of the ensemble is removed.

By always resetting the whole ensemble (case (a) above), the experts generally don't get the chance to learn a stable representation of the underlying target concept. By removing new experts (case (b) above), the old outdated experts are not removed after a concept change while young experts don't have the chance to learn the new concept, impeding plasticity.

#### 4.2.4.3 A new deletion strategy based on a stochastic mechanism

One can soften the "eliminate the worst learner" strategy while still favoring the removal of the worst and potentially obsolete base learners. It suffices to pick randomly an expert from the subset of the $ds$ ($ds < N$) worst experts. In this way, the newest experts have a chance to learn enough of the regularities of the current environment to enter the pool of the top experts, and, at the same time, preserving the best performers. This promotes the plasticity of the system while not deteriorating its stability. The size $ds$ of the subset where experts can be picked up to be eliminated controls the plasticity-stability trade-off.

We studied the effect of five deletion sizes (by setting the value of $ds$) on the forgetting strategy: $N$, $0.75 * N$, $0.5 * N$, $0.25 * N$ and 1. The deletion size of 1 corresponds to the *replace the loser* strategy (i.e. replace the worst expert), while a deletion size of $N$ means that an expert is selected randomly for deletion from the ensemble of experts.

**Experimental setup**

We evaluated the different deletion sizes on the *Line*, *SineH* and *Circle* artificial problems suggested by Minku et al. [69] which include 27 datasets with various types of concept changes (different severity and speed levels). One concept change event occurs at time step $t_{drift} = 1,000$ and the total number of examples is equal to $2,000$ (see Section 2.3.5.1).

In the experimental setup, the ensemble comprised either $N = 10$, $N = 20$ or $N = 30$ base learners. Unlike ensemble methods that combine weak learners (classifiers with a prediction accuracy slightly better than random guessing) to form a strong learner [33], we do not necessarily require relatively large ensembles to adapt to concept changes. The learners in the experiments were not designed to be weak à la boosting [33] and size values between 10 and 30 are common in such context [70, 85].

In order to be compared, base learners were evaluated on the most recent $\tau_{eval} = 20$ data points (time steps). The duration for maturity $\tau_{mat}$ and the deletion period $\tau$ were equally set to 20 time steps. As for the global prediction, it merely uses the prediction from the current best base learner. This simple configuration means that every 20 time steps, all base learners are mature and are evaluated on their last 20 predictions. According to the evaluation record and the deletion strategy, a learner is selected for deletion and is replaced by a new base learner with no memory of the past.The parameter values were fixed throughout all our experiments on Minku's artificial problems *Line*, *SineH* and *Circle*. We didn't optimize the parameter values because we wanted to see the capacity of the deletion strategies to adapt to the different types of concept changes starting with the same initial settings.

As with the majority of ensemble methods adapting to concept changes [13, 59, 70, 85], we used one type of learning models in the same ensemble. Note that, in such case, the diversity in the ensemble comes from the deletion strategy which, by introducing new learners with no knowledge of the past, allows the presence of learners trained on different window sizes of data. We first used decision trees as base learners for *SineH*, *Line* and *Circle*. To make sure that the results were not biased by a particular type of

base learners (decision trees here), we repeated the experiments using different types of learning models[1], more specifically:

- perceptrons for the *Line* problem.
- two-layer feed-forward neural networks with non-linear 3 hidden neurons for the *Circle* problem.
- incremental support vector machines [26] with a radial basis function for the *SineH* problem.

The configuration of the different learning models (the structure of the neural network, the kernel function of the support vector machines) were chosen according to preliminary experiments. For each problem, we evaluated the online predictive accuracy of the learning model (a single learner) when trained on the first examples from the stream, that is, before a drift occurs. We then retained the best settings for each problem. The predictive accuracy using the chosen learning models was around 0.83% in average in the preliminary experiments.

We show in Figure 4.1 the mean classification error using the different deletion sizes. Each of the artificial problems consists of 9 datasets (3 severity levels and 3 speed levels). The mean classification error is thus averaged over 3 problems $*$ 9 datasets $*$ 3 committee sizes $*2$ types of experts per problem $= 162$ experiments for each deletion size. All the experiments start with the same random seed so that, for each problem, we have the same experts at the beginning of the experiments. The average error is split into three consecutive periods $I_i$:

1. before the drift: $I_1 = [1, t_{drift}]$, where $t_{drift}$ is the number of time steps before the concept change starts to occur. In the above problems, $t_{drift} = 1,000$.
2. during the drift: $I_2 = [t_{drift} + 1, t_{drift} + \delta_{drift}]$. The drifting time $\delta_{drift}$, as mentionned previously, equals to 1, $0.25 * t_{drift}$ and $0.5 * t_{drift}$ for speed levels $High$, $Medium$ and $Low$, respectively. For the high speed level, the drifting period is equal to 1. Since one time step is not enough in the average computation, we compute the average on $0.1 * t_{drift}$ time steps, instead of $\delta_{drift} = 1$.
3. after the drift: $I_3 = [t_{drift} + \delta_{drift} + 1, 2 * t_{drift}]$, where $2 * t_{drift}$ is the total number of instances in the stream.

We show in Figures 4.2, 4.3 and 4.4 the online predictive peformance of the ensemble on the $SineH$, $Line$ and $Circle$ problems, respectively, when $N = 10$ or 20.

[1] Other learning models could do as well

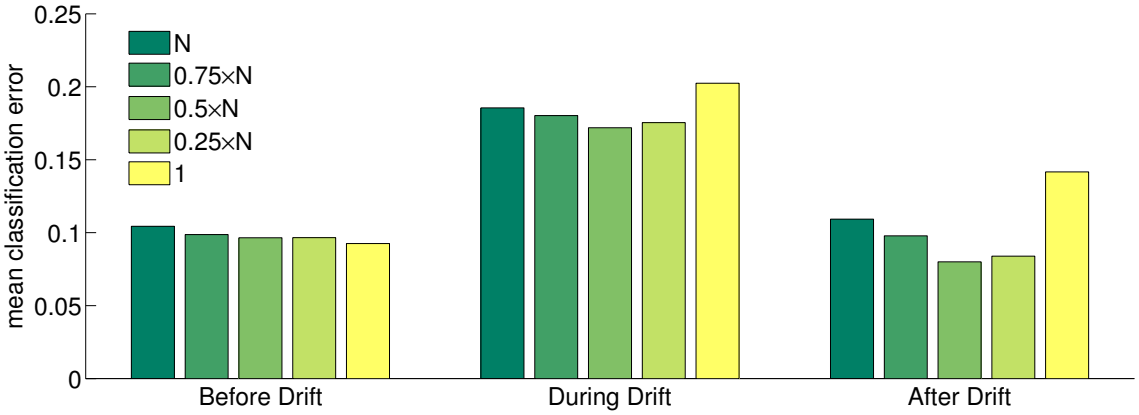


FIGURE 4.1: The mean classification error using different deletion sizes. The error is averaged over several experiments on the *Line*, *SineH* and *Circle* datasets.

In order to better visualize the reactivity of the ensemble to the concept change, the online performance is reset to 0.5 (the chance prediction rate) at time step $1,000$, that is, when the concept change occurs in case of a sudden change or when it starts happening if the change is continuous. The pair ($Sp$,$S$) on the top of each plot denotes the speed and severity levels of the concept drift, which ranges from *Low*, to *Medium* and *High*. The $i$-value in $ds(i)$ refers to the deletion size.

### Empirical results

From Figure 4.1, we notice the following:

### Case of a stationary environment

The deletion size $ds = 1$ gives the best predictive accuracy when learning in a stationary environment before any drift happens. The experts trained on the smallest windows of training examples are generally the ones that tend to be removed from the ensemble since they perform poorly compared to experts that have benefited from a large training set. Meanwhile, the remaining experts tune up their knowledge of the current concept, improving their classification record.

By increasing $ds$, the probability of removing a relatively good expert is also increased which hurts the classification accuracy of the ensemble. We might expect a catastrophic predictive accuracy in the extreme case, when $ds = N$. However, the accuracy is only hurt to some extent. In fact, since in our settings, the expert with the highest evaluation record predicts the target value of an incoming instance, only one "good" expert is required in the ensemble to give correct predictions. Removing all the experts in the pool requires

at least $N * \tau$ time steps. By this time, the first deleted expert is replaced by a new expert trained on $N * \tau$ training data, a size that is enough for the expert to have a relatively good predictive accuracy on these artificial problems. In case of a weighted vote, we expect the predictive accuracy to be hurt more, since the presence of a relatively large number of "bad" experts, make them win the vote over the few "good" experts.

**Case of a concept drift**

While $ds = 1$ gives the best predictive accuracy when the concept is stable, it is not the case after the concept drift. In fact when the drift severity is relatively small, the predictive performance of old experts won't decrease significantly after the drift, making it difficult for new promising experts to reach a higher predictive performance at the time of a deletion operation. Hence, new experts will likely be deleted, impeding the adaptation to the new concept. This problem is not encountered with high severity and high speed drifts since all experts will have a relatively bad predictive record after the drift and thus there won't be a preference over which expert to keep or to remove.

A relatively large deletion set increases the probability of a newly added expert to survive a deletion. It also allows one to remove all the experts from the ensemble after a concept change. Thus, for maximum plasticity, the best deletion size is $ds = N$. A maximum plasticity is beneficial only when the change is sudden and severe because in such case, all the old experts should be removed. In other cases, we would rather keep some of the old experts for knowledge transfer.

The experiments suggest that $ds$ should be small enough for stability and large enough for plasticity. With a minimum deletion size ($ds = 1$), the ensemble has the lowest classification error before the drift because *stability is favored over plasticity*. With a maximum deletion size however ($ds = N$) the ensemble *favors plasticity over stability* which hurts the classification performance when learning stationary concepts.

Choosing a deletion size that is half the size of the ensemble ($ds = 0.5 * N$) seems to correspond to a satisfactory trade-off between plasticity and stability. This choice gives the lowest classification error in average, before and after the concept drift in our experiments.

**Further analysis**

To see the details of the different deletion strategies, we analyze the results in Figures 4.2, 4.3 and 4.4. We notice that when the speed of change is slow, deleting randomly from all the committee or deleting among the $ds$ worst experts is nearly the same during the drifting period. In fact, this period is characterized by the need to classify examples belonging to the old and new concepts. Examples from the new concept will be generally misclassified by experts trained on the old concept, resulting in perturbated weights values, moving up and down depending on how many examples the experts misclassified in the last $\tau_{eval}$ time steps. Hence, there won't be a clear boundary between the "good" and the "bad" experts and the experts in the deletion set will change with each deletion operation. As a result, $ds = 1$ and $ds = N$ will have the same impact on the classification performance for some time until the new concept starts to take over the old one and the weight values become more expressive.

We also notice that the results on $SineH$ and $Line$ are quite similar, with only $ds = 1$ causing difficulties in the adaptation to the concept change. This suggests that for $SineH$ and $Line$, the maturity age $\tau_{mat}$ is small and unadapted to the learning problem. In fact, the youngest expert is always removed as 20 time steps (training samples) are not enough to outperform its rivals. However, once the deletion size gets larger ($ds = 2$), the young experts have a higher chance of surviving deletion, overcoming the unadapted parameter values. We would expect that relatively large deletion sizes would hurt the adaptation ability of the ensemble. However, it seems that even when any expert can be subject of removal, the ensemble is still able to provide a relatively good predictive accuracy. Resetting the whole ensemble needs at least $\delta_{reset} = N * \tau_{mat}$ time steps. Hence, a possible explanation is that capturing the underlying target concept requires less than $\delta_{reset}$ time steps. Thus, even if the ensemble is always reset, there will always be an expert trained on enough examples to provide good prediction results.

With $Circle$, the behavior of the predictive performance is different than $SineH$ and $Line$ depending on the deletion size. Even with $ds = 2$ (see Figure 4.4 top) the ensemble has difficulties adapting to the change, suggesting that $Circle$ is a more complex learning problem, requiring an even larger maturity age for young experts to learn the new concept and outperform their rivals. With $ds = 5$, the ensemble gives the best results. However, once the deletion size gets larger, the predictive performance is hurt. Unlike $SineH$ and $Line$, removing a good expert is rather costly to the algorithm since an expert requires more training data to capture the underlying target concept.

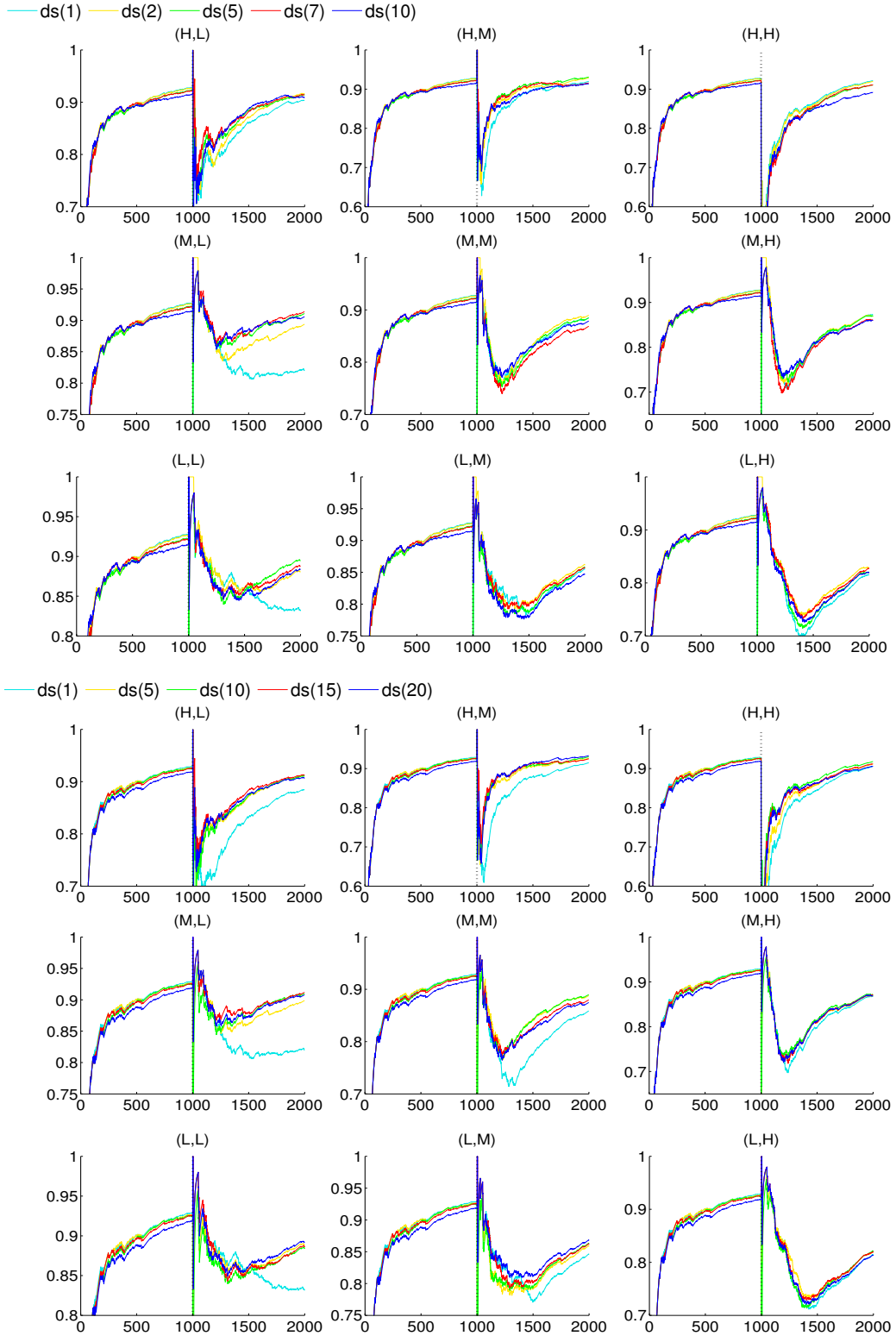


FIGURE 4.2: The online performance on the databases of the *SineH* problem using different deletion sizes and using support vector machines as base learners, with N=10 (top) and N=20 (bottom). The $x$-axis represents the training examples (time step) and the $y$-axis represents the online classification performance (the percentage of correctly classified instances so far). The online performance is reset at time step $1,000$ when the concept changes.

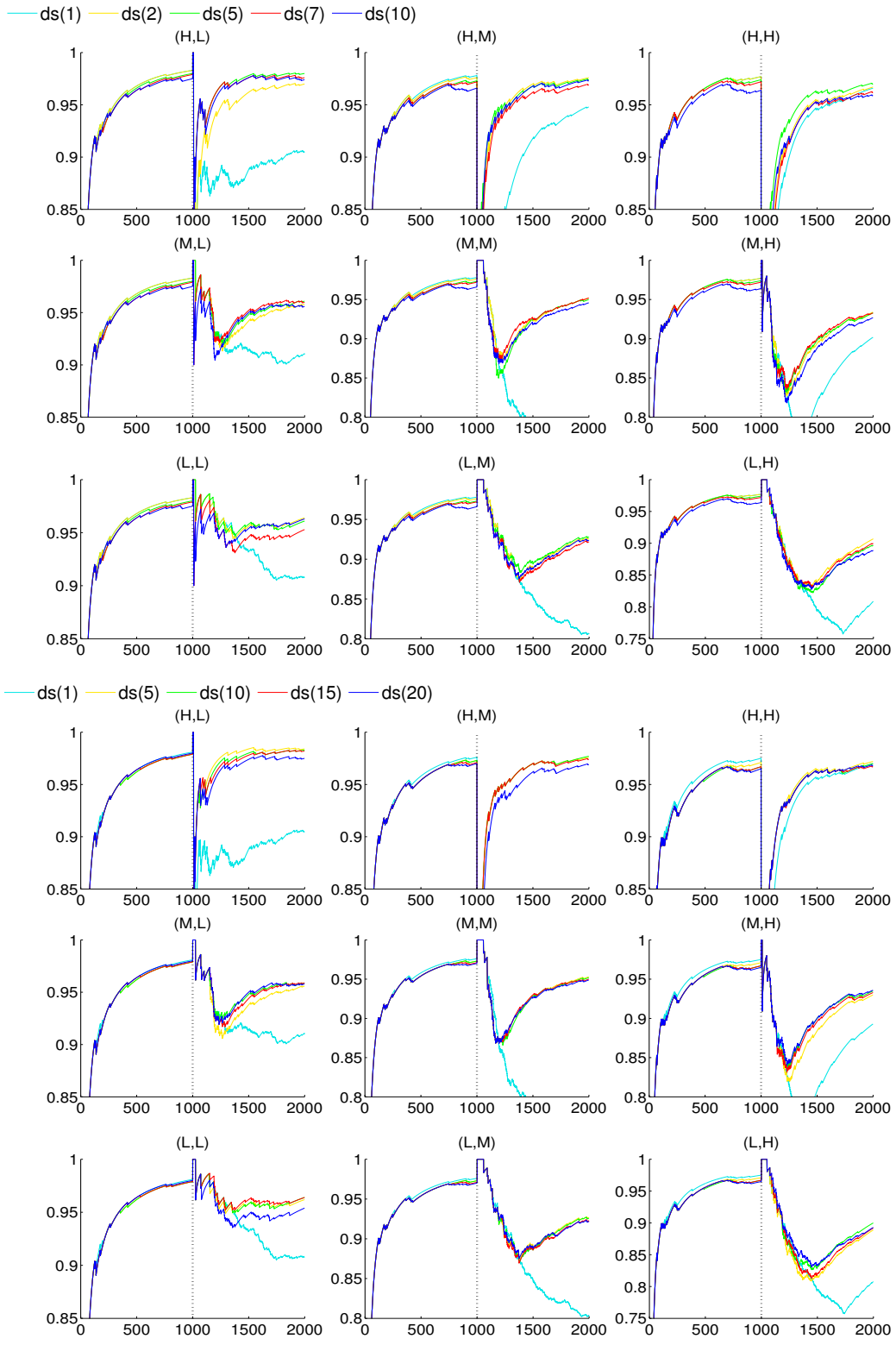


FIGURE 4.3: The online performance on the databases of the *Line* problem using different deletion sizes and using decision trees as base learners, with N=10 (top) and N=20 (bottom). The $x$-axis represents the training examples (time step) and the $y$-axis represents the online classification performance (the percentage of correctly classified instances so far). The online performance is reset at time step $1,000$ when the concept changes.

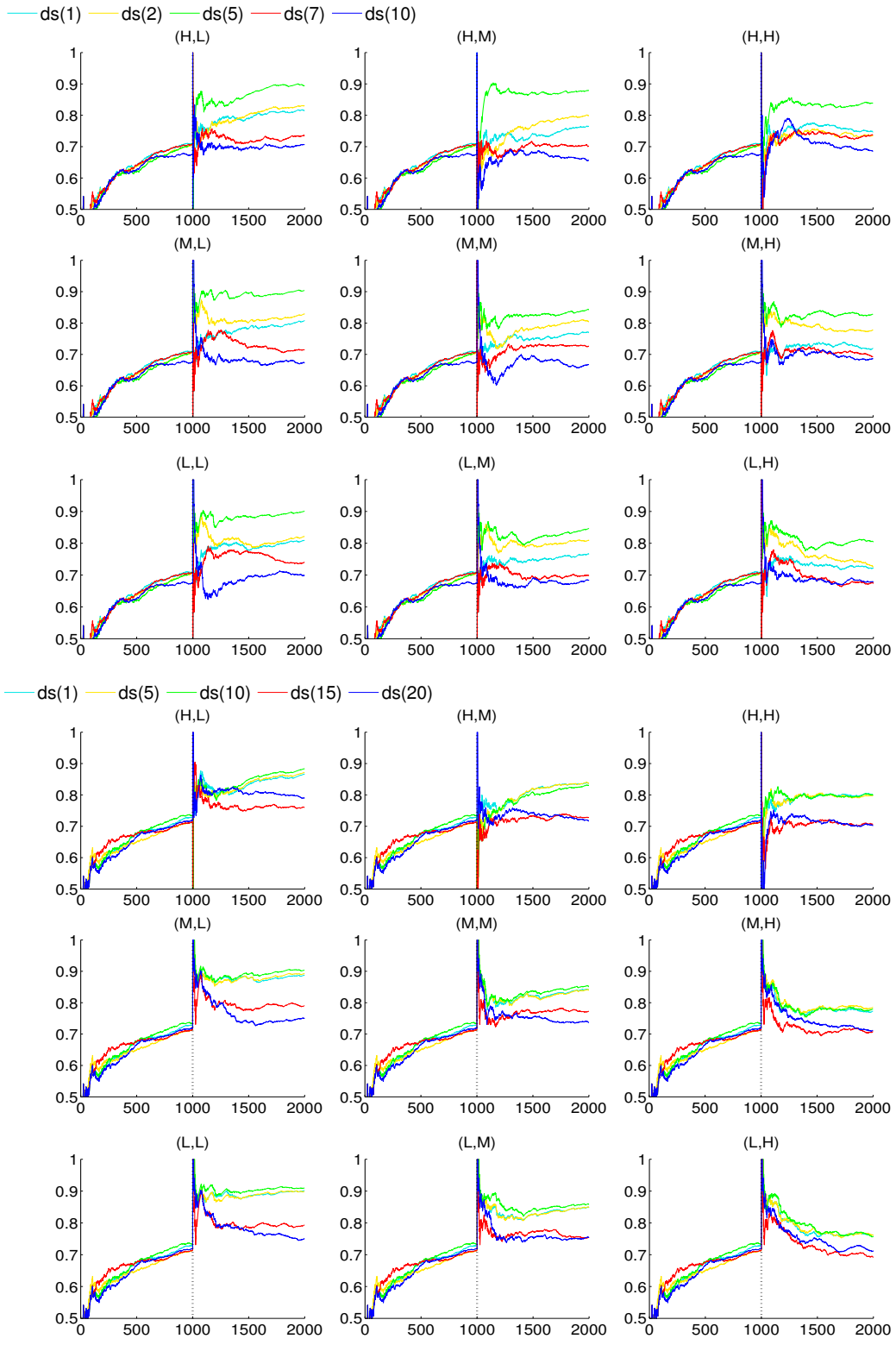


FIGURE 4.4: The online performance on the databases of the *Circle* problem using different deletion sizes and using neural networks as base learners, with N=10 (top) and N=20 (bottom). The $x$-axis represents the training examples (time step) and the $y$-axis represents the online classification performance (the percentage of correctly classified instances so far). The online performance is reset at time step $1,000$ when the concept changes.

## 4.3 DACC

Based on the previous analysis, we suggest an ensemble algorithm that adapts implicitly to concept changes using an ensemble of experts. DACC's algorithm, which is a main contribution in this work, stands for *Dynamic Adaptation to Concept Changes*.

We present its main components:

1. The committee of predictors
2. The committee's evolution
3. The weighting functions
4. The committee's final prediction
5. Processing training examples
6. Time & memory constraints
7. Computational complexity

We then study different aspects of DACC, namely:

7. The diversity levels in the committee
8. The stability-plasticity dilemma
9. The effect of the preset parameters on the predictive performance
10. Recommendations on the choice the preset parameters

Finally, we evaluate DACC against a number of online algorithms on the artificial, semi-artificial and real datasets presented in Chapter 3. DACC's pseudo-code is shown in Algorithm 1.

### 4.3.1 The committee of predictors

Instead of adding one expert at a time, DACC can add differents types of experts or what we call a *pool of types of experts*, in order to create more diversity in the committee. The experts are asked to predict the label of an example in a classification task.

Experts can be:

- Different learning models. For instance, decision trees, neural networks, support vector machines, among others.
- Learning models with different structures. For instance, neural networks with different number of neurons, layers, etc...
- Learning models with different training functions.
- Prediction rules. For instance, a rule that always predicts the last observed class, or the majority class.

We define the maximum number of pools as $N_{pool}$ and the maximum number of predictors in the committee as $N$. Hence,

$$N = N_{pool} \times size(pool) \tag{4.1}$$

### 4.3.2 The committee evolution

The committee is initially empty. At each time step, a new training data $e = (\mathbf{x}, y)$ is received and a new pool of types of experts, trained on $e$ is added to the committee. The predictors that are already in the committee are also trained incrementally on the new training data. This operation is repeated until the committee reaches its maximum size $N$. After reaching its maximum size, the committee is updated by the deletion strategy which deletes and adds new experts.

Instead of removing the worst expert(s) of the ensemble, as with the *replace the loser* strategy, DACC's deletion strategy selects randomly $m$ members from *the worst half of the committee* and forces them to retire. To keep the committee size fixed, $m$ is equal to the size of the pool of types of experts (i.e. $m = size(pool)$).

Deleting experts occur as follows. With each deletion operation, each predictor is evaluated and a *weight* is assigned accordindly. The higher the weight, the better the predictor. The predictors are then split into two sets:

- $H_{best}$ represents the best half of the predictors, with the highest weights. The predictors in $H_{best}$ are protected from deletion.
- $H_{worst}$ represents the remaining predictors. DACC selects randomly $size(pool)$ predictors from $H_{worst}$ for deletion.

In case of a concept change, this deletion strategy increases the expectation of a young expert $h_{new}$ with still a relatively bad performance record to survive in the committee, allowing it to further improve its predictive performance regarding the new concept, before being evaluated for the next deletion.

In order to control the rate of deletion, we impose all the experts to be mature before expelling an expert from the committee. An expert is mature if it has observed at least $\tau_{mat}$ training instances. Thus, when a predictor is deleted, the committee waits until the newly added predictor becomes mature before the next deletion operation. This condition ensures that a period of $\tau = \tau_{mat}$ time steps separates two consecutive deletions.

An expert $h_{bad}$ belonging to the worst half of the committee survives a deletion operation with probability

$$p = \frac{N/2 - size(pool)}{N/2} \tag{4.2}$$

Each time $h_{bad}$ escapes deletion, it is given another $\tau_{mat}$ time steps of training data before the next deletion operation. The expectation of $s$, the number of times $h_{bad}$ survives a deletion operation, is:

$$\begin{aligned} E[s] &= \sum_{m=1}^{\infty} mp^m(1-p) = p(1-p)\sum_{m=1}^{\infty} mp^{m-1} \\ &= p(1-p)\frac{\mathrm{d}}{\mathrm{dp}}\left(\sum_{m=1}^{\infty} p^m\right) = p(1-p)\frac{\mathrm{d}}{\mathrm{dp}}\left(\frac{p}{1-p}\right) \\ E[s] &= \frac{p}{(1-p)} \end{aligned}$$

By replacing $p$ with its value from equation 4.2, we get:

$$\begin{aligned} E[s] &= \frac{N/2 - size(pool)}{size(pool)} \\ &= \frac{size(pool) \times N_{pool}/2 - size(pool)}{size(pool)} \\ E[s] &= \frac{N_{pool}}{2} - 1 \end{aligned} \tag{4.3}$$

Increasing the life expectancy of the relatively bad experts makes DACC less exposed to cases where young experts are expelled from the ensemble because they didn't get enough time to improve their predictive performance. In other words, the suggested deletion strategy is less sensitive to a high deletion rate than the "replace the loser" strategy. A higher deletion rate entails a faster reactivity to a potential concept drift.

It is important to note that resets will happen all the time, even for stationary datasets, but this behaviour should not have a negative impact on the committee's predictive performance. We show here an example, clarifying how the deletion strategy allows the committee to *adapt* to a variety of concept changes while being able to learn a *stable* concept.

**Example**

Let's consider one concept drift event that occurs at time step $t_{drift}$. The training data from time step $t_1$ to $t_{drift}$ are generated for concept $C_{old}$. Then, a sudden concept change with a class severity of 20% occurs in one time step, replacing the old concept $C_{old}$ with $C_{new}$. As a result, 20% of the input space have modified labels in $C_{old}$ according to the new concept $C_{new}$.

**Stability** When learning $C_{old}$, the two sets $H_{worst}$ and $H_{best}$ become rapidly stable. The predictors in $H_{best}$, being protected from deletion, will have their classification performance improved with time as new training data are received and learnt. The improved performance will be translated into larger weights which will further keep them in $H_{best}$. When a predictor $h \in H_{worst}$ is removed, it is replaced by a new predictor $\tilde{h}$. As pointed previously, a deletion operation requires all the predictors to be mature. Thus, the next deletion operation occurs $\tau_{mat}$ time steps later. The preset value of $\tau_{mat}$ might not be enough for $\tilde{h}$ to outperform the classification performance of any of the predictors in $H_{best}$. As a result, the new predictor $\tilde{h}$ remains in $H_{worst}$ with a probability of $size(pool)/size(H_{worst})$ to be removed with each deletion operation. The predictors in $H_{worst}$ that escaped deletion for a relatively long time will have a good classification performance but will still remain in $H_{worst}$ since $H_{best}$ cannot contain more than half of the committee.

The predictors in $H_{best}$ accumulate knowledge about $C_{old}$ with each newly processed training data. They assure the learning of a *stable* concept. The predictors in $H_{worst}$ change a lot due to deletion operations; they are replaced by new predictors ready to react to a concept change. The predictors in $H_{worst}$ assure the *plasticity* of the committee.

**Plasticity** After the concept drift, the classification performance of the predictors in $H_{best}$ starts declining. Even if, after time step $t = t_{drift}$, they will learn training data from $C_{new}$, it is not easy to forget the acquired knowledge from concept $C_{old}$. The decrease in their performance depends on the severity of the change: the larger the change, the higher the damage.

The classification performance of the predictors in $H_{worst}$, on the other hand, starts improving since they yet haven't learnt any stable concept. After enough training examples, some of the predictors in $H_{worst}$ will outperform other predictors in $H_{best}$. When the classification performance of a predictor $h_w \in H_{worst}$ becomes higher than the one of the worst predictor $h_b \in H_{best}$, $h_w$ and $h_b$ switch places which makes $h_w$ immune against removal. This will protect $h_w$ from deletion and will allow it to learn furthermore the concept $C_{new}$. With at least one predictor that learns the new concept $C_{new}$, the committee is able to adapt to the concept change when using the MAX combination function, as we will see shortly in Section 4.3.4.

### 4.3.3 The weighting functions

At each time step, the experts are evaluated on their recent predictive performance and a weight is assigned accordingly.

In DACC, the weight $w_h$ of an expert $h$ represents its mean predictive performance over the last $\tau_{eval}$ predictions. We chose this weight measure for the advantages presented in Section 4.2.3. Accordingly, the *weight* of an expert $h$ at time step $t$ is computed as follows:

$$w_h(t) = \begin{cases} \sum_{j=1}^{\tau_{eval}} acc_h(\mathbf{x_{t-j}})/\tau_{eval} & \text{if } h \text{ is mature} \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

where

$$acc_h(\mathbf{x_{t-j}}) = \begin{cases} 1 & \text{if } \tilde{y}_{t-j,h} = y_{t-j} \\ 0 & \text{if } \tilde{y}_{t-j,h} \neq y_{t-j} \end{cases} \tag{4.5}$$

The evaluation window $\left[(\mathbf{x_{t-\tau_{eval}}}, y_{t-\tau_{eval}}), (\mathbf{x_{t-\tau_{eval}+1}}, y_{t-\tau_{eval}+1}), \ldots, (\mathbf{x_{t-1}}, y_{t-1})\right]$ contains the $\tau_{eval}$ most recent training examples. In the above equations, the pair $(\mathbf{x_{t-j}}, y_{t-j})$ denotes the $j$-th most recent training example; $\tilde{y}_{t-j,h}$ is $\mathbf{x_{t-j}}$'s label as predicted by the expert $h$ at time step $t-j$, just before $h$ was trained on the pair $(\mathbf{x_{t-j}}, y_{t-j})$. In DACC, only mature experts are allowed to contribute to the final prediction. Hence, the weight of an expert is set to zero if unmature.

### 4.3.4 The final prediction

The committee predicts the label $\tilde{y}$ of an incoming example $\mathbf{x}$. The label of $\mathbf{x}$ is first unknown to the committee. It is just after the committee predicts its label $\tilde{y}$ that the real label $y$ is revealed.

In order to construct the committee's final prediction $\tilde{y}$, we first compute the *weight* of each expert in the committee. Then, according to the weight values, the predictions of the experts on the current received instance $\mathbf{x}$ are combined using the *cmb* function. An empirical comparison between the different combination functions (V, WV, WVD and MAX) presented in Section 4.2.2 showed that simple voting (V) gives the highest online error since it doesn't take into account the goodness/badness of the predictor, while WV leads to a fall in the predictive accuracy of the ensemble shortly after the drift [48]. This was explained by the fact that when a committee contains a relatively large number of bad experts, their weights will sum up and will be high enough to win the vote.

In DACC, we avoid using a simple vote (V) or a weighted vote (WV) as a combination function. Instead, we rely on the two combination function: MAX and WVD.

#### 4.3.4.1 MAX

With MAX, the member with the highest weight is selected for prediction. In case of ties, the members sharing the same highest weight vote for the final prediction. When using MAX, it is not necessary to remove a large number of experts in order to adapt to a concept change. Once a fresh expert learns enough training data from the new concept and its weight becomes higher than the others, it is selected to predict on the behalf of the committee. Therefore, this combination function entails a faster reactivity to a concept change than this of a simple weighted vote.

To show more how this function combines the predictions of the committee members, we tested DACC on the artificial problems by Minku (see Section 2.3.5.1): *Line*, *Boolean*, *SineH* and *Circle*, using MAX as a combination function. We used the same configuration as before, that is, $\tau_{mat} = \tau_{eval} = 20$. For simplicity, only one type of base learners is used in the same ensemble. Hence, $size(pool) = 1$ and $N = N_{pool}$ according to Equation 4.1. The ensemble comprised a total of $N = 10$ base learners.

In order to show that our ensemble method is general, in the sense that one could use any online learning algorithm as a base learner, we used different types of learning models for the different problems. Again, we used one type of learning models in the same ensemble. Hence, the diversity in the ensemble (resulting from the presence of learners

with different memories of the past) is due to the frequent addition of new learners. Note that higher levels of diversity could be realized by mixing different types of learners in the same ensemble, leading possibly to a higher predictive performance. However, our goal here is to explain the behavior of the MAX function and not to maximize DACC's predictive accuracy. Hence, for sake of simplicity, one type of learning models were used in the same ensemble, more specifically:

- decision trees for the *Boolean* and *Circle* problems.
- support vector machines with a radial basis function for the *SineH* problem.
- perceptrons for the *Line* problem

The learning models were chosen according to preliminary experiments[2] where for each problem, we evaluated the online predictive accuracy of the learning model (a single learner) when trained on the first examples from the stream (before a drift occurs). We then retained the best settings for each problem.

We plot in Figures 4.5 and 4.6, the maximum weight in the committee at each time step along with the percentage of predictors selected for the committee's final prediction. With an evaluation window of size $\tau_{eval} = 20$, we get 21 different values for the weight, ranging from 0 when all of the 20 predictions are wrong, to 1 in the ideal case with no errors. In the plots, the 21 values levels of the weights are mapped into the interval [0 100]. We notice the following:

Using our MAX combination function doesn't mean that most of the time, one single expert will be selected for prediction. Actually, all the predictors that have the same number of correctly classified instances over the last $\tau_{eval}$ time steps will have the same weight. Thus, sharing the highest weight is not a rare case.

When the change is *sudden* (see first rows in Figures 4.5 and 4.6), we notice that, just after the drift, the number of classifiers (experts) selected for the final prediction decreases, before re-increasing. In fact, the first deleted classifier after the drift improves with time as training data from the new concept is learnt. Once its weight becomes higher than the others, it is selected for the final prediction. As time goes by, more outdated classifiers will be replaced. The predictive performance of new classifiers will improve with time, increasing their weight. New classifiers that reach the highest weight cooperate by voting for the final prediction.

When the change is *continuous* (see second and third rows in Figures 4.5 and 4.6), we see that the weights go up and down during the transition between the consecutive concepts.

[2]Other learning models could do as well

Since classifiers are asked to predict labels for both the old and the new concept during the transition, there won't be a specific classifier whose performance is clearly better than the rest. There will most likely be a vote between "medium" performing classifiers, that is, old experts trained on instances from the old concept and new experts that start learning instances from the new concept. The hope is that combining votes from both parties (old and new experts) during the transition period will give more accurate predictions than using predictions of the old (or new) experts only.

#### 4.3.4.2 WVD

With WVD, a weighted vote is performed, but after suppressing the predictions of the members whose weights fall into the lower half of the weights interval. While WVD can lead to a slightly slower reactivity compared to MAX, it is more robust in case of noise [48]. In both MAX and WVD, an unmature expert does not contribute to the final prediction. It is assigned a weight of zero.

### 4.3.5 Processing training examples

When an example $(\mathbf{x}_t, y_t)$ is received, it is passed to each expert in the ensemble. The example is processed by an expert $h$ using the *process* function, usually a training function that updates the expert's hypothesis, incrementally with $(\mathbf{x}_t, y_t)$.

Nevertheless, it is also possible to decide another type of processing that updates experts differently. That is, when choosing the *pool of types of experts*, we can also choose the type of processing related to each defined expert. For instance, we may decide that the *process* function of a particular type of experts does not revise the expert's hypothesis. This allows the presence of experts in the ensemble that give random predictions.

### 4.3.6 Time & memory constraints

In order to meet the computational requirements and the memory constraints of online learning, the *train* function updates the hypothesis of a learning model or more generally, an expert, incrementally, based on the last received example $(\mathbf{x_t}, y_t)$. After updating all the experts, $(\mathbf{x_t}, y_t)$ is no longer needed and is therefore discarded from the memory. A large ensemble size impacts the memory utilization as well as the time required to update the experts. Limiting the ensemble size is a straightforward solution. Nevertheless, since experts are evaluated and updated independently of each other, we might also consider parallelizing DACC's algorithm.

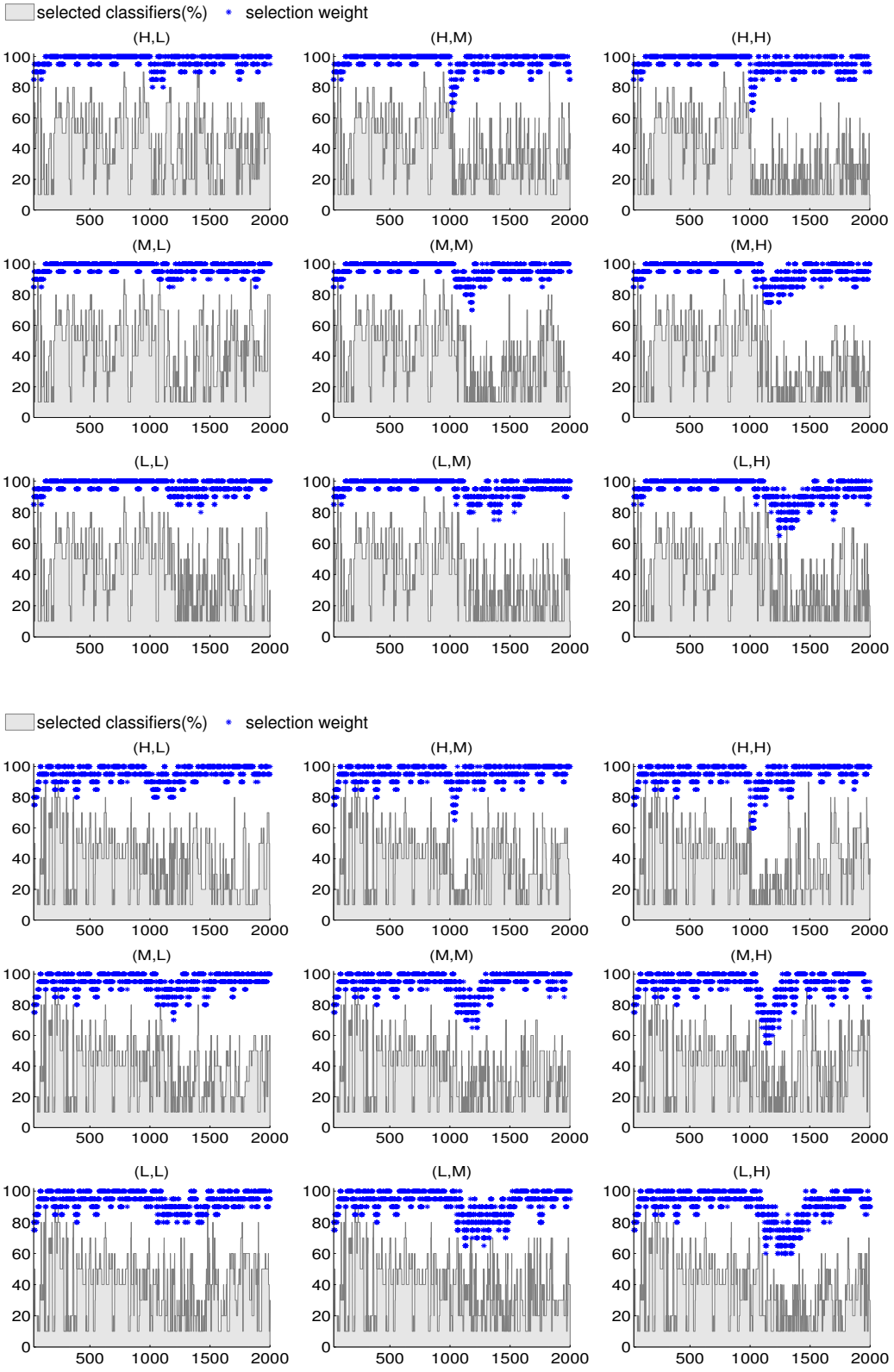


FIGURE 4.5: DACC: We show in blue the maximum weight in the committee and in gray the percentage of classifiers selected for the committee's final prediction when using the MAX function. Top: the 9 datasets of the *Circle* problem. Bottom: the datasets of the *SineH* problem.

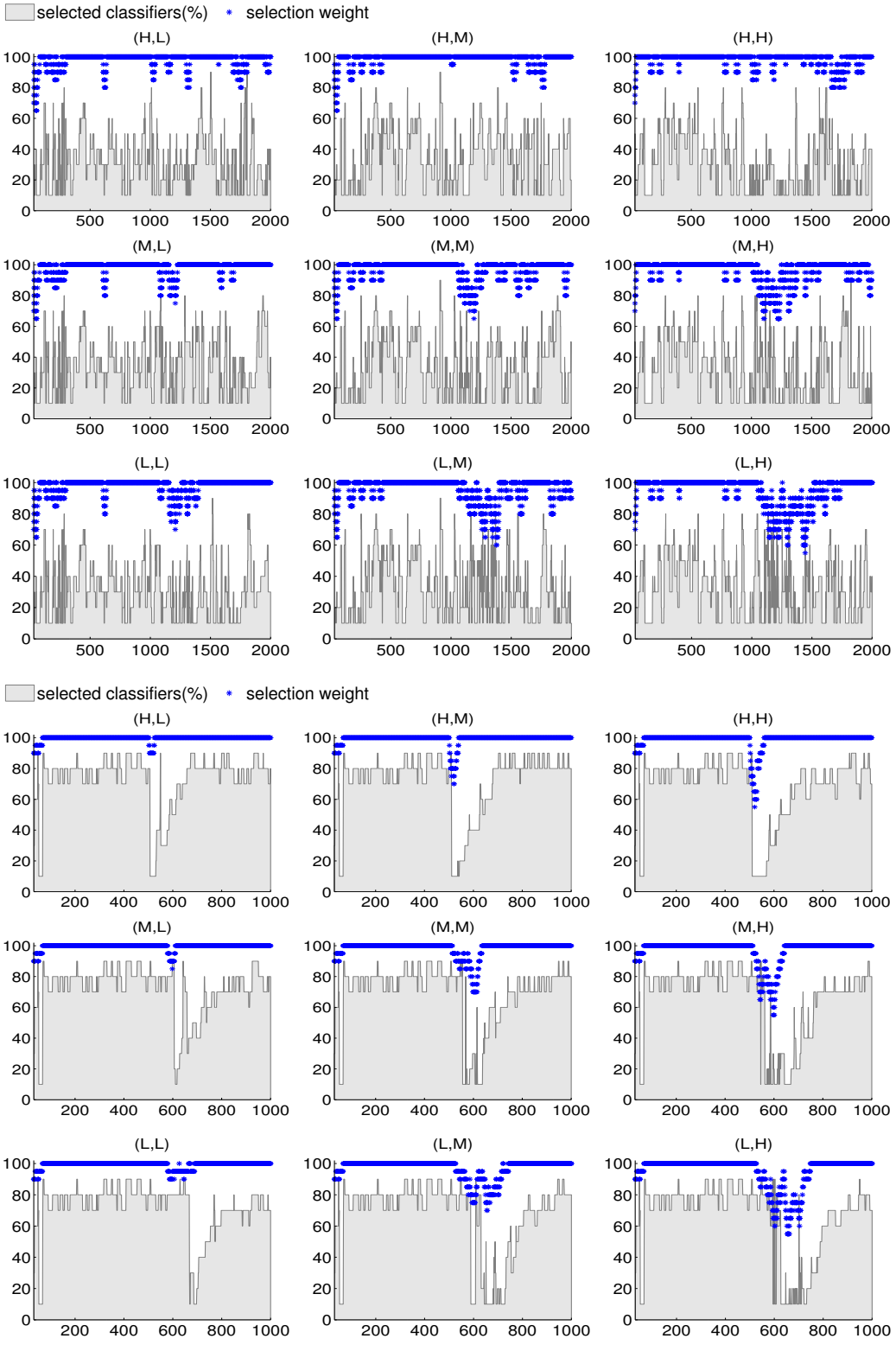


FIGURE 4.6: DACC: We show in blue the maximum weight in the committee and in gray the percentage of classifiers selected for the committee's final prediction when using the MAX function. Top: the 9 datasets of the *Line* problem. Bottom: the datasets of the *Boolean* problem.

**Algorithm 1:** DACC algorithm

**Input**: *Pool* the pool of types of experts, $N_{pool}$ the number of pools, $\tau_{mat}$ the maturity age, and $\tau_{eval}$ the number of instances on which the members are evaluated.

```
1  begin
2      C ← ∅;                                              /* Ensemble of experts */
3      for t = 1 to ∞ do
4          i_t = (x_t, y_t);
           /* ------------------------------------------------------------ */
           /* Predict target value                                         */
           /* ------------------------------------------------------------ */
5          for all h ∈ C do
6              w_h ← eval(h, τ_eval) ;                  ▷The weighting functions
7              ỹ_h ← h(x_t) ;                              ▷The final prediction
8              Ỹ ← Ỹ ∪ ỹ_h;
9              W ← W ∪ w_h;
10         end
11         ỹ_t ← cmb(W, Ỹ) ;                               ▷The final prediction
           /* ------------------------------------------------------------ */
           /* Train experts                                                */
           /* ------------------------------------------------------------ */
12         reveal real value y_t;
13         for all h ∈ C do
14             w_h ← eval(h, τ_eval) ;                  ▷The weighting functions
15             h ← process(h, i_t) ;                 ▷Processing training examples
16             age(h) ← age(h) + 1;
17         end
           /* ------------------------------------------------------------ */
           /* Delete experts                                               */
           /* ------------------------------------------------------------ */
18         if ∀h ∈ C, age(h) ≥ τ_mat then
19             C_w are the size(C)/2 experts with the lowest weights;
20             for k = 1 : size(Pool) do
21                 h_min ∈ C_w is randomly selected ;        ▷The committee evolution
22                 C ← C − {h_min};
23             end
24         end
           /* ------------------------------------------------------------ */
           /* Add experts                                                  */
           /* ------------------------------------------------------------ */
25         if size(C) < size(Pool) * N_pool then
26             for all h_new ∈ Pool do
27                 h ← process(h_new, i_t) ;             ▷Processing training examples
28                 age(h) ← 1;
29                 C ← C ∪ h;
30             end
31         end
32     end
33 end
```

### 4.3.7 Computational complexity

We discuss here the computational complexity of DACC, assuming the sequential implementation presented in Algorithm 1. DACC's main operations include:

1. Predicting the target value of an input $\mathbf{x}_t$ (lines 5-10)
2. Combining the individual predictions (line 11)
3. Training the experts on the training example $(\mathbf{x}_t, y_t)$ (lines 12-17)
4. Deleting experts from ensemble (lines 18-24)
5. Adding new experts to ensemble (lines 25-31)

It is easy to see that the complexity of steps (1) and (2) is O($N$), where $N$ is the size of the ensemble, because the prediction and training operations apply to each expert in the ensemble. Computing the weight value of each expert (lines 6 and 14) does not increase the complexity of steps (1) and (2), since the weight of an expert represents its predictive performance on the last $\tau_{eval}$ time steps and thus can be updated incrementally with each new prediction.

The complexity of step (3) depends on the type of combination function used. In case of MAX, finding the highest weight values takes O($N$) computations. It is important to note that finding a maximum value can take less than O($N$), for instance O($\log_2 N$) using a binary search algorithm [65]. However, in our MAX function, all experts sharing the maximum weight value should be retrieved, and thus a full scan of the weight values is required. In case of WVD, the weight values should be sorted first. This can be done using classical sorting algorithms, for instance, binary tree sort [4] and the likes, with a complexity average of O($N\log N$) for these algorithms.

In step (4), weight values are updated and the experts are removed randomly from the worst half of the ensemble. This entails sorting the weight values which, again, has a complexity of O($N\log N$). Then, size($Pool$) experts are removed from the ensemble, adding a complexity of O(size($Pool$)).

Finally, step (5) adds size($Pool$) experts to the ensemble and trains them on the last received example $(\mathbf{x}_t, y_t)$, leading to an additional complexity of order O(size($Pool$)). This step happens either when the ensemble hasn't yet reached its maximal size or as a result of step (4), that is when experts are removed from the ensemble and should be replaced with new experts. The former case occurs $N_{pool}$ times at the very beginning of the streaming process while the latter case repeats every $\tau_{mat}$ time steps (the rate of deletion) during the streaming process.

Overall, when a deletion operation is executed, the computational complexity of DACC is $O(3N + N \times \log N + 2 \times \text{ size}(Pool)) = \mathbf{O}(N \times \mathbf{log}N)$ when using MAX and $O(2N + 2N \times \log N + 2 \times \text{ size}(Pool)) = \mathbf{O}(N \times \mathbf{log}N)$ when using WVD. When experts are not removed from the ensemble, the computational complexity becomes $\mathbf{O}(3N) = \mathbf{O}(N)$ when using MAX and $O(2N + N \times \log N) = \mathbf{O}(N \times \mathbf{log}N)$ when using WVD.

### 4.3.8 Implicit diversity levels

According to [69], high diversity in the ensemble helps reduce the initial drop in accuracy that happens just after the concept change. When the concept is stable, however, low diversity gives more accurate results. In this section, we aim to see if DACC's deletion strategy simulates such behavior, creating different levels of diversity depending on whether the concept is stable or changing.

#### 4.3.8.1 Experimental setup

As a measure of diversity, we compute the kappa statistics [19]. This measure evaluates the degree of agreement between the predictions on a set of items by two base predictors or classifiers [3]. Let's consider classifiers $h_1$ and $h_2$, each classifying $m$ items into one of $L$ classes, and a contingency table where cell $C_{ij}$ represents the number of instances $\mathbf{x}$ that $h_1$ classifies as class $i$ and $h_2$ as class $j$. We define $\theta_1$ as the observed agreement among the two classifiers:

$$\theta_1 = \sum_{i=1}^{L} \frac{C_{ii}}{m}$$

We also define $\theta_2$, the hypothetical probability of chance agreement, as follows:

$$\theta_2 = \sum_{i=1}^{L} \left( \sum_{j=1}^{L} \frac{C_{ij}}{m} \sum_{j=1}^{L} \frac{C_{ji}}{m} \right)$$

The kappa statistics is then defined as:

$$\kappa = \frac{\theta_1 - \theta_2}{1 - \theta_2} \tag{4.6}$$

In case of complete agreement, $\kappa = 1$. If there is no agreement other than what would be expected by chance, $\kappa = 0$.

[3] Other agreement statistics should do as well.

We consider a stream of instances that are received from time step $t_1$ until $t_{end}$ and one concept drift event occurs at time step $t_{drift}$. The transition between the old and new concepts lasts for $\delta_{drift}$ time steps.

In order to see how the levels of diversity evolve, we plot the Kappa-Error diagram, a scatterplot where each point corresponds to a pair of experts [12, 67]. The $x$-value is their kappa value and the $y$-value is the product of the classification error of each expert on a test set. The Kappa-Error diagram is plotted at the following time steps:

- just before the drift starts, $t_1 = t_{drift} - 10$
- just after the drift starts, $t_2 = t_{drift} + 10$
- halfway after the drift has started, $t_3 = t_{drift} + (t_{end} - t_{drift})/2$
- at the end of the experiment, $t_4 = t_{end}$

For $t_1$, the test set contains all the data before the drift i.e. the training data generated for the first stable concept. For $t_2$, $t_3$ and $t_4$, the test set is the remaining data.

As previously seen, DACC divides the ensemble into two sets with each deletion operation: the worst and best half of the committee, referred to as $H_{worst}$ and $H_{best}$, respectively. Without loss of generality, we assume that the pool of type of predictors contains one type of predictors only. Thus, $size(pool) = 1$, $N_{pool} = N$ and one predictor is removed with each deletion operation from $H_{worst}$.

#### 4.3.8.2 Empirical results

We show in Figure 4.7 the Kappa-Error diagrams of DACC's experts on the *SineH* artificial problem (see Section 2.3.5.1). The parameters of the ensemble are: $N = \tau_{eval} = \tau_{mat} = 20, cmb =$MAX and the experts are support vector machines with a radial basis function. We notice the following:

Before the drift, there are two distinct clouds of points: the first one has relatively high kappa values and low classification errors [4]. This cloud represents the experts in $H_{best}$ which have a *low diversity* and have learnt enough training data from the first concept. The other points have kappa values, they represent the experts in $H_{worst}$ who are removed frequently from the committee and thus have a *high level of diversity*.

Just after the drift, the classification error goes up for all the points since the experts have not yet forgotten the old concept and have not observed enough training data from the new one.

[4] In the subfigures of the first row, this cloud is hidden behind the red circles.

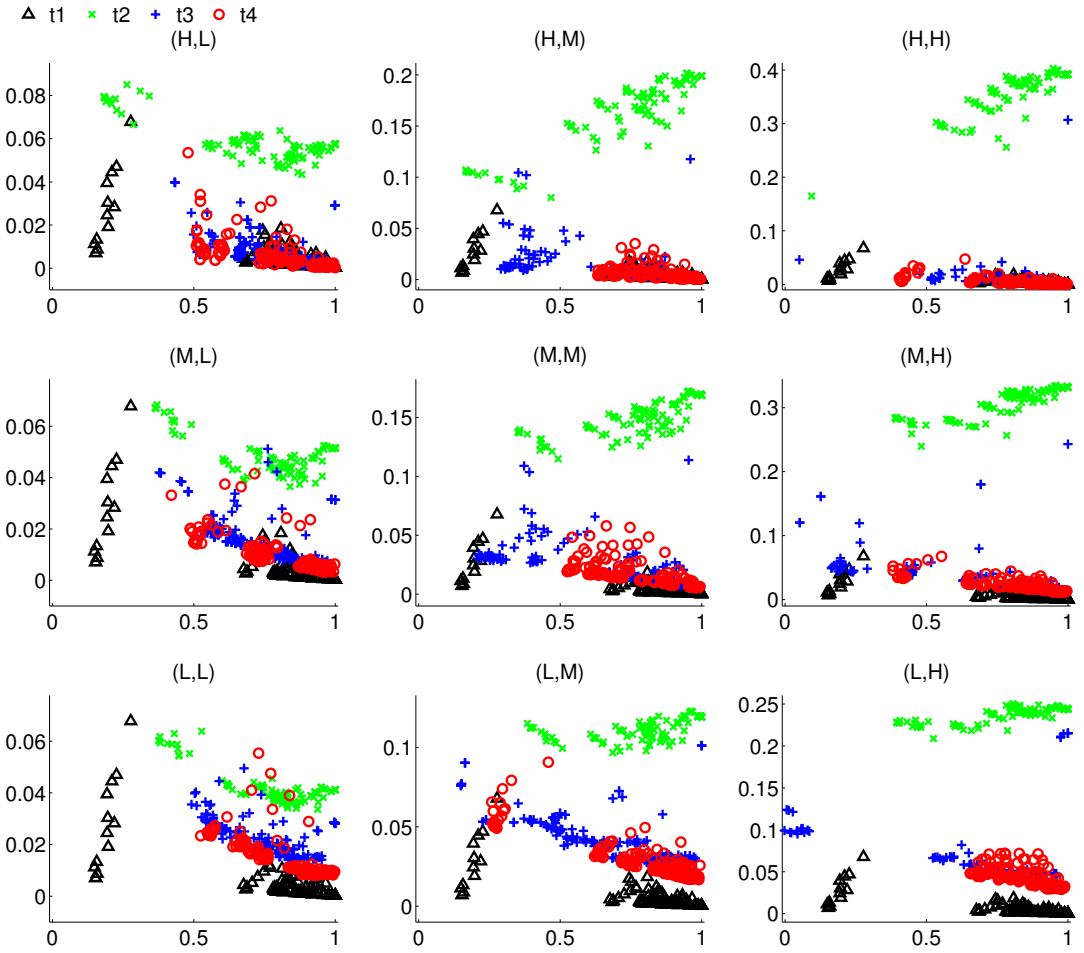


FIGURE 4.7: DACC: The Kappa-Error diagrams for the *SineH* problem with a committee of size 20 and a support vector machine (SVM) in the pool of predictors. The $x$-axis represents the kappa value of the pairs of classifiers and the $y$-axis is the product of their classification error on a test set.

Halfway after the drift, the diversity has increased in the ensemble. This is due to the deletion of experts, previously in $H_{best}$, that had learnt the old concept but were then deleted according to their classification performance on the new concept. The classification error decreases comparing to time step $t_2$ as the experts learn more training data from the new concept.

At the end of the experiment, the classification error has reached the same level as at time step $t_1$. The diversity has also decreased in the ensemble.

The different levels of diversity in the ensemble evolve implicitly with time via the deletion operations. The experts are deleted according to their classification performance, a value that is sensitive to a concept change. While a low diversity and a small classification error in the ensemble is an evidence of a stable concept learnt by some experts, a high

diversity level among other experts allow them to react to a potential change in the current stable concept.

### 4.3.9 The stability-plasticity dilemma

According to the stability-plasticity dilemma, the memory size of the classifier is expected to grow with time when the concept is stable and to shrink when the concept changes. In this section, we analyze the memory size of the expert(s) selected by DACC when classifying instances in a stream.

#### 4.3.9.1 Experimental setup

We consider a stream of instances that are received from time step $t_1$ until $t_{end}$ and one concept drift event occurs at time step $t_{drift}$. The transition between the old and new concepts lasts for $\delta_{drift}$ time steps (see Figure 4.8).

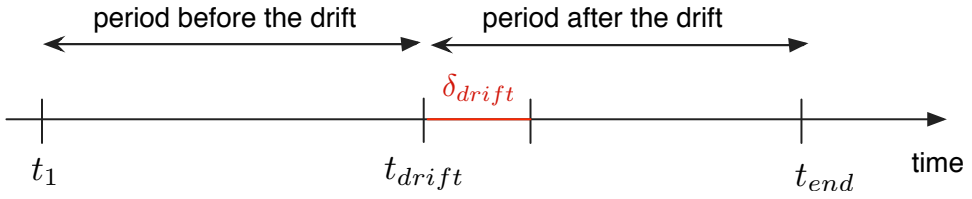


FIGURE 4.8: The drift scenario described in Section 4.3.9

In order to see how the committee deals with the stability-plasticity dilemma, we plot the memory size of the classifier selected for the committee's final prediction. If more than one classifier is selected for the final prediction, we plot the mean memory size of these classifiers. We also show for each memory size:

- The number of training data that belong to the old concept. We refer to these data as coming from the *period before the drift:* $[t_1, t_{drift}]$.
- The number of training data coming from the *period after the drift:* $[t_{drift}+1, t_{end}]$.

When the concept change is continuous ($\delta_{drift} > 1$), the training data from the *period after the drift* contains instances for both the old and the new concept from time step $t_{drift}$ to $t_{drift} + \delta_{drift}$ i.e. until the transition to the new concept is complete. When the concept change happens in one time step, the training data from the *period after the drift* belong to the new concept only.

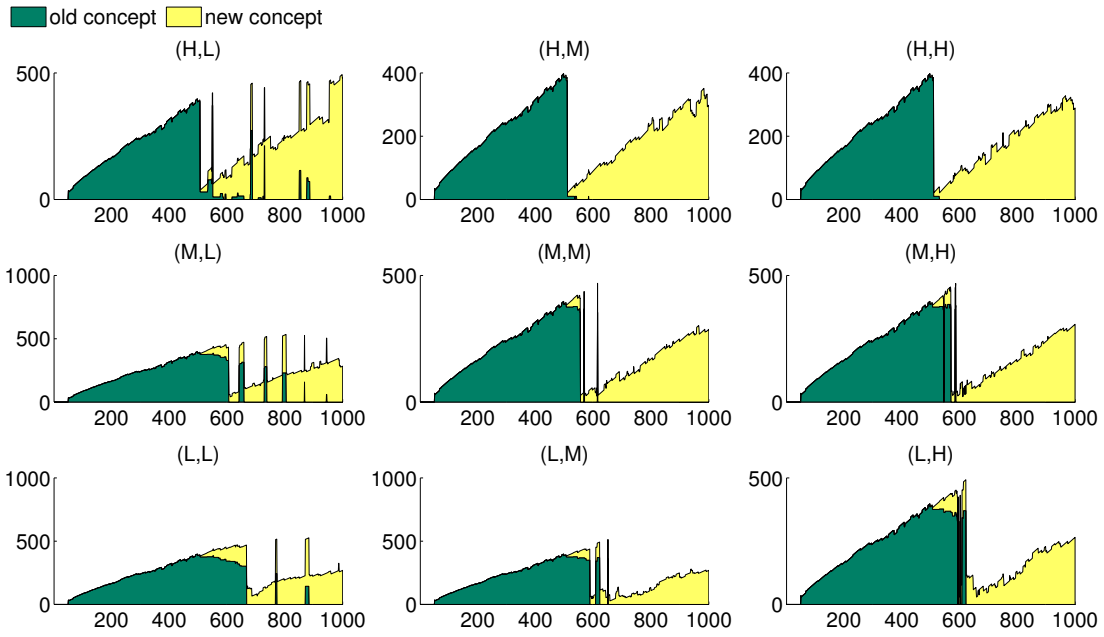


FIGURE 4.9: DACC: The memory size on the *Boolean* problem with a committee of size 30 and a decision tree in the pool of predictors. The pair (*Sp*,*S*) denotes the speed and severity levels of the concept drift, which ranges from *Low*, to *Medium* and *High*.

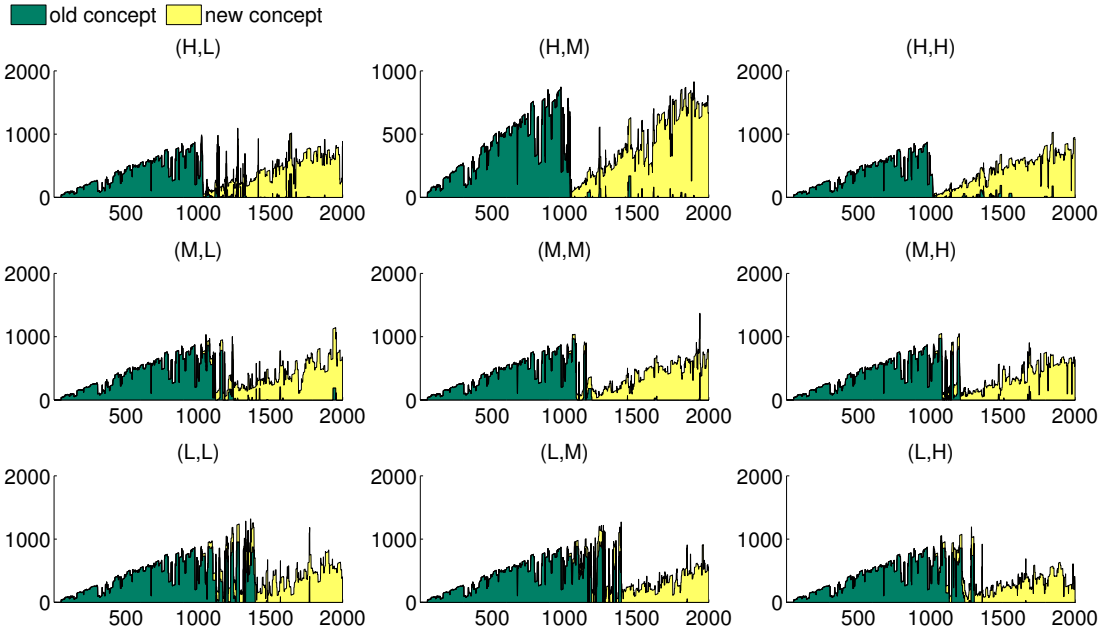


FIGURE 4.10: DACC: The memory size on the *SineH* problem with a committee of size 30 and a support vector machine in the pool of predictors. The pair (*Sp*,*S*) denotes the speed and severity levels of the concept drift, which ranges from *Low*, to *Medium* and *High*.

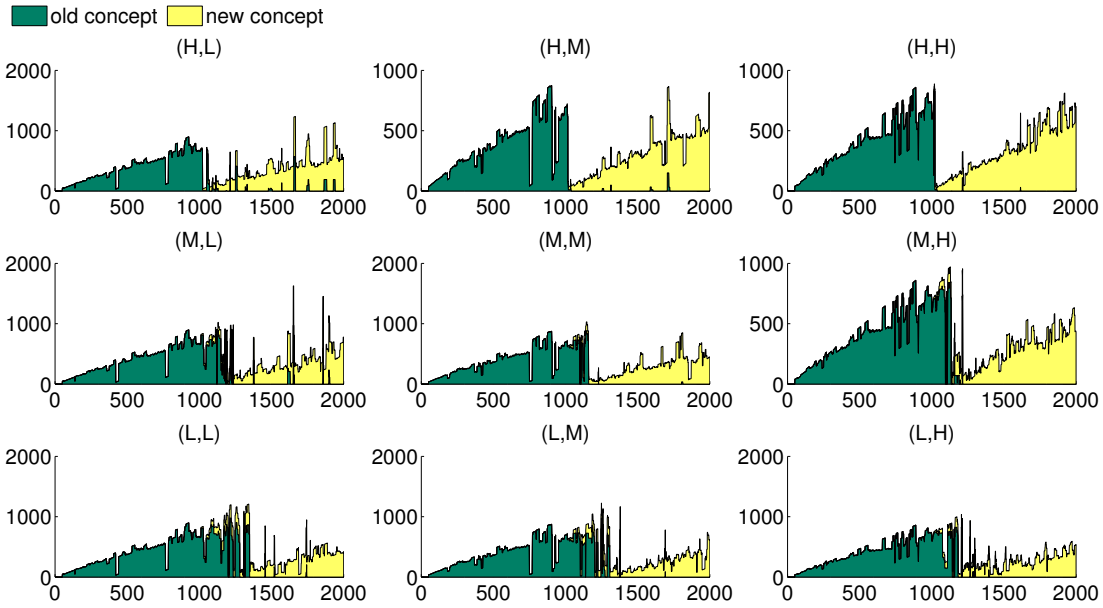


FIGURE 4.11: DACC: The memory size on the *Line* problem with a committee of size 30 and a decision tree in the pool of predictors. The pair ($Sp$,$S$) denotes the speed and severity levels of the concept drift, which ranges from *Low*, to *Medium* and *High*.

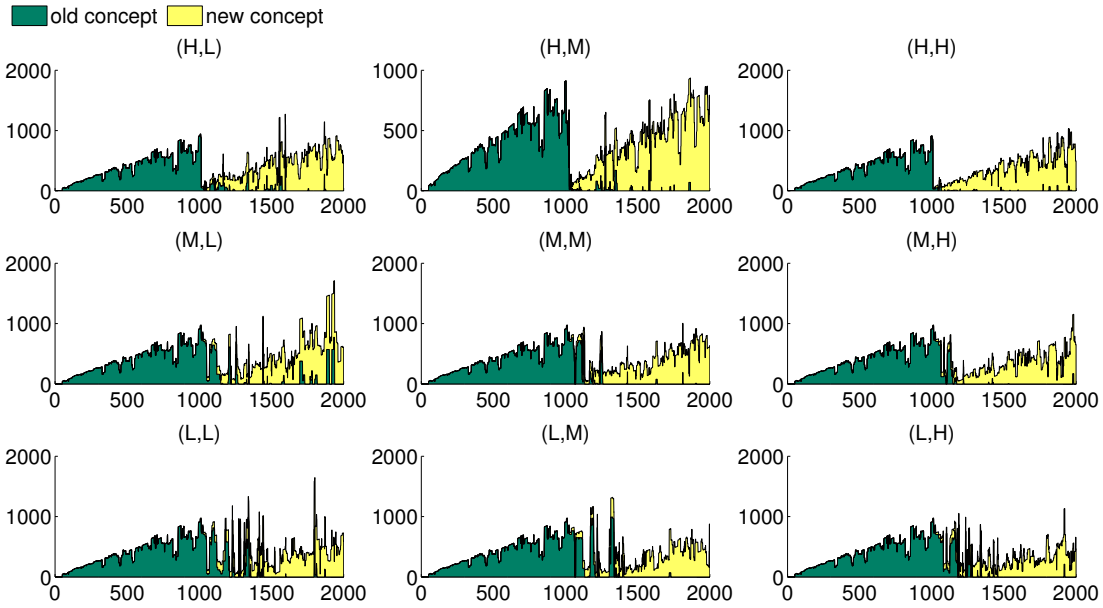


FIGURE 4.12: DACC: The memory size on the *Circle* problem with a committee of size 30 and a decision tree in the pool of predictors. The pair ($Sp$,$S$) denotes the speed and severity levels of the concept drift, which ranges from *Low*, to *Medium* and *High*.

#### 4.3.9.2 Empirical results

We show in Figures 4.9, 4.10, 4.11 and 4.12, the memory size on the artificial problems: *Boolean*, *SineH*, *Line* and *Circle* (see Section 2.3.5.1), respectively. In the former datasets, the drift happens halfway through the stream, with $t_1 = 1$ and $t_{end} = 2 * t_{drift}$. Since the datasets simulate non-noisy environments, we use the MAX combination function which selects the classifier with the highest weight to predict an instance's class. If more than one classifier share the same best weight, they vote for the final prediction.

We notice that, when learning the first concept, all the data come from the period before the drift since the drift hasn't occurred yet. The memory size increases with time because, during this period of stability, the larger the window size the better the prediction. *Increasing the memory size* allows DACC to learn the *stable* concept $C_{old}$.

After the drift, the *memory size is small*, allowing DACC to *adapt to the concept change*. The memory size also increases with time in order to regain stability and learn the new concept $C_{new}$. We can see that, the slower the speed of the concept change, the more the data we see from the old concept just after the drift. This comes from the fact that during the transition to the new concept $C_{new}$, the committee still needs to classifiy instances from $C_{old}$ and thus knowledge from the old concept is required. As a result, the time at which the yellow color takes completely over the green color shifts towards the right of the time axis as the speed of change get slower.

In Figure 4.9, when the severity level of the concept change is low, we notice several peaks in the memory size after the drift, with both green and yellow colors. Hence, even after the drift, the memory may contain training data from period before the drift. This comes from the fact that the less the severity of the change, the more the resemblance between $C_{old}$ and $C_{new}$ and thus, we can take advantage of the acquired knowledge from $C_{old}$ to predict on $C_{new}$. As the severity increases, this knowledge share disappears.

### 4.3.10 Effect of parameters

In this section, we study the impact of DACC's parameters and/or components on its predictive performance. We analyze the effect of:

1. The maturity age
2. The pool of types of experts
3. The committee size
4. The deletion strategy

#### 4.3.10.1 Maturity age

The maturity age $\tau_{mat}$ represents the number of training data a predictor should learn before being able to give its prediction. The prediction of an unmature perdictor is not taken into account in the committee's final prediction. The sole exception is when all the predictors in the committee are not mature. In this case, the unmature predictors share a weight of zero and the final prediction is the result of a *majority vote*. The maturity age plays another role: it controls the *deletion frequency* in the committee. Since all the predictors should be mature before a deletion operation, $\tau_{mat}$ training data (or equivalently $\tau_{mat}$ time steps) separate two consecutive deletion operations. Thus, a high maturity age implies a small deletion frequency and vice versa.

**Experimental setup**

We consider a stream of instances that are received from time step $t_1$ until $t_{end}$ and one concept drift event occurs at time step $t_{drift}$. The transition between the old and new concepts lasts for $\delta_{drift}$ time steps (see Figure 4.13).

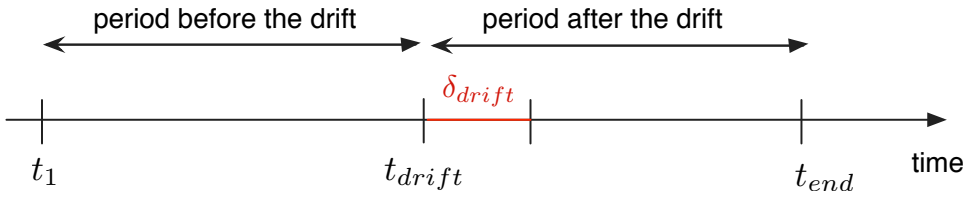


FIGURE 4.13: The drift scenario described in Section 4.3.10.1

In order to see the effect of the maturity age on our ensemble method, we evaluated DACC on the datasets of the *Line* artificial problem (see Section 2.3.5.1), with different maturity ages. In the *Line* datasets, the drift happens halfway through the stream, with $t_1 = 1$ and $t_{end} = 2 * t_{drift}$.

For each dataset and maturity age, we computed the following measures:

- The online performance
- The mean classification error
- The mean age of the committee members

We conducted two sets of experiments with the following configurations:

1. The experts in the ensemble are also of the same type, this time, decision trees, with $N = N_{pool} = 20$, $\tau_{eval} = 20$ and $cmb = \text{MAX}$.

2. The experts in the ensemble are all of the same type, that is, perceptrons, with $N = N_{pool} = 30$, $\tau_{eval} = 20$ and $cmb = \text{MAX}$.

The results of the two sets experiments are shown in Figures 4.14 and 4.15, respectively.

**Empirical results**

The first deletion operation occurs at time step $t = N_{pool} + \tau_{mat}$, after the committee has reached it maximum size and all the members are mature. Before $t$, the mean age increases linearly with time as new training data is observed. Starting at $t$, deletion operations will change the evolution of the mean age value depending on the age of the deleted members. In Figures 4.15 and 4.14, we can see a drop in the mean age every $\tau_{mat}$ steps which corresponds to a deletion operation. We also notice the following:

A higher maturity age implies a smaller deletion frequency and as a result, a slower adaptation to concept change. See, for instance, the concept changes with a high speed level (first row in Figures 4.15 and 4.14).

When the concept change is slow or continuous, a high deletion frequency may affect the classification performance during the beginning of the transition from concept $C_{old}$ to $C_{new}$. See, for instance, the concept changes with a slow speed level (third row in Figure 4.15).

In fact, when the concept starts to change, the classifiers are asked to predict labels for instances of both $C_{old}$ and $C_{new}$. Their classification performance will go up and down which will make them subject to a higher chance of deletion. A smaller deletion rate, however, implies less deletion operations and thus more stability in the committee during the transition.

In Figure 4.15, we see a slight difference in the online performance when learning the old concept depending on the maturity age. This difference comes from the majority vote of unmature predictors. As we previously mentionned, unmature committee members vote for the final prediction and once a predictor becomes mature, the majority vote of the unmature committee becomes a vote between the best mature predictors. Thus, the higher the maturity age, the longer the time the majority vote of the unmature predictors takes over the vote of the best predictors.

In Figure 4.16, we compute for each maturity age the mean classification error, averaged over different committee sizes (10, 20 and 30), different pools of predictors (a single decision tree or a single perceptron) and the 9 datasets of the *Line* problem. The average error is split into the three consecutive periods: *before*, *during* and *after* the drift, as explained in Section 4.2.4.

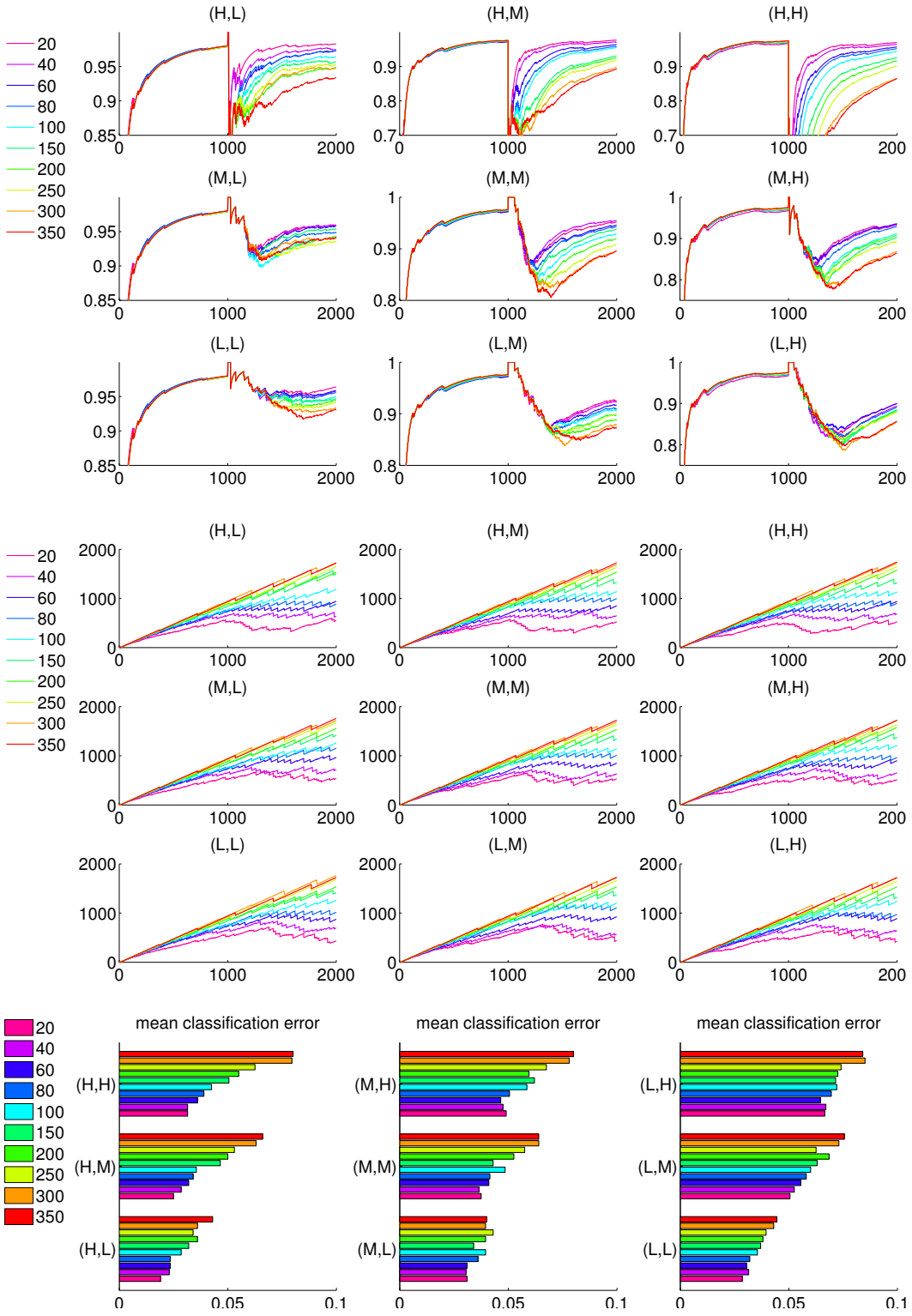


FIGURE 4.14: DACC: The effect of maturity age on the 9 datasets of the *Line* problem with a committee of size 20 using decision trees as base learners. From top to bottom: the online error, the mean age of the committee members and the classification mean error.

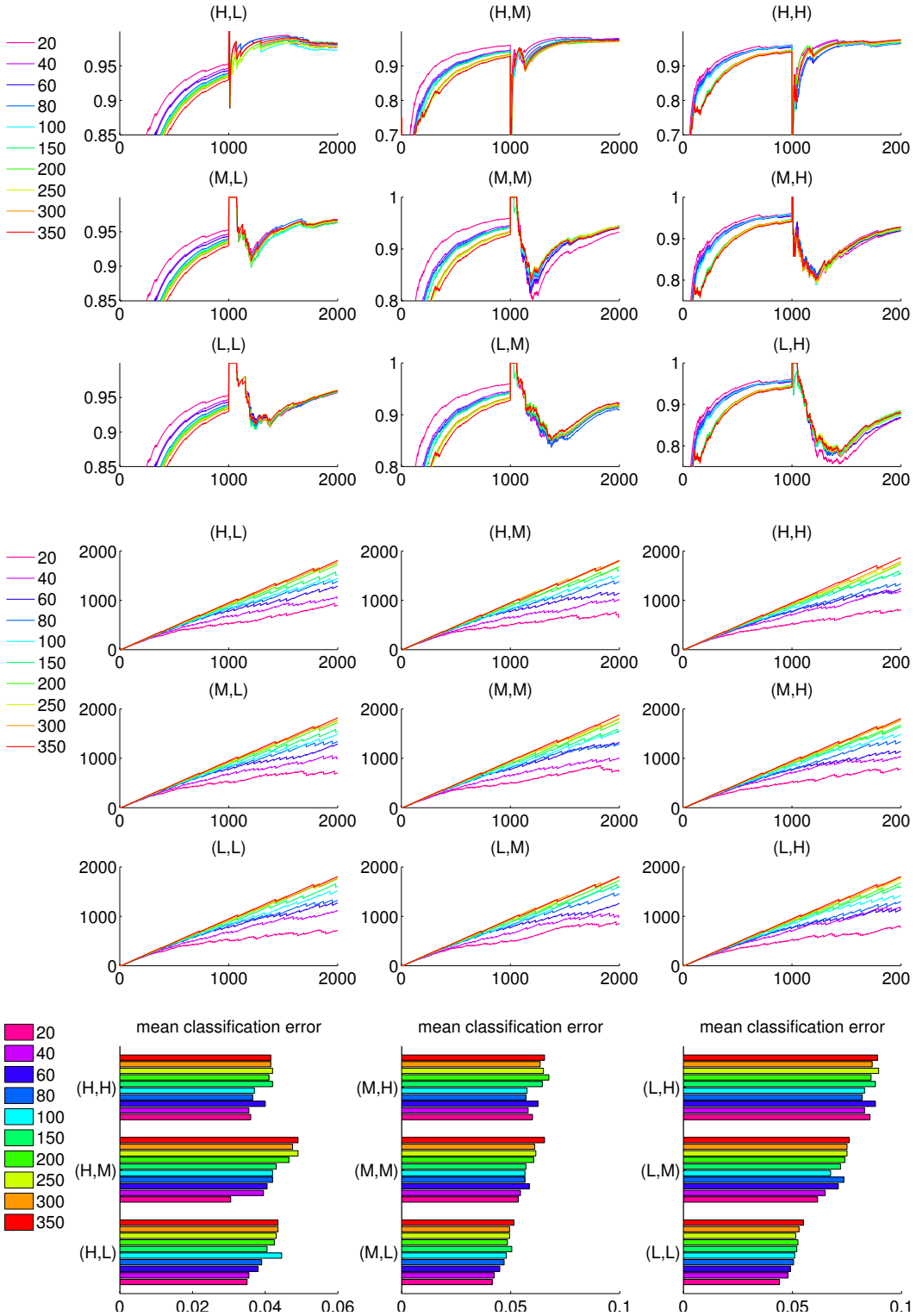


FIGURE 4.15: DACC: The effect of maturity age on the 9 datasets of the *Line* problem with a committee of size 30 using perceptrons as base learners. From top to bottom: the online error, the mean age of the committee members and the classification mean error.

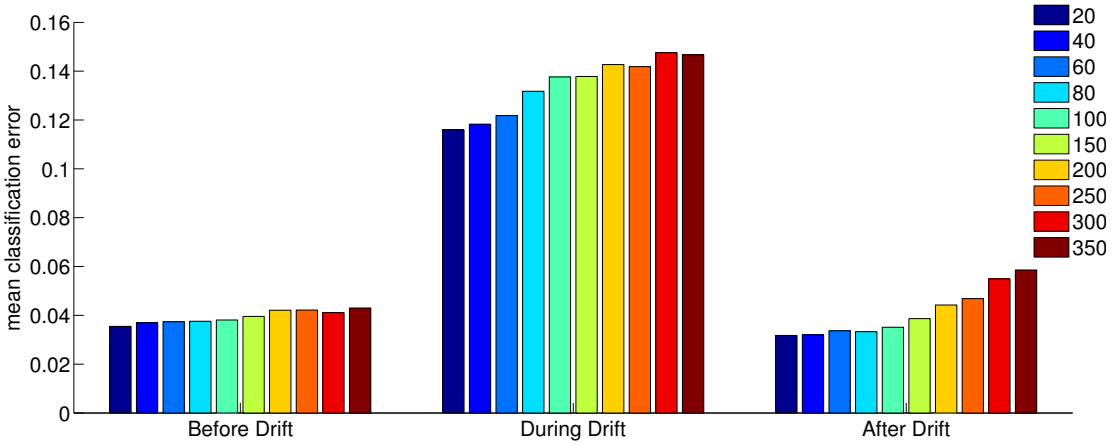


FIGURE 4.16: DACC: The mean classification error depending on the maturity age. The mean is computed over three time periods: before, during and after the drift.

As we can see, the maturity age has its largest impact on the predictive performance during and after the drift. As we pointed out before, a higher maturity age slows down the reactivity to a concept change and as a result, delays the learning of the new concept.

#### 4.3.10.2 Pool of types of experts

With each insertion of new members in the committee, DACC adds a pool of types of predictors, that is, different types of experts. Experts can be different learning models, learning models with different structures and/or training functions. It is also possible to add static prediction rules that don't change with time. For instance, a rule that always predicts the same class value.

**Experimental setup**

We study here the advantage of *mixing* different types of experts in the same ensemble. In order to do so, we conducted experiments on three of the datasets of the *Plane* artificial problem (see Section 2.3.5.1), using the following pools of types of experts:

1. *Dt*: the ensemble contains a single type of experts, that is, decision trees trained on every example received.
2. *Pois-dt*: the ensemble contains a single type of experts, decision trees trained each on $K$ copies of each training examples, where $K$ follows a $Poisson(\gamma = 1)$ distribution.

3. *Pois-dt-p*: the ensemble contains a single type of experts, decision trees trained each on $K$ copies of each training examples, where $K$ follows a $Poisson(\gamma = 1)$ distribution. Unlike the trees in *Pois-dt*, these trees are virtually pruned after each training. Virtual pruning is done according to the minimum description length principle. This is generally useful when the instances are known to be noisy, which is the case in the *Plane* datasets.

4. *All*: the ensemble contains the three different types of decision trees presented above.

After preliminary experiments, we set the remaining settings to the following values: $\tau_{eval} = 10$, $\tau_{mat} = 5$, $cmb$=WVD, and $N_{pool} = 25$. We show in Figure 4.17, the online performance of DACC using the four pools of types of predictors. The online performance is reset when the concept changes, at time step 500, and is averaged over 5 runs.

**Empirical results**

The comparison of the first three pools shows the following:

With the first pool of predictors containing a simple decision tree, the online performance is clearly lower than this of the other pools, after learning the first and second stable concepts at time steps 500 and $1,000$, respectively. However, when the speed and severity of the change are high (figure top-right), this pool of predictors reacts faster to the change, raising the online performance sooner than with other pools of predictors.

The second and third pools of predictors have a close performance on the first two databases (top-right and top-left), with the third pool being clearly better than the second one on the third database (bottom). They both have a higher online predictive accuracy than this of the first one. However, as previously mentionned, we notice a slower reaction to the concept change when the change is high an severe (top-right).

When mixing the three types of decision trees, we notice that the ensemble takes advantage of each type of predictor and gives better prediction results. This suggests that we don't need to have a priori knowledge of the more suitable type of predictors for the adaptative learning task. DACC will do the work by automatically removing the worst predictors in the ensemble. However, a larger pool entails a larger ensemble size. With four experts in the pool, the ensemble size is $N = N_{pool} * 4 = 100$, versus only 25 experts for the other three pools.

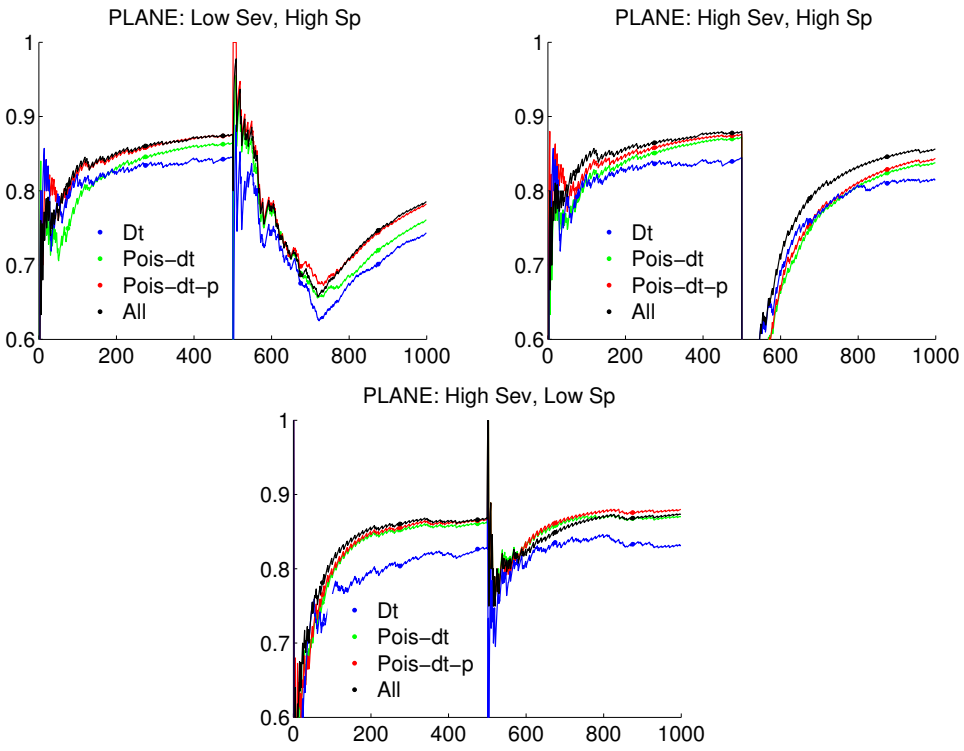


FIGURE 4.17: DACC: The online performance on three of the databases of the *Plane* problem using different pools of predictors. The online performance is reset at time step 500, when the concept changes. The pair (*Sp*,*S*) denotes the speed and severity levels of the concept drift, which ranges from *Low*, to *Medium* and *High*.

#### 4.3.10.3 Committee size

In order to study the impact of the committee size on DACC's predictive performance, we conducted several experiments with different values for $N_{pool}$: 10, 20, 30, 40, 50 and 100, and this, on the datasets of the *Plane*, *Boolean*, *SineV* and *SineH* artificial problems (see Section 2.3.5.1).

We used a single decision tree in the pool of predictors i.e. $N = N_{pool}$. The maturity age $\tau_{mat}$ and the evaluation size $\tau_{eval}$ were set to 20. The combination function selected the predictor(s) with the highest weights i.e. *cmb* =MAX.

We show in Figure 4.18, the mean classification error averaged over the three consecutive periods: *before*, *during* and *after* the drift, as explained in Section 4.2.4. We see no significant difference in the classification performance depending on the committee size: what we can get from a large committee of size 100, we can also get from a small committee of size 10.

The results seem somehow unrealistic as there should be a size value $N_{min}$ that is small enough to make the ensemble fail in its adaptation to concept changes. To investigate the matter, we reconducted the same experiments after reducing the size of the committee

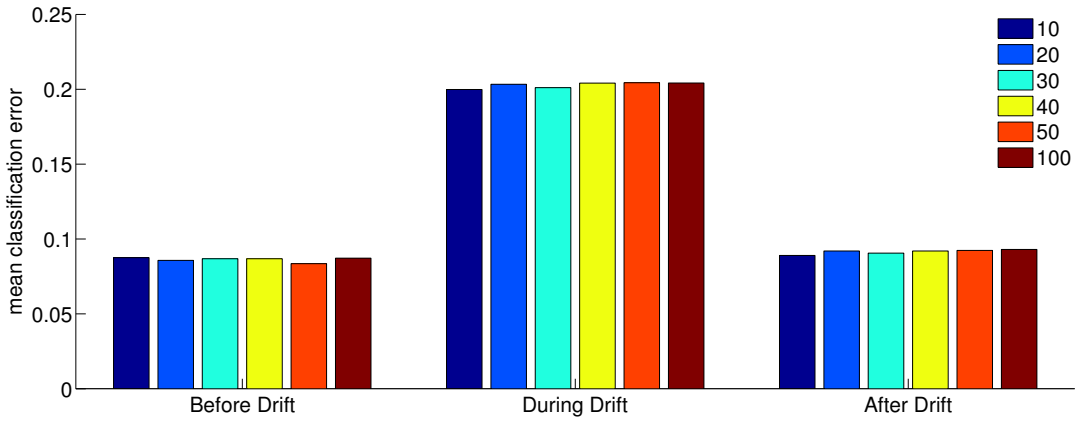


FIGURE 4.18: DACC: The mean classification error depending on the committee size. The mean is computed over three time periods: before, during and after the drift.

to $N = 5$ and $N = 2$. We show in Figure 4.19 the online predictive performance on the datasets of the *Boolean* and *Circle* problems in case of a sudden and severe drift, that is, in the presence of concept changes with high levels of speed and severity. All the experiments start with the same random seed so that, for each problem, we have the same experts at the beginning of the experiments.

We notice that with *Boolean*, even with only two learners, the ensemble is able to recover from the concept change, regaining its predictive performance at the end of the experiment. In fact, the first learner $h_1$ was kept in the committee when learning the first concept while the second learner was constantly removed. When the concept changed (time step $t = 1,000$), $h_1$, who was outdated, got removed from the ensemble and was replaced by a new learner $\tilde{h}_1$ with no memory of the past. At the time of the next deletion ($t = 1,020$), the predictive performance of $\tilde{h}_1$, trained on instances from the new concept, was sufficiently higher than the one of the second learner who at the time, was trained on instances belonging to both the old and new concepts. As a result, $\tilde{h}_1$ was kept in the committee while the second learner was constantly removed. While the first learner was used for stability (the learning of the first and second stable concepts), the second learner was used for plasticity as it allowed the removal of its outdated fellow and took over it in the classification process during the brief transition between the consecutive concepts.

With *Circle*, the situation is different: the ensemble is not able to adapt to the concept change with only two learners. In fact, after the change, the deletion strategy couldn't make a clear distinction between the two learners as both of them had a bad predictive accuracy regarding the new concept. Hence, both learners were deleted repeatedly, and the ensemble failed to follow the new trends. This suggests that the frequency of deletion was too high (i.e. the maturity age was too small). Hence, the youngest learner, expected

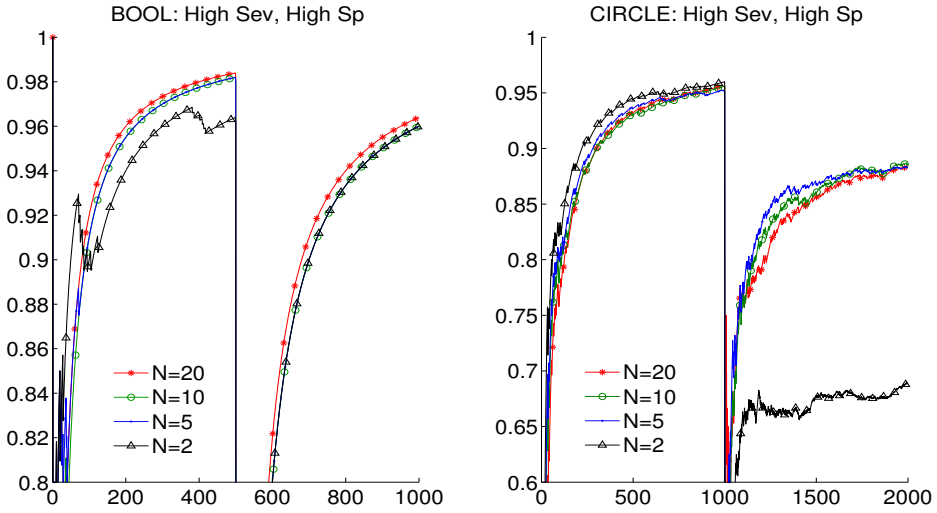


FIGURE 4.19: DACC: The online performance on databases from the *Boolean* (left) and *Circle* (right) problems using different committee sizes. The online performance is reset when the concept changes. The pair (*Sp*,*S*) denotes the speed and severity levels of the concept drift. The maturity age and the evaluation size are set to $\tau_{mat} = \tau_{eval} = 20$ and the base learners are decision trees.

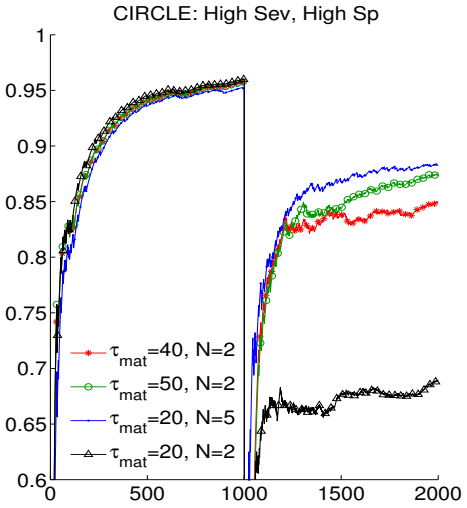


FIGURE 4.20: DACC: The online performance on a database from the *Circle* problem using different combinations of the maturity age and the committee size. The online performance is reset when the concept changes. The pair (*Sp*,*S*) denotes the speed and severity levels of the concept drift. The evaluation size is set to $\tau_{eval} = 20$ and the base learners are decision trees.

to adapt to the changing conditions, was not able to learn enough training data before being evaluated for deletion. It was not given enough time to improve its predictive performance and was constantly deleted.

We claimed earlier in this chapter that DACC's deletion strategy overcomes an unadapted maturity value by increasing the deletion size, that is, the number of learners from

which a "bad" learner is deleted. The deletion size is proportional to the ensemble size ($ds = N/2$). In this case, it is equal to 1. By increasing the ensemble size to $N = 5$ learners, the expectation of young experts to survive deletion also increases (see Section 4.3.2), and the ensemble regains its adaptation capability as shown in Figure 4.19.

If the adaptation problem of the ensemble of two learners lies in the small maturity age as we supposed, then increasing the former value should solve the issue. We show in Figure 4.20 the effect of higher maturity ages. We can clearly see that by increasing the maturity age, the ensemble is able to adapt to the change even with a small committee size. In fact, with $\tau_{mat} = 50$ and $N = 2$, we get nearly the same results as with $\tau_{mat} = 20$ and $N = 5$ at the end of the experiment.

To sum up, we conclude that the maturity age $\tau_{mat}$ and the committee size $N$ are interdependent. When choosing an adapted value for $\tau_{mat}$, the predictive performance of DACC does not rely on the value $N$. This was shown in the left plot of Figure 4.19 and also in Figure 4.18 where the increase of $N$ from 10 to 100 didn't have a significant impact on the classification accuracy. In addition, a value of $N$ that is large enough can compensate a small value for $\tau_{mat}$. While a small maturity age leads to a high plasticity, a large committee size overcomes the problem of deleting only young experts when the environment evolves.

#### 4.3.10.4 Deletion strategy

In this set of experiments, we aim at evaluating DACC's deletion strategy, the main component allowing to adapt to potential concept changes. In order to do so, we evaluate DACC's performance against this of another ensemble which operates exactly like DACC, except for its deletion strategy. While DACC deletes experts in a regular manner by selecting experts randomly from the worst half of experts, the other approach uses the *replace the loser* strategy and deletes the loser according to a threshold value on its recent classification performance $\theta_d$.

Thus, an expert is removed from the ensemble if:

- the expert is mature,
- the expert recent classification accuracy is below $\theta_d$, and
- the expert is the worst expert in the ensemble.

DACC with the modified deletion strategy will be referred to as TDS (for *Threshold Deletion Strategy*).

The evaluation of the deletion strategy is concerned with the following aspects:

- the *reactivity* to concept changes and the ability to recover from a drift.
- the ability of *knowledge transfer* between two consecutive concepts i.e. the ability to classify instances belonging to the old concept after a concept change.
- the advantage of using the deletion strategy over the use of an online learning system with no adaptation mechanism

Knowledge transfer is useful in two cases. First, during the drifting time in case of a non-instantaneous drift since instances are generated for both the old and the new concept. Secondly, in case of a non severe concept change where knowledge acquired from the learning of the old concept can help classify instances from the new concept.

In order to evaluate the ability of knowledge transfer, we evaluate TDS and DACC against a learning system consisting of a single classifier that learns incrementally with each new received training example. When the concept changes, we reset the classifier's memory, allowing it to learn from scratch, with no knowledge of the past. This classifier, which we will refer to as the *perfect forgetter*, is used to assess the ability of knowledge transfer just after the drift. If a learning approach is able to transfer knowledge between consecutive concepts, it is expected to have a performance higher than this of the *perfect forgetter* when a new concept appears.

We also compare DACC and TDS with a learning system consisting of a single classifier that does not adapt to concept changes. In other words, the classifier learns the current concept without erasing its memory i.e. it keeps learning as new training data becomes available. The former classifier, referred to as *the NoDriftHandler* allows to see the advantage of the adaptation strategy.

**Experiment (1)** This set of experiments compares the following learning systems:

- DACC: our ensemble of classifiers described in Section 4.3
- TDS: the modified version of DACC which uses the *replace the loser* strategy with a deletion threshold to remove experts from the ensemble. TDS is evaluated with three threshold values: 0.6, 0.7, 0.8.
- The *perfect forgetter*: a single classifier that learns the current concept. We erase its memory when the concept starts changing.
- The *noDriftHandler*: a single classifier that does not adapt to concept changes.

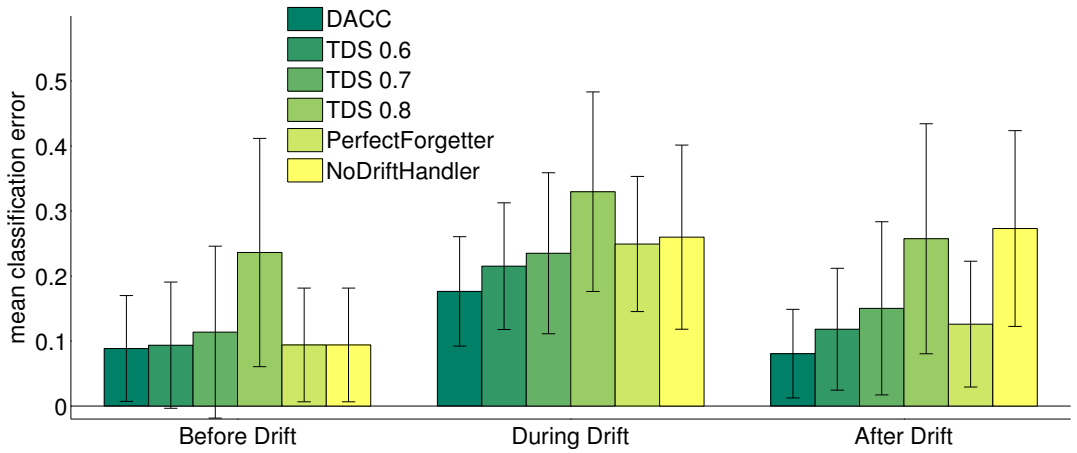


FIGURE 4.21: The mean classification error of DACC, TDS, the *perfect forgetter* and the *noDriftHandler*, before, during and after the drift, as calculated on the datasets of the artificial problems: *SineH*, *Circle*, *Line*, *SineV*, *Boolean* and *Plane*.

We evaluated the different approaches on the 54 databases of the six artificial problems suggested by Minku: *SineH*, *Circle*, *Line*, *Boolean*, *SineV* and *Plane* (see Section 2.3.5.1). Each dataset is a stream of instances with one concept drift event that occurs at time step $t_{drift}$, halfway through the stream. The drift lasts for $\delta_{drift}$ time steps.

The experiments were conducted using the following settings. The pool of types of experts consists of a single decision tree. The ensemble size is set to $N = N_{pool} = 20$ and the combination function is MAX. The maturity age is set to $\tau_{mat} = 20$, and the weight of an expert reflects its classification accuracy on the last $\tau_{eval} = 20$ time steps. The experiments on the databases of the *Line*, *SineH* and *Circle* problems were reconducted, replacing the decision tree in the pool with other types of experts:

- a perceptron for the *Line* problem.
- a 2 layers non linear neural networks with 3 hidden neurons for the *Circle* problem.
- a support vector machine with a radial basis function for the *SineH* problem.

Therefore, the total number of experiments was equal to: 6 problems* 9 databases (using decision trees) + 3 problems * 9 databases (using different experts) = 81 experiments.

We show in Figure 4.21 the classification error of DACC, TDS, the *noDriftHandler* and the *perfect forgetter*, averaged on the 81 experiments detailed above. The average error is split into the three consecutive periods: *before*, *during* and *after* the drift, as explained in Section 4.2.4. The *perfect forgetter* is reset at time step $t_{drift}$. We notice the following:

#### Before the drift

The *noDriftHandler* and the *perfect forgetter* both have the same classification error before the drift since they act the same way. The only difference between these approaches comes after the drift since the *perfect forgetter* resets its classifier, while the *noDriftHandler* keeps learning without any explicit change in its classifier. TDS 0.8 has the largest classification error before the drift due to its relatively high performance threshold which causes many deletions in the committee even though the concept is stable. DACC, on the other side, is able to learn the stable concept before the drift even though classifiers are deleted all the time. Thus, the deletion strategy doesn't have a negative impact on the ensemble's predictive performance in the stationary case.

#### During the drift

As we can notice, the larger the threshold value, the larger the classification error of TDS during the drift. In fact, when the performance threshold is high, there are more chances for TDS to remove experts from the ensemble when the concept changes, preventing knowledge transfer via old committee members. DACC on the other hand keeps its experts for a longer time due to the frequency of deletion, keeping old experts even when their performance drops significantly. The classification error of DACC, being lower than this of the *perfect forgetter*, shows that DACC is able to achieve knowledge transfer during the drift.

#### After the drift

TDS 0.8 has a large classification error after the drift because, as pointed out before, its threshold value causes many consecutive deletion operations which prevents the adaptation to the concept change.

The *noDriftHandler*, as expected, has the worst classification accuracy. As mentionned previously, the *noDriftHandler* is a single classifier that learns continuously as new training data is received. The memory of previously training data is not reset as in the other approaches. If the classifier's learning algorithm is lossless, which is the case with the decision trees, the memory of old training examples is kept even after the drift which makes the *noDriftHandler* unable to adapt to the new concept. Lossless online algorithms are available for decision trees, Naive Bayes models and others [92, 100]. Nevertheless, many online learning models like the online backpropagation in neural networks [43] have the property of being lossy i.e. they tend to forget the old acquired knowledge from the old concept when enough training data from the new concept have been learnt.

The forgetting strategy can thus be implicitly embedded in the classifier learning algorithm which makes it able to forget the old concept and adapt to the new one. However, the implicit forgetting strategy is usually slower than the explicit one, designed for the learning system to adapt to evolving concepts.

The *perfect forgetter* is expected to adapt the best to concept changes since it simulates an approach with a perfect drift detection system. Even though it seems to be the best case scenario, we notice that DACC outperforms the *perfect forgetter* not only *during* the drift but also *after* the drift. In fact, in case of a continuous drift (i.e. $\delta_{drift} > 1$), the reinitialized classifier at time step $t_{drift}$ will classify and learn data from both concepts until time step $t_{drift} + \delta_{drift}$. This has two consequences. First, learning training data from both concepts will hurt the learning of the new concept. Secondly, resetting the classifier once the drift is detected makes it difficult for the *perfect forgetter* to classify instances from the old concept during the concept transition since the old acquired knowledge is erased.

After the drift is completed at time step $t_{drift} + \delta_{drift}$, the performance of the *perfect forgetter* on the new concept will depend on:

- the drifting time: for how long training data for both concepts were learnt?
- the severity of the drift: how many training examples learnt from the old concept during the transition period $\delta_{drift}$ are wrongly classified according to the new concept? These examples can be seen as noisy instances with incorrect target values, according to the new concept.
- the classification model used: is it lossy or lossless? Will the classifier forget the outdated training examples from the old concept after observing enough examples from the new one?

It is important to note that we provide the drift starting time $\delta_{drift}$ to the *perfect forgetter*. A practical approach simulating the *perfect forgetter* should be embedded with an explicit concept drift detection mechanism that should be sensitive enough to detect concept drifts, and robust enough to avoid false alarms. Add to that the fact that our *perfect forgetter* resets its classifier at the beginning of the drift. In a practical approach, and when the drift is continuous, we should also decide *when* to reset the classifier(s): at the beginning of the drift, at the middle of the transition, or at the end? The answer to these questions requires a priori knowledge of the drift properties such as the drift speed and severity, among others.

### Summary

The experimental results give an insight of DACC's performance in the presence of drifting concepts. The results suggest the following:

- DACC can achieve knowledge transfer
- DACC's deletion strategy does not affect the predictive performance when the concept is stable
- DACC's deletion strategy makes possible the adaptation to various types of concept drifts with different levels of speed and severity, on different problems, and this using the same preset parameters values.

**Experiment (2)** For a more detailed analysis of DACC's deletion strategy, we show in Figures 4.22, 4.23 and 4.24 the online performance of DACC and TDS on the *Boolean*, *Circle* and *SineH* databases, respectively, with the same settings as in Experiment 1. In the following, we analyze both TDS and DACC's deletion strategy.

### TDS deletion strategy

TDS uses the "replace the loser" strategy to delete experts from the ensemble, and this, only if the performance record of at least one of the mature experts falls below a predefined threshold $\theta_d$. Some of the problems related to such deletion strategy (discussed in Section 4.2.4) can be observed:

***Selecting a small threshold value:*** In the *Boolean* problem (see Figure 4.22), when the change is the least severe (see databases $(M, L)$ and $(L, L)$), TDS with threshold values 0.8, 0.7 and 0.6 doesn't adapt to the concept drift due to relatively small threshold values. The classification accuracy of the classifiers doesn't go below any of the preset thresholds and thus the committee doesn't remove any of its members after the drift. Since the classifiers are lossless decision trees, the knowledge acquired from the old concept is kept after the drift which makes it harder to predict correctly on the new concept.

***Selecting a high threshold value:*** In the *Circle* problem (see Figure 4.23), TDS 0.8 is not able to learn any of the concepts. In fact, the threshold value is relatively high and the committee removes a predictor at each time step. While TDS's threshold value 0.8 gives the best predictive performance for the database $(H, L)$ of the *Boolean* problem (see Figure 4.22), it does not allow the ensemble to adapt to any of the concept changes of the *Circle* problem.

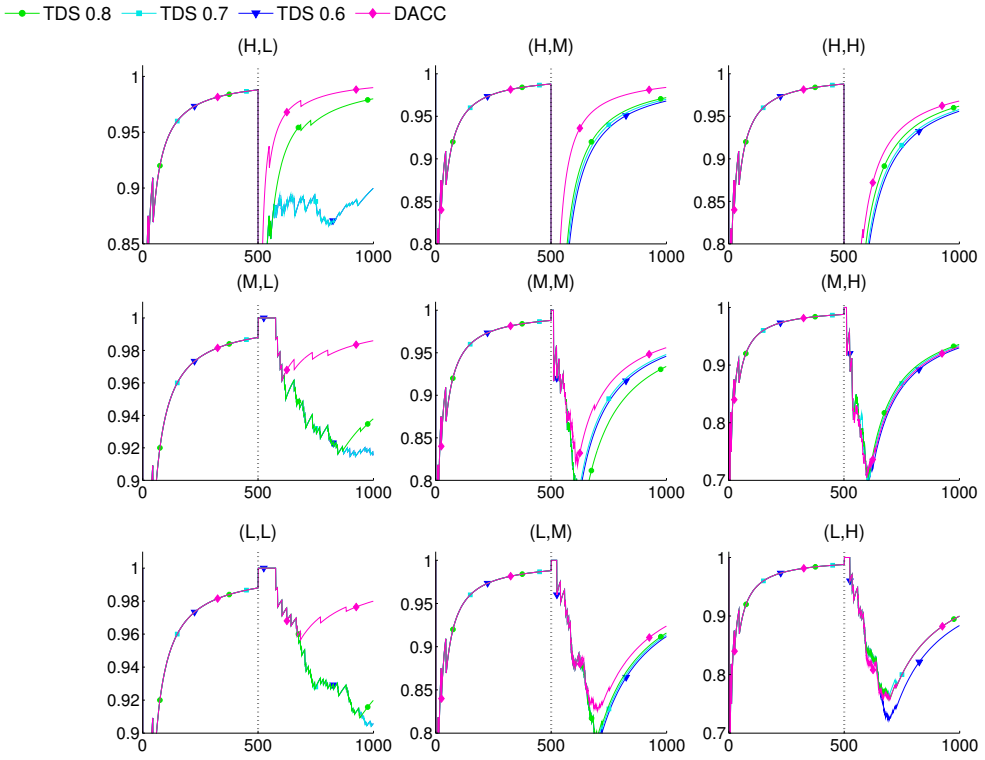


FIGURE 4.22: DACC vs TDS: The online performance on the databases of the *Boolean* problem. The experts are lossless decision trees and $N = 30$. The $x$-axis represents the training examples (time step) and the $y$-axis represents the online classification performance (the percentage of correctly classified instances so far). The online performance is reset at time step 500 when the concept starts changing.

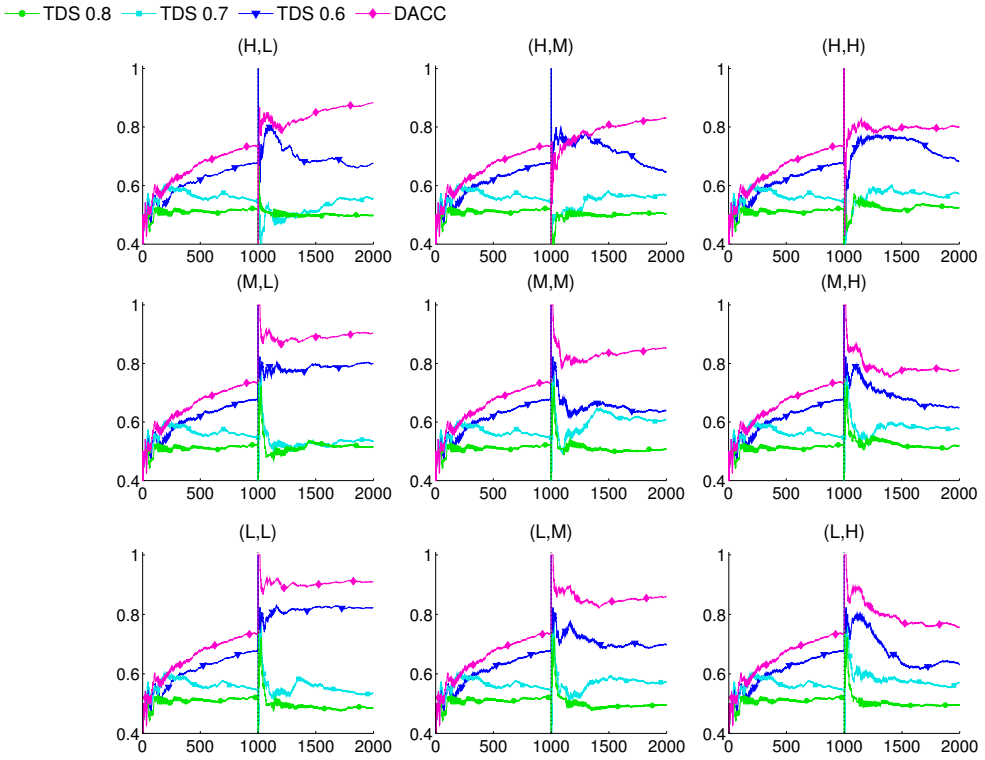


FIGURE 4.23: DACC vs TDS: The online performance on the databases of the *Circle* problem. The experts are lossy feedforward neural networks and $N = 20$. The $x$-axis represents the training examples (time step) and the $y$-axis represents the online classification performance (the percentage of correctly classified instances so far). The online performance is reset at time step $1,000$ when the concept starts changing.

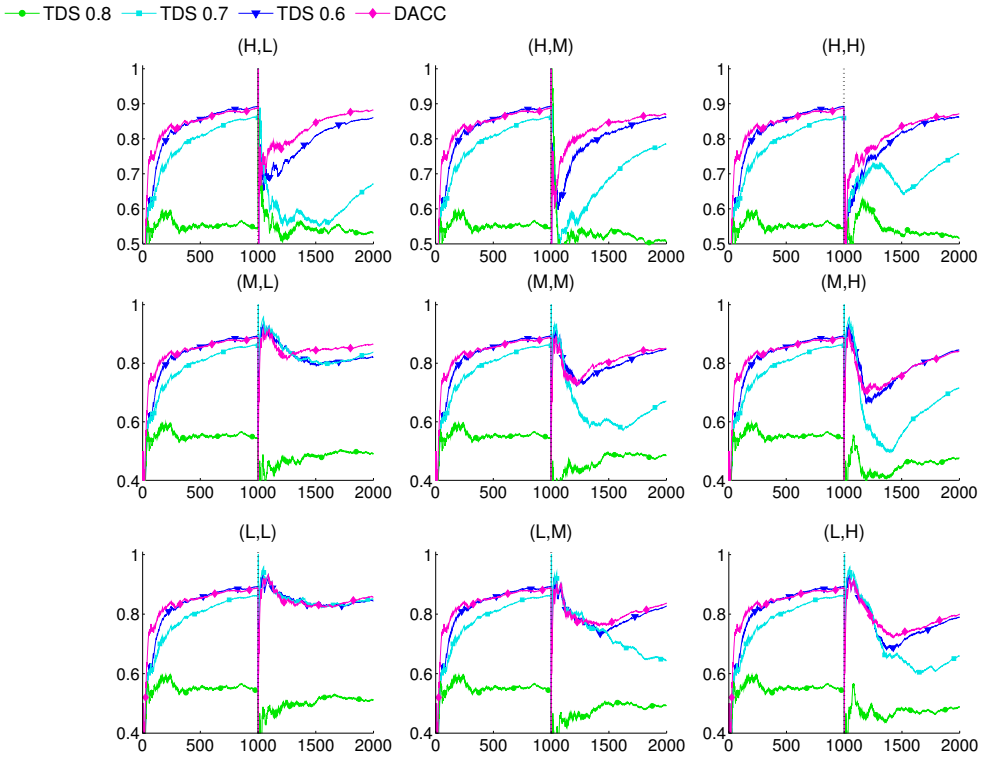


FIGURE 4.24: DACC vs TDS: The online performance on the databases of the *SineH* problem. The experts are lossless decision trees and $N = 20$. The $x$-axis represents the training examples (time step) and the $y$-axis represents the online classification performance (the percentage of correctly classified instances so far). The online performance is reset at time step $1,000$ when the concept starts changing.

***Repeated cycles of deletion:*** In the *SineH* problem (see Figure 4.24), TDS 0.7 adapts slowly to the concept change $(H, L)$ with *high* speed and *low* severity levels. In fact, the committee is reset multiple times from the beginning of the drift until time step 1600, the time when the online performance starts going up. Another example of repeated deletions is observed in the *Circle* problem (see Figure 4.23) where TDS 0.7 does not adapt to the concept change $(H, L)$ either. In the former case, the committee removes continuously the same subset of classifiers from the beginning and until the end of experiment. By removing the same classifiers, the outdated classifiers trained on the old concept are kept after the drift and the newly added classifiers are constantly removed without getting the chance to learn the new concept.

***Knowledge transfer:*** Generally, the less the severity of the change, the more the resemblance between the consecutive concepts and thus, the more the advantage of the knowledge transfer. In the *Boolean* problem, the severity levels *low*, *medium* and *high* have class severity of 11%, 44% and 67%, respectively. Hence, even with the *high* severity level, 33% of the input space keeps the same class labels after the drift. In Figure 4.22, when using TDS 0.7 on the *Boolean* problem and when the speed level is *high* (see the databases $(H, sev)$ in the first row), we don't see the effect of knowledge transfer after the

drift, in severity levels *medium* and *high*. We would expect a higher online performance after the drift as the severity of the change decreases. However, TDS 0.7 removes its $N$ classifiers in $N$ time steps just after the drift. As a result, all the experts are removed and no knowledge is transferred.

#### DACC's deletion strategy

According to DACC's deletion strategy, the committee members are not removed at once when a drift happens. DACC keeps its members for a longer time due to its frequency of deletion which forces the existence of old members for at least $N_{pool} * \tau_{mat}$ time steps after the beginning of the drift.This allows the committee to transfer knowledge from the old concept to the new one and if the concept change is continuous, DACC can still predict on the old concept instances during the drift. In Figure 4.22, we can see the effect of knowledge transfer on the *Boolean* problem when the concept changes suddenly (see the databases $(H, sev)$ in the first row). After the drift, old classifiers trained on the old concept transfer their knowledge by predicting on the new concept. Knowledge transfer lasts until a recently added classifier learns the new concept and then takes the lead in predicting. We can see how the effect of knowledge transfer diminishes as the severity of the drift increases. Another advantage of DACC is that it doesn't use threshold values to remove experts from the committee. This frees the algorithm from additionnal parameters, and makes it less sensitive to preset values. Unlike TDS, DACC is able to adapt to a variety of concept changes starting with the same parameter values. Finally, DACC removes a committee member every $\tau_{mat}$ steps in anticipation of change. A relatively small maturity age allows DACC to delete members frequently from the ensemble and thus *react* rapidly to a concept drift. Since DACC always removes a classifier, it is sometimes faster in reacting to concept drifts than other approaches that wait until the predictive error exceeds a certain threshold or that a weight goes below a threshold.

#### Summary

Evaluating DACC's deletion strategy with this of TDS shows the sensitivity of the threshold deletion strategy to its preset parameters. While a threshold value may be adapted to a particular problem, it may not be on other problems. Thus, TDS requires a priori knowledge of the drift properties such as the severity and speed of change. DACC by contrast is less sensitive to the preset parameters. In fact, with the same parameter values, DACC was able to adapt to the concept changes of all the datasets of the six artificial problems suggested by Minku, with a classification error that is smaller in average than this of a *perfect forgetter* that learns new concepts from scratch.

TABLE 4.1: The parameter of DACC

| Parameter | Description |
|---|---|
| *Pool* | The pool of type of experts used in the ensemble. |
| *cmb* | The combination function used which is either MAX or WVD. |
| $\tau_{eval}$ | The size of the evaluation window used to compute the weights of the experts |
| $\tau_{mat}$ | The maturity age of an expert. |
| $N_{pool}$ | The maximum number of *pools of type of experts* allowed in the ensemble |

### 4.3.11 Choice of parameters

We show in Table 4.1 the parameters that should be predefined for DACC's ensemble method. Based on intuition and empirical studies on DACC, we give suggestions regarding the choice of the parameter values. It is important to note that these suggestions are aimed at helping the user in his choices but are not optimal. In the comparisons with state of the art methods in Section 4.4, we will see that when DACC's parameter values are finely tuned, the parameters can be set differently than our recommendations below.

#### The experts

The ensemble can consist of different types of experts, that is, different learning models or learning models with different structures. For instance, neural networks with different layers or number of neurons. The choice of the type of learners is not specific to DACC as all learning algorithms require to predefine the type of learning model used. The advantage of DACC is that different types of learners can be present all together at the same time in the same ensemble. By evaluating all the learners before predicting the label value $y_t$, DACC implicitly evaluates the most adapted type(s) of learners at time step $t$. Hence, even if some predictors in the pool were badly selected (ex: unadapted model or structure to the current learning problem), the algorithm will manage to remove the worst predictors and keep the better ones. And even when "bad" predictors still remain in the ensemble, they won't be selected for the ensemble's final prediction according to the combination functions MAX and WVD.

#### The combination function

When combining the predictions of the learners in the ensemble, we consider two functions: MAX and WVD. While WVD can lead to a slightly slower reactivity to a concept change compared to MAX, it is more robust in case of noise. The choice of the combination function is also related to the relationship we expect from the learners in the ensemble. When the learners are all trained incrementally on every instance they observe in their lifetime, we might consider that they are in *competition* with each other, relative

to their memory of training data. Hence, for each label $y_t$, if we want to choose the prediction of the learner with the most adapted memory size, MAX would be a natural choice. If the learners in the ensemble are not trained on all the training data they observe, as with online bagging and boosting methods for instance [77], members of the ensemble can be seen as weak learners and their predictions are generally combined using a weighted vote to get a good predictive accuracy. Hence, WVD would be rather used in a *cooperative* scenario.

#### The evaluation window

The parameter $\tau_{eval}$ represents the size of the window used to evaluate the predictive accurary of the members of the ensemble. While too small windows would lead to unstable weights, too large windows would incorporate outdated data points when the concept evolves, resulting in misleading weight values. In the majority of our experiments conducted on the artificial problems of Minku, an evaluation window of size 20 offered a good compromise.

With a priori knowledge of the dynamics of the environment, our choice of $\tau_{eval}$ could be tuned. If the concept changes are expected to be slow, a large value for $\tau_{eval}$ will be preferred over a small one since it will provide more data points for the evaluation process and a small number of outdated points during a potential concept change. In case of severe and rapid concept changes, large window sizes should be avoided since they will eventually include outdated testing points when a change occurs.

#### The maturity age

The maturity age $\tau_{mat}$ represents the number of training data a predictor should learn before being able to give its prediction. The maturity age plays another role: it controls the *deletion frequency* in the ensemble. Since all the predictors should be mature before a deletion operation, $\tau_{mat}$ training data separate two consecutive deletion operations. By setting $\tau_{mat} >= \tau_{eval}$, we ensure that all predictors are evaluated on the same window of data at the time of deletion. For a high plasticity, the maturity age should be small enough to allow the introduction of new members in the ensemble at a high deletion frequency (in anticipation of a concept change). Hence, we can decrease the maturity age such that $\tau_{mat} = \tau_{eval}$.

#### The size of the ensemble

By defining the pool of type of experts ($Pool$) and by setting the value $N_{pool}$, we get the maximum capacity of the ensemble $N$ by the equation: $N = N_{pool} \times size(Pool)$. The size of the ensemble $N$ plays two roles. First, it impacts the diversity of experts in terms of different memory sizes: the larger the ensemble, the more the memory sizes to choose upon when predicting the label $y_t$ of an instance $\mathbf{x}_t$. The diversity of window sizes allows the ensemble to deal implicitly with the *stability-plasticity dilemma*. While old learners trained on large window sizes are reliable in periods of stability, young learners are more ready to adapt to concept changes.

The size of the ensemble plays another key role: it controls the deletion size $ds$, that is, the number of experts from which one or more experts will be selected for removal ($ds = \frac{N}{2}$). As we previously explained in Section 4.2.4.3, enlarging the deletion size overcomes the problem of deleting young and promising experts when the deletion frequency is high, equivalently, when $\tau_{mat}$ is small. Hence, when the predefined value of $\tau_{mat}$ is unadapted to the learning task (small enough for a high plasticity but not large enough to protect promising experts from deletion), a large ensemble size $N$ will increase the expectation of a young expert with still a relatively bad performance record to survive in the ensemble according to Equation 4.3, allowing it to further improve its predictive performance before the next evaluation. Note that a large ensemble implies more computations and hence comes at the price of a larger execution time.

## 4.4 DACC: Comparison with Other Systems

We evaluated DACC against other online learning systems that operate in evolving environments. In Section 4.4.1, DACC is compared with CDC, an ensemble method that deletes experts according to the a threshold on the recent classification performance. DACC is also compared with two pioneer drift handling systems: IB3 and the FLORA. In Section 4.4.2, DACC is compared with other systems, more specifically: (a) DWM, a well-cited approach that adapts implicitly to concept changes, (b) a modified online bagging method that detects concept drifts explicitly using EDDM drift detector, and (c) DDD, a weighted combination of four ensembles of experts. Finally, in Section 4.4.3, DACC is compared with a number of methods provided with the open source MOA software [11] which includes a collection of online learning methods, able to learn under stationary and/or evolving environments. The former evaluation process is concerned with both the predictive accuracy of the methods and their processing time.

### 4.4.1 DACC vs CDC

In this set of experiments, we compare DACC with CDC, the ensemble learning algorithm designed by Stanley to cope with evolving concepts. The comparison with CDC was driven by the fact that CDC's performance dominates two pioneer drift handling systems: IB3 and FLORA, as it was shown by Stanley in [85].

CDC differs from DACC in three major points. First, its deletion strategy removes the worst expert in the ensemble according to a threshold value on its recent classification performance $\theta_d$. Secondly, the ensemble's predictions are combined using a weighted vote. The weight of an expert reflects its classification performance on the last $\tau_{eval}$ time steps. Finally, the experts in CDC are decision trees. While the algorithm design is not restricted to this specific type of learning models, it cannot mix different types of learning models in the same ensemble.

#### 4.4.1.1 STAGGER and FLORA datasets

For fair comparison with CDC, we strictly duplicated the experiments of Stanley which were performed on STAGGER and FLORA artificial problems. The STAGGER problem simulate sudden concept changes while the FLORA problem consists of two datasets with slow and moderate concept changes, respectively.

In CDC, the maximum committee size was 10, the age of maturity for a committee member was 10, and the performance record for a particular hypothesis was taken over the last 10 instances it processed. The classifiers were decision trees trained incrementally on each observed example. The minimum performance level to avoid retirement was 80%.

The same settings were used for DACC, except for the 80% threshold value which is not used in DACC. Hence, to be comparable with CDC, we set $\tau_{eval} = \tau_{mat} = N = 10$. Besides, we chose the MAX function to combine the predictions of individual experts. As previously mentionned in Section 4.3.10.2, using different types of experts in the pool of predictors allows us to take advantage of the assets of each type of predictor while minimizing their drawbacks on the committee's classification performance, improving the overall predictive performance. However we didn't mix different types of experts in order to have a fair comparison with CDC which doesn't support more than one type of classifier in the same ensemble. Decision trees were used as in the experiments conducted by Stanley on CDC [85].

For evaluation purpose, with each training instance, the committee is asked to classify an independent test set labeled according to the underlying concept of the training instance,

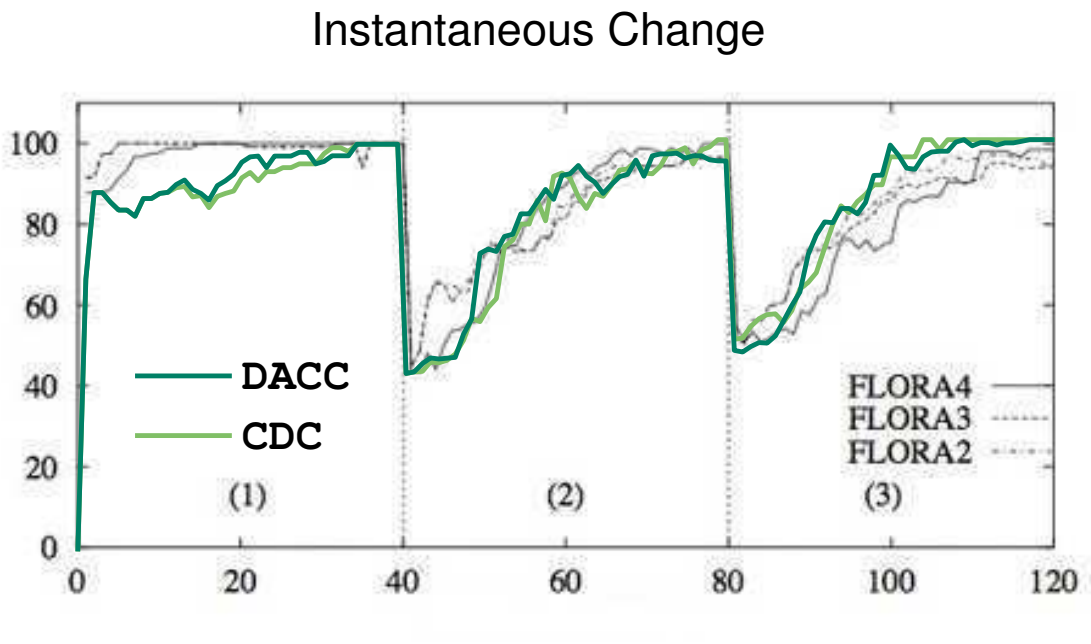


FIGURE 4.25: DACC vs CDC: a performance comparison on the STAGGER artificial dataset. The $x$-axis represents the training examples (time step) and the $y$-axis represents the percentage of classification accuracy on a test set labeled according to the underlying target concept of the training example.

and the percentage of correct classifications is recorded. In the FLORA datasets, during the drift between the two concepts, the testing instances are labeled according to the degrees of dominance $v_o$ and $v_n$ of the old and new concepts, according to Equation 2.3. The size of the testing set for each training instance is 100 for STAGGER and 200 for the FLORA datasets.

Stanley compared the classification performance of CDC with this of the different FLORA systems, explained in Section 3.2. We show in Figures 4.25, 4.26 and 4.27 the performance recorded for the different approaches on the STAGGER and FLORA problems, averaged over 10 instantiations of the artificial datasets. The results of the FLORA systems were taken from Stanley's paper. Only experiments on CDC and DACC were reconducted so they can be compared on the same testing sets.

The STAGGER dataset simulates instantaneous concept changes, allowing to assess how quickly algorithms can react to a sudden change. In sudden changes, any old acquired information should be forgotten quick enough if it becomes non genuine regarding the new concept. As shown in Figure 4.25, DACC and CDC have comparable results on the STAGGER dataset. After the drift, CDC resets its committee, allowing new members to learn the new concept. In DACC, members are also removed from the committee, but every $\tau_{mat} = 10$ time steps. Thus, the committee is not reset as quickly as in CDC after the drift. However, since the member(s) with the highest weight classifies the current instances in DACC, the committee doesn't need to be reset to give correct answers. Once

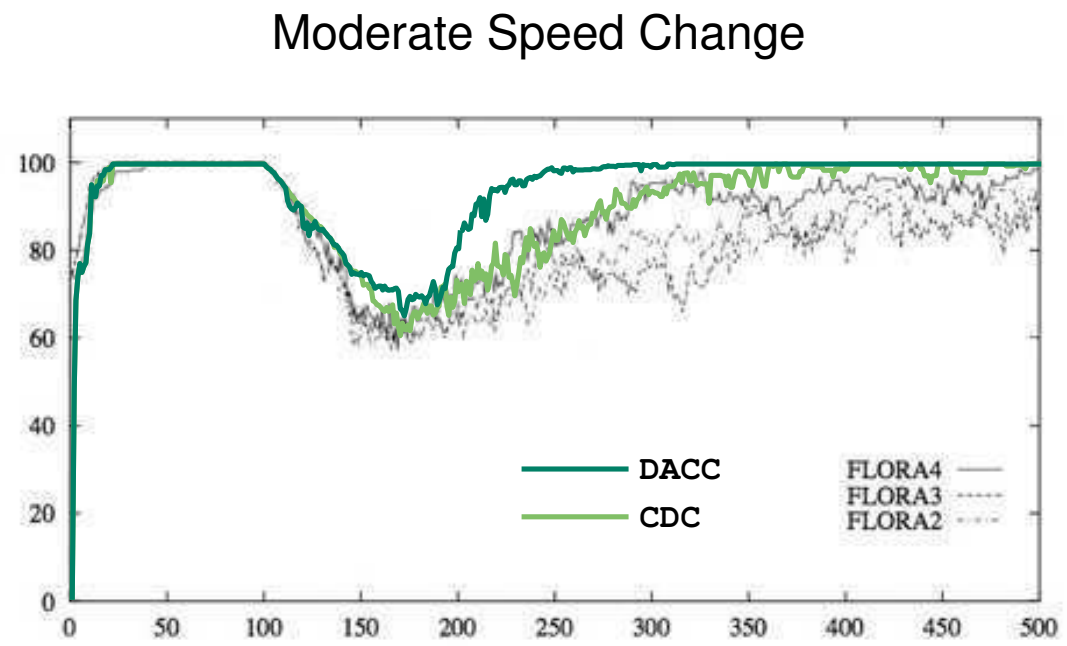


FIGURE 4.26: DACC vs CDC: a performance comparison on the FLORA artificial dataset, with a medium speed of change. The $x$-axis represents the training examples (time step) and the $y$-axis represents the percentage of classification accuracy on a test set labeled according to the underlying target concept of the training example.

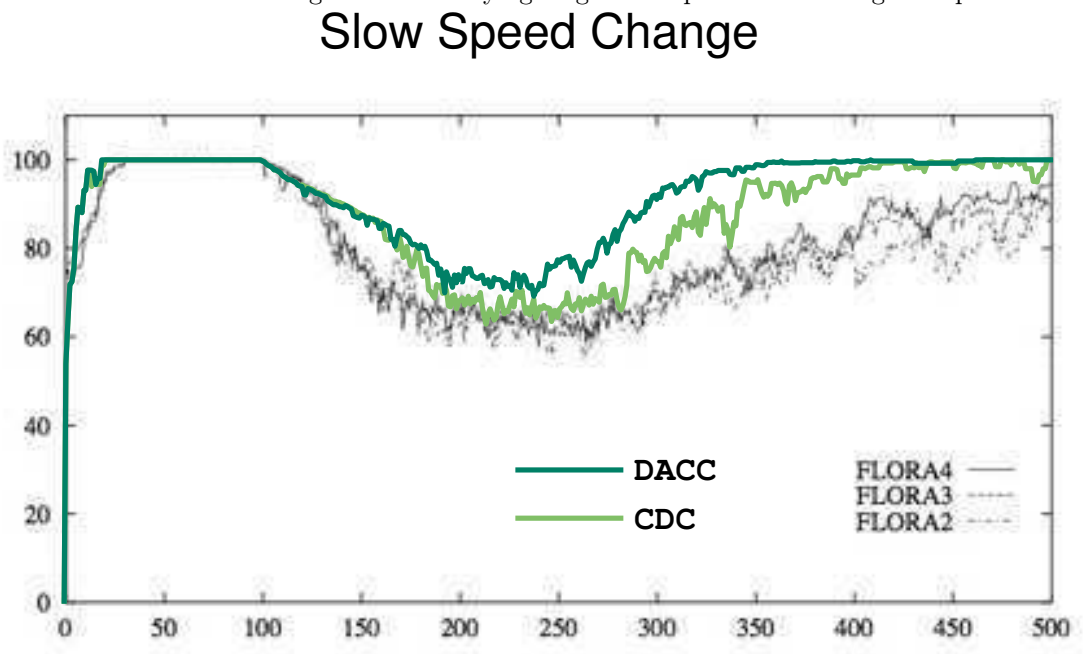


FIGURE 4.27: DACC vs CDC: a performance comparison on the FLORA artificial dataset, with a slow speed of change. The $x$-axis represents the training examples (time step) and the $y$-axis represents the percentage of classification accuracy on a test set labeled according to the underlying target concept of the training example.

the weight of a newly added member becomes higher than the others, the young member takes the lead in predicting the instances' class values.

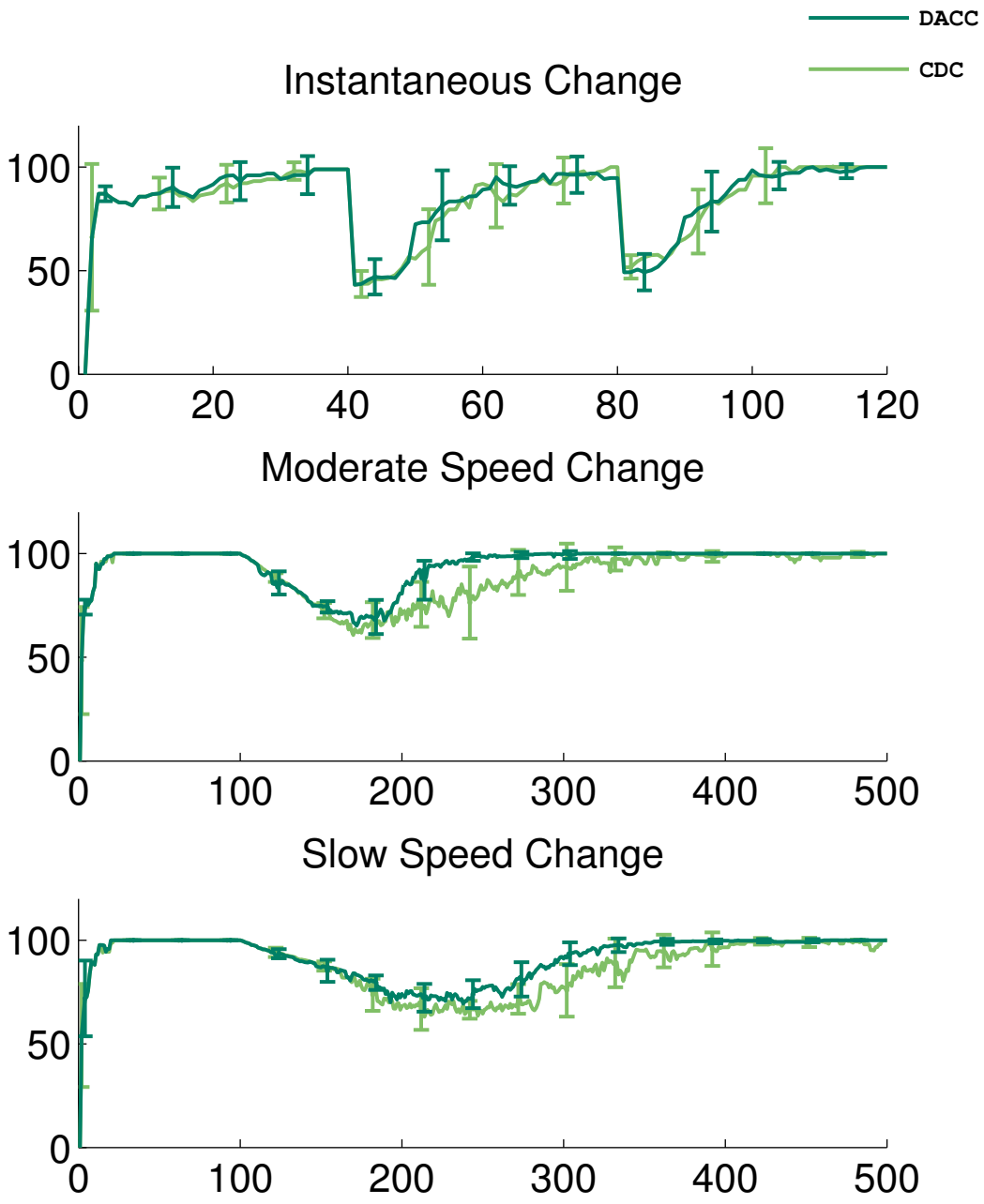


FIGURE 4.28: DACC vs CDC: a performance comparison on the STAGGER dataset (upper figure), and the FLORA datasets with medium (middle figure) and slow (bottom figure) speed of change. The $x$-axis represents the training examples (time step) and the $y$-axis represents the percentage of classification accuracy on a test set labeled according to the underlying target concept of the training example. We plot the mean predictive accuracy along with the standard deviation, as DACC and CDC are evaluated over 10 instantiations of the artificial datasets.

The FLORA datasets simulate continuous concept changes, allowing to evaluate the knowledge transfer ability of both approaches. From Figures 4.26 and 4.27, we notice that DACC outperforms CDC on the FLORA datasets, achieving better knowledge transfer than CDC during the drifting time. In fact, depending on the threshold value and the classification problem, the deletion strategy of CDC might reset the committee just after the drift which prevents knowledge transfer via old committee members. DACC however keeps its members for a longer time due to its frequency of deletion which, as pointed out before, forces the existence of old members for at least $N_{pool} * \tau_{mat}$ time steps after the beginning of the drift. We also notice that DACC recovers faster from the drift than CDC. In fact, after the end of the drift, members of the committee are trained on some examples from the old concept. This hurts their classification performance regarding the new concept but not enough to be removed from the committee by CDC. Since the classes are predicted by a weighted vote, the classification performance of CDC is affected by the presence of these members.

We show in Figure 4.28 the mean performance of DACC and CDC on the three problems, along with the standard deviations on the 10 instantiations of the artificial datasets.

#### 4.4.1.2 COLD dataset

DACC and CDC were also confronted with a real scenario where the task is to recognize the place of a robot exploring the Saarbrücken laboratory of the COLD database explained in Section 2.3.5.3. The robot visits 4 rooms in the following order: corridor, bathroom, corridor, one-person office, corridor, printer and corridor. The images from the dataset weren't used in their pixelated form. They were transformed into a feature space of dimension 128 using the method described by Guillaume et al in [42]. The image was first encoded using global descriptors. The encoded image was then projected into a Self-Organizing Map (SOM) that has been previously trained in an unsupervised way on 1/3 of the images of the COLD database. This offline training allows to learn a visual dictionary that describes the environment and should not be confused with the online supervised learning of the place classification task.

We show in Figure 4.29, the online performance of DACC and CDC on the robot dataset, averaged over 10 runs, with the following settings: $\tau_{mat} = \tau_{eval} = 10, N = 30$ and $cmb =$ MAX. CDC was tested with three different deletion thresholds: 0.6, 0.7 and 0.8. We reset the online performance with each visited place, indexed from $P_1$ to $P_7$.

We also show in Figure 4.30, for each approach, the time steps at which a member retires from the committee. We notice that, when visiting places $P_2$, $P_6$ and $P_7$, CDC reacts slower than DACC to the change. This is due to two reasons. First, since CDC uses

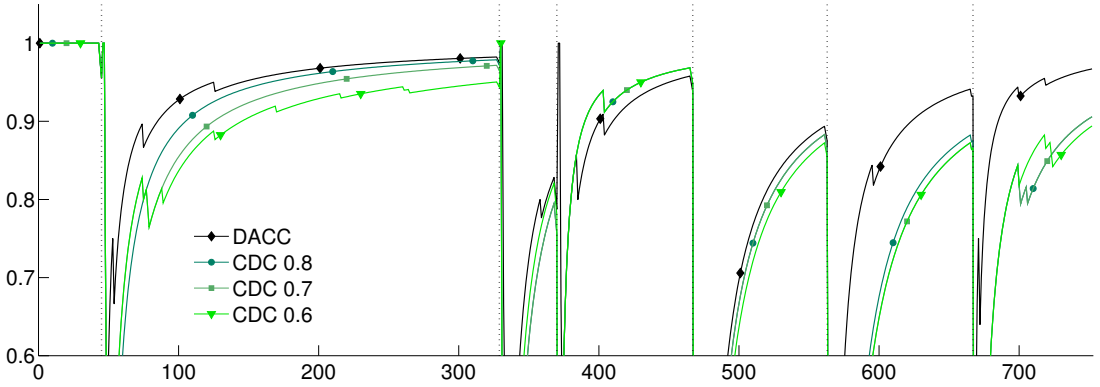


FIGURE 4.29: DACC vs CDC: the online classification performance on the Saarbrücken laboratory of the COLD database. The online performance is reset with each visited place.

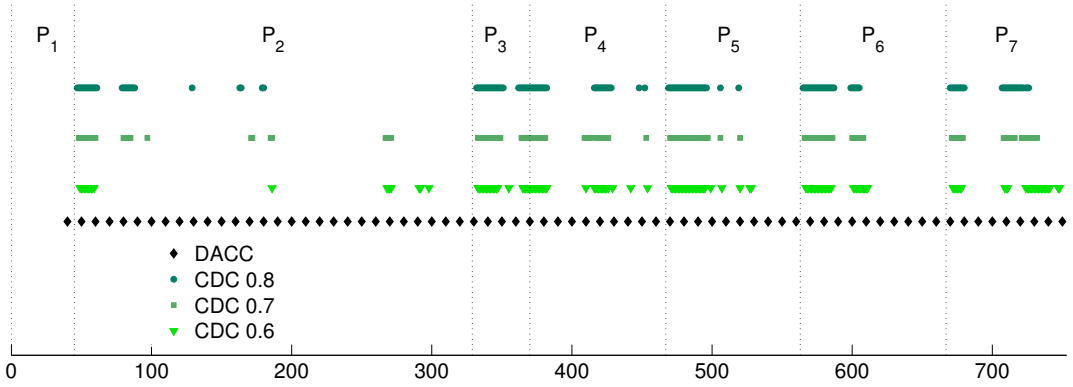


FIGURE 4.30: DACC vs CDC: the time step at which an expert is removed from the committee, when testing DACC and CDC on the Saarbrücken laboratory of the COLD database. The vertical dotted lines represent a new visited place.

a weighted vote, it needs to get rid of more than half the weight of its old members in order for young members -trained on the description of the new room- to win the vote. Secondly, DACC removes a member every $\tau_{mat}$ steps in anticipation of change. This allows DACC to react faster than CDC to a potential change. After entering $P_4$, CDC have a slightly better performance than DACC. As we see in Figure 4.30, it happened that CDC deleted some old members just before entering $P_4$ which made it ready for the upcoming change.

In this experiment, the robot visits the same place (corridor) many times, that is $P_1$, $P_3$, $P_5$ and $P_7$. We believe that in such case it would be useful for the robot to memorize the description of the visited rooms, instead of adapting passively to it. We discuss this further in Chapter 5.

TABLE 4.2: DACC vs CDC: The mean classification error computed over the experiments of Section 4.4.1, along with the predefined parameter values, with $\tau_{mat} = \tau_{eval}$.

| Dataset | DACC | | CDC 0.8 | | CDC 0.7 | | CDC 0.6 | | NoDriftH. | | Configuration |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | mean | std | mean | std | mean | std | mean | std | mean | std | $(\tau_{eval}, N, cmb)$ |
| CAR | 14.66 | 0.24 | 23.47 | 0 | 15.14 | 0 | **13.98** | 0 | 40.77 | 0 | 20,20,MAX |
| IRIS | **15.33** | 1.0 | 15.38 | 0 | 18.34 | 0 | 20.71 | 0 | 34.62 | 0 | 20,20,MAX |
| Circle | **6.92** | 1.43 | 35.66 | 1.12 | 8.09 | 3.5 | 8.73 | 0.78 | 12.56 | 3.45 | 20,20,MAX |
| Line | **3.72** | 1.34 | 8.12 | 1.68 | 4.64 | 1.25 | 7.01 | 2.89 | 10.52 | 4.19 | 20,20,MAX |
| Boolean | **2.75** | 1.75 | 3.85 | 1.38 | 4.17 | 1.29 | 4.47 | 1.21 | 17.63 | 10.26 | 20,20,MAX |
| SineH | **13.0** | 1.25 | 47.81 | 0.91 | 20.66 | 3.76 | 13.26 | 1.27 | 21.25 | 5.38 | 20,20,MAX |
| SineV | **4.32** | 1.12 | 7.98 | 1.65 | 5.15 | 1.18 | 5.85 | 1.69 | 11.08 | 4.52 | 20,20,MAX |
| Plane | 18.37 | 1.51 | 20.55 | 1.4 | **18.01** | 1.76 | 19.95 | 1.67 | 27.23 | 5.07 | 20,20,WVD |
| STAGGER | **14.33** | 3.11 | 16.08 | 1.88 | 17.75 | 2.15 | 20.25 | 3.31 | 32.52 | 5.09 | 10,10,MAX |
| FLORA-M | **5.34** | 0.99 | 10.19 | 3.43 | 6.32 | 1.28 | 6.02 | 1.12 | 16 | 1.18 | 10,10,MAX |
| FLORA-S | **7.7** | 1.36 | 12.46 | 2.71 | 9.21 | 1.34 | 9 | 1.25 | 18.9 | 0.92 | 10,10,MAX |
| COLD | **6.04** | 0.12 | 7.83 | 0 | 8.23 | 0 | 9.16 | 0 | 35.46 | 0 | 10,30,MAX |

#### 4.4.1.3 Other datasets

We show in Table 4.2 the mean classification error of DACC, CDC and a learning system that does not handle drifting concepts, on some of the artificial, semi-artificial and real problems described in Section 2.3.5.

The experiments used artificial, semi-artificial and real datasets. The base learners (experts) are decision trees. The system that does not handle drifting concepts is a single decision tree trained on every training example received. CDC is evaluated with three thresholds: 0.6, 0.7 and 0.8. For Minku's artificial problems: *Circle*, *Line*, *Boolean*, *SineH*, *SineV* and *Plane*, the error was averaged over the 9 different datasets of the corresponding problem. For STAGGER and FLORA problems, the error was averaged over 10 instantiations of the datasets. For IRIS, CAR and the COLD datasets, the error was averaged over 10 runs on the same dataset. Nevertheless, repeating the experiment multiple times with CDC and the *noDriftHandler* gave the same results since there are no randomness in the CDC algorithm nor in the decision trees learning models.

From Table 4.2, we notice that DACC has the smallest classification error in all cases, except for the CAR and *Plane* datasets. The results on the CAR dataset suggest that a deletion threshold of 0.6 is adapted to the CAR learning problem. Hence, removing experts with a classification accuracy smaller than 60% allows the ensemble to adapt to the simulated concept changes. For the *Plane* dataset, the slight difference between DACC and CDC 0.7 is likely due to the noise in the *Plane* dataset. Despite that WVD was used with DACC as a combination function to mitigate the effect of noise on the predictive performance, CDC 0.7 had still a lower classification error. In fact, in a noisy environment, even when an expert predicts correctly the label of an example, its predictions is considered wrong if the example's class was altered by noise. As a

result, the expert will be mistaken for a bad expert, affecting its weight. With CDC, an expert is not removed from the ensemble as long as its weight value doesn't go below the predefined threshold value. With DACC, however, a relatively small weight exposes the expert into the possibility of being removed from the ensemble. Nevertheless, this does not affect DACC's predictive performance tremendously. DACC's performance is the second best, ahead of CDC 0.6 and 0.8, and with a classification error larger than this of CDC 0.7 by a margin of 0.36% only.

In most cases, DACC gives better results than CDC. DACC overcomes the difficulty of finding the appropriate performance threshold according to which experts are considerate unadapted to the current concept. Fixing a threshold value not only requires a priori knowledge of the drift properties, it also assumes that the same threshold value is adapted to all upcoming concept drifts. This assumption does not hold generally.

### 4.4.2 DACC vs DDD, EDDM, DWM

The previous experiments showed the improvement brought by DACC over CDC. In this section, we compare DACC with most recent drift handling systems: (a) DDD (b) DWM and (c) a modified online bagging method using EDDM drift detector to adapt to evolving concepts. For simplicity, the latter method will be referred to as EDDM.

Diversity for Dealing with Drifts (DDD) is the online ensemble learning algorithm designed by Minku et al. to cope with evolving environments. DDD was previously evaluated on the six artificial problems *Line*, *Circle*, *SineH*, *SineV*, *Boolean* and *Plane*, against DWM, EDDM and online bagging. In online bagging, whenever a training example is available, it is presented $K$ times for each expert, where $K$ is drawn from a $Poisson(1)$ distribution. The classification is done by unweighted majority vote, as in offline bagging. In the remainder of this section, the online bagging system will be referred to as the *noDriftHandler* since it is not designed to handle drifting concepts.

Dynamic Weighted Majority (DWM) uses a weighted ensemble of experts, with the weights reflecting the experts classification performance during their lifetime. DWM does not use a drift detection method. It adapts implicitly to concept changes by adding and removing experts. The removal is controlled by a predefined threshold on the experts weights. In this way, new experts can be created to learn new concepts and poorly performing experts, which possibly learnt old concepts, can be removed. The Early Drift Detection Method (EDDM), on the other hand, detects concept drifts explicitly and the learning system is reset when a concept drift is confirmed by the detection system. The algorithms are described in Chapter 3 and the parameters of each approach are detailed in Table 4.3.

TABLE 4.3: The parameter description of the approaches discussed in Section 4.4.2.

| Approach | Parameter | Description |
|---|---|---|
| EDDM | $\alpha$ | Threshold that determines whether the warning level is triggered. |
| | $\beta$, $\beta < \alpha$ | Threshold that determines whether a drift is considered to be detected. |
| DWM | $\rho$, $\rho < 1$ | Multiplier constant for expert's weight decrease. |
| | $p$ | Interval of time steps in which the system can decrease weights, add or remove experts. |
| | $\theta$ | Threshold for removing experts based on their weights. |
| DDD | $\lambda_l$ | Parameter for encouraging low diversity in the underlying ensemble learning algorithm. |
| | $\lambda_h$ | Parameter for encouraging high diversity in the underlying ensemble learning algorithm. |
| | $W$ | Multiplier constant for the weight of the old low diversity ensemble. |
| | $\gamma$ | Parameter for the drift detection method to be used with the approach. |
| Modified Online Bagging | $\lambda$ | Parameter used by *Poisson* to encourage more or less diversity. |
| DACC | $\tau_{eval}$ | The size of the evaluation window used to compute the weights of the experts. |
| | $\tau_{mat}$ | Parameter defining the maturity age of an expert. |
| | $N_{pool}$ | The maximum number of *pools of type of experts* allowed in the ensemble. |
| | *cmb* | The combination function used which is either MAX or WVD. |

TABLE 4.4: The parameters choice of DDD, EDDM and DWM for the artificial datasets. $W = 1$, $\lambda_l = 1$ and $\theta = 0.01$ were fixed.

| Dataset | Values for Preliminary Experiments | | | | | Chosen Values |
|---|---|---|---|---|---|---|
| | $\lambda_h$ (DDD) | $\gamma$, $\beta$ (DDD, EDDM) | $\alpha$ (EDDM) | $\rho$ (DWM) | $p$ (DWM) | |
| Circle | {0.0005 0.001, 0.005, 0.01, 0.05, 0.1, 0.5} | {0.75, 0.85, 0.95} | {0.96, 0.97, 0.98, 0.99, 1.1, 1.2, 1.3} | {0.001, 0.01, 0.1, 0.3, 0.5, 0.7, 0.9} | {1, 10, 20} | $\lambda_h = 0.05$, $\gamma = \beta = 0.95$, $\alpha = 0.99$, $\rho = 0.5, p = 1$ |
| Plane | | | | | | $\lambda_h = 0.05$, $\gamma = \beta = 0.95$, $\alpha = 0.99$, $\rho = 0.5, p = 5$ |
| Boolean | | | | | | $\lambda_h = 0.1$, $\gamma = \beta = 0.95$, $\alpha = 0.99$, $\rho = 0.5, p = 10$ |

TABLE 4.5: The parameters choice of DACC for the artificial datasets.

| Dataset | Values for Preliminary Experiments | | | Chosen Values |
|---|---|---|---|---|
| | $\tau_{mat}, \tau_{eval}$ (DACC) | $N_{pool}$ (DACC) | *cmb* (DACC) | |
| Circle | {5, 10, 20} | {20, 25, 30} | {MAX, WVD} | $\tau_{eval} = \tau_{mat} = N_{pool} = 20, cmb =$ WVD |
| Plane | | | | $\tau_{eval} = 10, \tau_{mat} = 5, N_{pool} = 25, cmb =$ WVD |
| Boolean | | | | $\tau_{eval} = \tau_{mat} = 5, N_{pool} = 25, cmb =$ WVD |

For each approach and problem, Minku et al. [70] chose the parameter values that experimentally gave the best classification performance on an average of 5 preliminary runs. We did the same with DACC on the *Boolean*, *Circle* and *Plane* problems. For each problem, we chose the parameters $\tau_{mat}, \tau_{eval} \in \{5, 10, 20\}$, $N_{pool} \in \{20, 25, 30\}$ and $cmb \in \{\text{MAX}, \text{WVD}\}$ that gave the lowest mean classification error on 5 runs. The best parameter values for DDD, DWM and EDDM are shown in Table 4.4, while the best parameters values for DACC are shown in Table 4.5.

In the experimental setup, the experts were lossless ITI online decision trees [92]. In DWM, the decision trees are trained with every received training example. In DDD, EDDM and the *noDriftHandler*, the example is learnt $K$ times by a decision tree, where $K$ follows a Poisson($\lambda$) distribution. In EDDM and the *noDriftHandler*, $\lambda$ is set to 1, which is the value used for the original online bagging. In DDD, different $\lambda$ values create ensembles with different levels of diversity, with smaller $\lambda$ values entailing larger diversity levels. In DACC, the type of decision trees remained to be chosen. In all of our preliminary experiments, the experts were decision trees trained on each observed example, as with DWM. However, in case of the noisy *Plane* datasets, we noticed that training a decision tree on all the data it observed during its lifetime hurt its classification performance. Thus, for the *Plane* problem, we used three decisions trees in DACC's pool of types of experts:

- $\text{tr}_1$, a tree trained on each observed example $(\mathbf{x}, y)$;
- $\text{tr}_2$, a tree trained on $K$ copies of $(\mathbf{x}, y)$ where $K$ follows a $Poisson(\lambda)$ distribution;
- $\text{tr}_3$ which is the same as $\text{tr}_2$ but is pruned to reduce overfitting.

The effects of mixing the different types of decision trees on DACC's performance on the *Plane* dataset were previously discussed in Section 4.3.10.2. The value of $\lambda$ is set to 1 for $tr_2$, as with EDDM and the $noDriftHandler$.

In the experimental setup, the maximum ensemble size was set to 25 for EDDM, the *noDriftHandler* and DDD, while in DWM, the ensemble size is adjusted automatically. It is important to note that since DDD uses a maximum of 4 ensembles of size 25 each, the maximum number of decision trees was 100. As for DACC, the ensemble size is the product of two values: the size of the pool of types of experts, and the maximum number of pools allowed in the ensemble $N_{pool}$. The value of $N_{pool}$ was chosen according to the preliminary experiments. The size of the pool was equal to 1 for all datasets except for the *Plane* datasets where the pool contains the three types of decision trees.

We summarize in Table 4.6, the types of decision trees used for each approach and problem, along with the maximum number of decision trees.

TABLE 4.6: The type of decision trees used in the ensemble approaches discussed in Section 4.4.2, along with the maximum ensemble size.

| Dataset | Maximum number of experts/Types of experts in the ensemble | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | DACC | | DDD | | EDDM | | DWM | | NoDriftH. | |
| Circle | 20 | $tr_1$ | 25*4 | $tr_2$ | 25 | $tr_2$ | auto | $tr_1$ | 25 | $tr_2$ |
| Plane | 25*3 | $tr_1, tr_2, tr_3$ | 25*4 | $tr_2$ | 25 | $tr_2$ | auto | $tr_1$ | 25 | $tr_2$ |
| Boolean | 25 | $tr_1$ | 25*4 | $tr_2$ | 25 | $tr_2$ | auto | $tr_1$ | 25 | $tr_2$ |

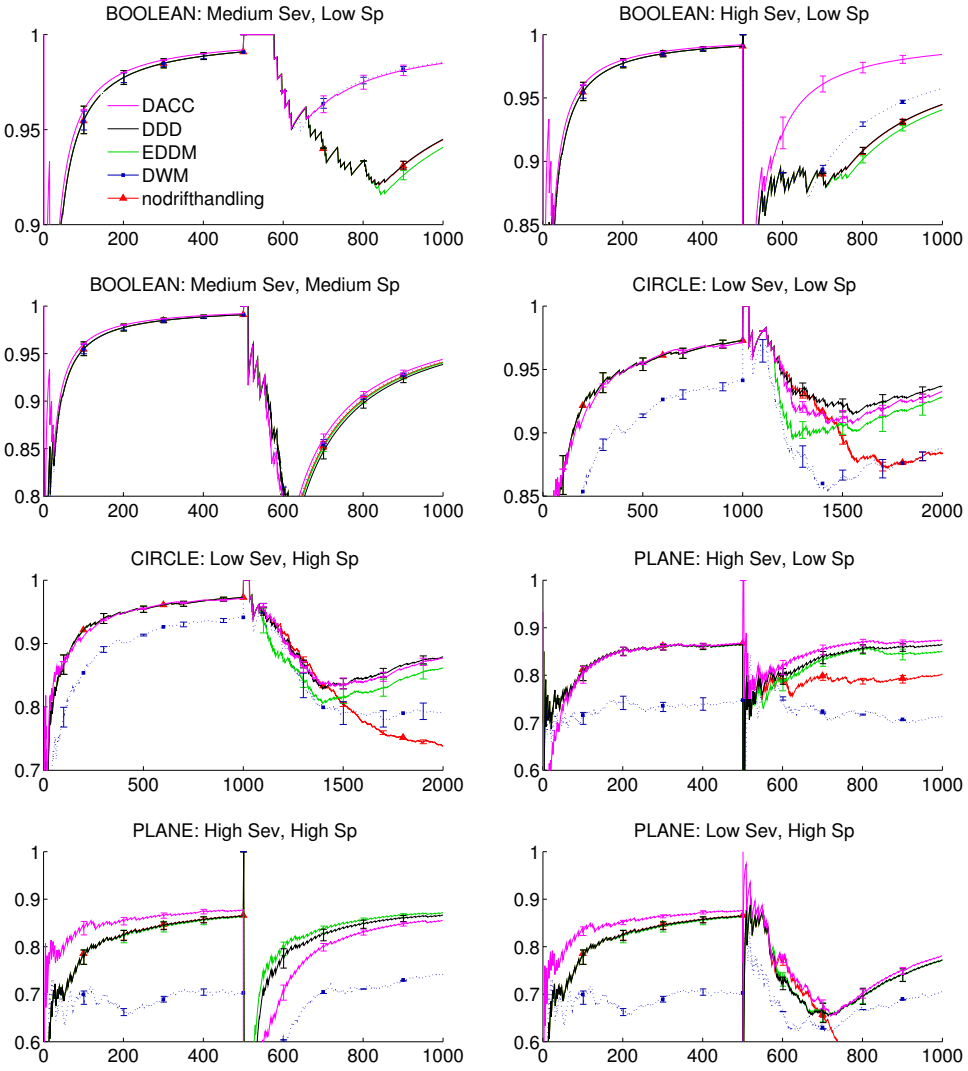


FIGURE 4.31: The online classification performance of DACC, DDD, EDDM, DWM and a no drift handling approach on a selected number of datasets from the *Boolean*, *Plane* and *Circle* artificial problems.

We show in Figure 4.31, the online performance of the different approaches on some of the datasets of the *Plane*, *Circle* and *Boolean* problems, averaged over 30 runs. While the power of DDD comes from using multiple ensembles, DACC's performance can be boosted by using different types of classifiers, as seen in the results on the *Plane* datasets. We notice however that DACC reacts slower than DDD and EDDM in case of high severity and speed. In fact, just after the change, members in DACC trained on

the old concept predict the classes of the new concept. Since the change is severe, the predictions are mostly wrong. In DDD and EDDM, however, the severe change is easily detected by the detection mechanism and a new ensemble is directly used to predict the upcoming labels.

In the *Boolean* problem, the approaches fail to adapt to the drift with low severity, which takes place without being detected. DACC however removes committee members periodically, regardless of whether the concept is stable or changing, allowing its member to adapt to any potential change. In the remaining results, DACC performs at least as well as the other approaches.

### 4.4.3 DACC vs others systems

Massive Online Analysis (MOA) is a software environment for implementing algorithms and running experiments for online learning from data streams [11]. MOA includes a collection of online learning (classification) methods, able to learn under stationary and/or evolving environments. In particular, MOA implements online boosting, bagging, Hoeffding trees, and Naive Bayes classifiers. MOA provides tools for evaluation, allowing to assess the classification performance as well as the execution time of a classification method.

We implemented DACC in the Java programming language, extending the MOA software. Our central goal in this section is to compare DACC with a selection of online learning methods available in MOA that can learn under evolving environments, adapting to drifting concepts. The online learning methods include:

- EDDM: a single learner using the early drift detection method described in Section 3.1.1. The learner is reset when a concept drift is detected.
- ABAG: the Adwin-bagging algorithm described in Section 3.3.5.
- ASHT: the ASHT-bagging algorithm described in Section 3.3.6.
- LBAG: the Leveraging-bagging algorithm described in Section 3.3.8.

We also evaluated the performance of two simple systems that don't learn from the data stream:

- Majority: this system predicts the label of the majority class observed so far.
- NoChange: this system predicts the label of the last observed class.

TABLE 4.7: The parameter description of the approaches discussed in Section 4.4.3.

| Approach | Parameter | Description |
|---|---|---|
| ASHT | $N_{ASHT}$ | The ensemble size. |
| | $f_1$ | A flag indicating whether the ensemble is weighted or unweighted. |
| | $f_2$ | A flag indicating whether a tree is reset after exceeding the maximum size value. |
| | $s$ | The maximum size value of the smallest tree in the ensemble. |
| LBAG | $N_{LBAG}$ | The ensemble size. |
| | $\lambda$ | Parameter used by *Poisson* to encourage more or less diversity. |
| | $\gamma$ | Parameter for the ADWIN drift detection method. |
| | $f_3$ | A flag indicating whether error-correcting output codes are used. |
| DACC | $\tau_{eval}$ | The size of the evaluation window used to compute the weights of the experts. |
| | $N_{DACC}$ | The maximum ensemble size. |
| | $cmb$ | The combination function used which is either MAX or WVD. |
| Adwin-BAG | $N_{ABAG}$ | The ensemble size. |
| MAJORITY | - | No parameters. |
| NoChange | - | No parameters. |
| EDDM | - | No parameters. |

#### 4.4.3.1 Experimental setup

We evaluated the different systems on the real datasets explained in Section 2.3.5.3, the SEA artificial dataset explained in Section 2.3.5.1, and the USENET semi-artificial dataset explained in Section 2.3.5.2. Before evaluating the different online systems, we should set their parameter values. Choosing random parameter values for each system is meaningless, entailing an unfair comparison. In our experimental framework, we decided to fine-tune the parameters of each method in order to compare the different online systems at their best. For each online system and data stream, we conducted preliminary experiments with different combinations of parameter values. The combination giving the highest classification performance was then selected to evaluate the corresponding system.

The parameters of each approach are shown in Table 4.7. The parameter values for the preliminary experiments are shown in Table 4.8. Note that EDDM has threshold parameters that trigger the drift detection alarms. These values are set automatically by the MOA software. Therefore, no parameters were fine tuned in EDDM.

The preliminary experiments were conducted as follows: for each data stream and online system, a preliminary experiment was run for each combination of the parameter values. According to Table 4.8, there are 72 combinations for LBAG, 48 for ASHT, 24 for DACC and 4 for ABAG. The preliminary experiments started with the same random seed in order to have the same learners at the beginning of all the experiments.

In our experiments, we evaluated the different systems using two types of base learners: an incremental Naive Bayes learner and a Hoeffding decision tree. The parameters of the Hoeffding decision trees were set to their default values in MOA.

We show in Tables 4.9 and 4.10 the best combinations of parameter values according to the preliminary experiments, using Hoeffding trees and Naive Bayes as base learners, respectively. Note that, when testing the LBAG ensemble method on the Forest dataset using Hoeffding trees, we excluded the combinations with $f_3 = 1$ because this option made the execution time very long ($> 1$ day). Note also that the parameters of ASHT were not tuned when using Naive Bayes learners since ASHT is designed to work only with Hoeffding trees.

After the parameters were chosen, we compared the different systems on the basis of two criteria: the mean classification error and the processing time (in CPU seconds).

#### 4.4.3.2 Empirical results

We show in Figures 4.32 and 4.33 the mean classification accuracy (in percentage) of the different online systems using either Hoeffding trees as base learners or incremental Naive Bayes learners, respectively. We also show in Table 4.11, the mean processing time in CPU seconds of the online ensemble methods, that is, DACC, ASHT, LBAG and ABAG. For each ensemble method, the processing time is averaged over all the preliminary experiments on all datasets.

In PAKDD, the task is to detect *fraudulent* customers which represent a minority class. Learning systems may have difficulties to learn the concept related to the minority class. Hence, to assess the performance of a learning system, it is better to compute the number of examples from the minority class that are correctly identified by the system. For instance, we can see from Figure 4.33 that the Majority system has the highest predictive accuracy. However, this indicator masks the fact that the Majority system does not detect any of the fraudulent clients. We computed the false negative rate (the number of *fraudulent* clients classified as *good* clients) for the Majority, LBAG and DACC which are the top three systems according to the predictive accuracy results. The false negative rate ($fn$) for Majority is 19.748% which represents the rate of the minority class. The $fn$ value for LBAG and DACC when using Hoeffding trees are close to this of the Majority system, suggesting that neither LBAG nor DACC learn the fraudulent clients. When using Naive Bayes learners, the $fn$ value improves, with an average value of 0.15% for DACC and 0.14% for LBAG. These results unveil the difficulty of learning in the presence of imbalanced classes. When learning from examples of the data stream, DACC gives the same importance to examples of each class, therefore making the assumption that

TABLE 4.8: The parameter values for preliminary experiments. For simplicity, we set $\tau_{mat} = \tau_{eval}$ for DACC.

| Values for Preliminary Experiments | | | | | | |
|---|---|---|---|---|---|---|
| $N_{LBAG}, N_{ASHT}$, $N_{ABAG}, N_{DACC}$ | $\lambda$ (LBAG) | $\gamma$ (LBAG) | $s$ (ASHT) | $f_1, f_2, f_3$ (ASHT, LBAG) | $cmb$ (DACC) | $\tau_{eval}$ (DACC) |
| $\{10, 20,$ $30, 50\}$ | $\{6, 10,$ $20\}$ | $\{0.01, 0.1,$ $0.002\}$ | $\{1, 2,$ $3\}$ | $\{0, 1\}$ | {MAX, WVD} | $\{10, 20,$ $50, 100\}$ |

TABLE 4.9: The parameter values chosen according to the preliminary experiments, using Hoeffding trees as base learners.

| Dataset | Chosen Values |
|---|---|
| PAKDD | $N_{ABAG} = 20, N_{DACC} = 10, \tau_{eval} = \tau_{mat} = 100, cmb = WVD$ $N_{LBAG} = 20, \lambda = 10, \gamma = 0.01, f_3 = 1, N_{ASHT} = 20, s = 1, f_1 = 0$ $f_2 = 1$ |
| KDD | $N_{ABAG} = 20, N_{DACC} = 50, \tau_{eval} = \tau_{mat} = 20, cmb = MAX$ $N_{LBAG} = 10, \lambda = 6, \gamma = 0.01, f_3 = 0, N_{ASHT} = 50, s = 2, f_1 = 0$ $f_2 = 1$ |
| Electricity | $N_{ABAG} = 30, N_{DACC} = 50, \tau_{eval} = \tau_{mat} = 10, cmb = MAX$ $N_{LBAG} = 20, \lambda = 6, \gamma = 0.1, f_3 = 0, N_{ASHT} = 10, s = 1, f_1 = 1$ $f_2 = 1$ |
| Airlines | $N_{ABAG} = 50, N_{DACC} = 50, \tau_{eval} = \tau_{mat} = 100, cmb = WVD$ $N_{LBAG} = 50, \lambda = 20, \gamma = 0.1, f_3 = 1, N_{ASHT} = 50, s = 1, f_1 = 1$ $f_2 = 1$ |
| Forest | $N_{ABAG} = 20, N_{DACC} = 50, \tau_{eval} = \tau_{mat} = 10, cmb = MAX$ $N_{LBAG} = 50, \lambda = 6, \gamma = 0.002, f_3 = 0, N_{ASHT} = 30, s = 2, f_1 = 1$ $f_2 = 1$ |
| Ozone | $N_{ABAG} = 10, N_{DACC} = 10, \tau_{eval} = \tau_{mat} = 100, cmb = WVD$ $N_{LBAG} = 30, \lambda = 6, \gamma = 0.002, f_3 = 1, N_{ASHT} = 10, s = 1, f_1 = 1$ $f_2 = 1$ |
| Usenet | $N_{ABAG} = 10, N_{DACC} = 30, \tau_{eval} = \tau_{mat} = 50, cmb = MAX$ $N_{LBAG} = 20, \lambda = 6, \gamma = 0.1, f_3 = 0, N_{ASHT} = 20, s = 1, f_1 = 0$ $f_2 = 1$ |
| SEA | $N_{ABAG} = 10, N_{DACC} = 10, \tau_{eval} = \tau_{mat} = 100, cmb = MAX$ $N_{LBAG} = 10, \lambda = 10, \gamma = 0.1, f_3 = 0, N_{ASHT} = 50, s = 1, f_1 = 1$ $f_2 = 1$ |

TABLE 4.10: The parameter values chosen according to the preliminary experiments, using Naive Bayes learners.

| Dataset | Chosen Values |
|---|---|
| PAKDD | $N_{ABAG} = 10, N_{DACC} = 50, \tau_{eval} = \tau_{mat} = 10, cmb = MAX$ $N_{LBAG} = 10, \lambda = 10, \gamma = 0.1, f_3 = 1$ |
| KDD | $N_{ABAG} = 10, N_{DACC} = 10, \tau_{eval} = \tau_{mat} = 10, cmb = MAX$ $N_{LBAG} = 10, \lambda = 6, \gamma = 0.01, f_3 = 1$ |
| Electricity | $N_{ABAG} = 10, N_{DACC} = 50, \tau_{eval} = \tau_{mat} = 10, cmb = MAX$ $N_{LBAG} = 10, \lambda = 10, \gamma = 0.1, f_3 = 0$ |
| Airlines | $N_{ABAG} = 30, N_{DACC} = 50, \tau_{eval} = \tau_{mat} = 100, cmb = WVD$ $N_{LBAG} = 50, \lambda = 6, \gamma = 0.002, f_3 = 1$ |
| Forest | $N_{ABAG} = 10, N_{DACC} = 50, \tau_{eval} = \tau_{mat} = 10, cmb = MAX$ $N_{LBAG} = 10, \lambda = 6, \gamma = 0.1, f_3 = 0$ |
| Ozone | $N_{ABAG} = 10, N_{DACC} = 10, \tau_{eval} = \tau_{mat} = 20, cmb = MAX$ $N_{LBAG} = 10, \lambda = 20, \gamma = 0.1, f_3 = 0$ |
| Usenet | $N_{ABAG} = 10, N_{DACC} = 30, \tau_{eval} = \tau_{mat} = 100, cmb = MAX$ $N_{LBAG} = 10, \lambda = 20, \gamma = 0.1, f_3 = 0$ |
| SEA | $N_{ABAG} = 10, N_{DACC} = 50, \tau_{eval} = \tau_{mat} = 100, cmb = MAX$ $N_{LBAG} = 10, \lambda = 6, \gamma = 0.01, f_3 = 0$ |

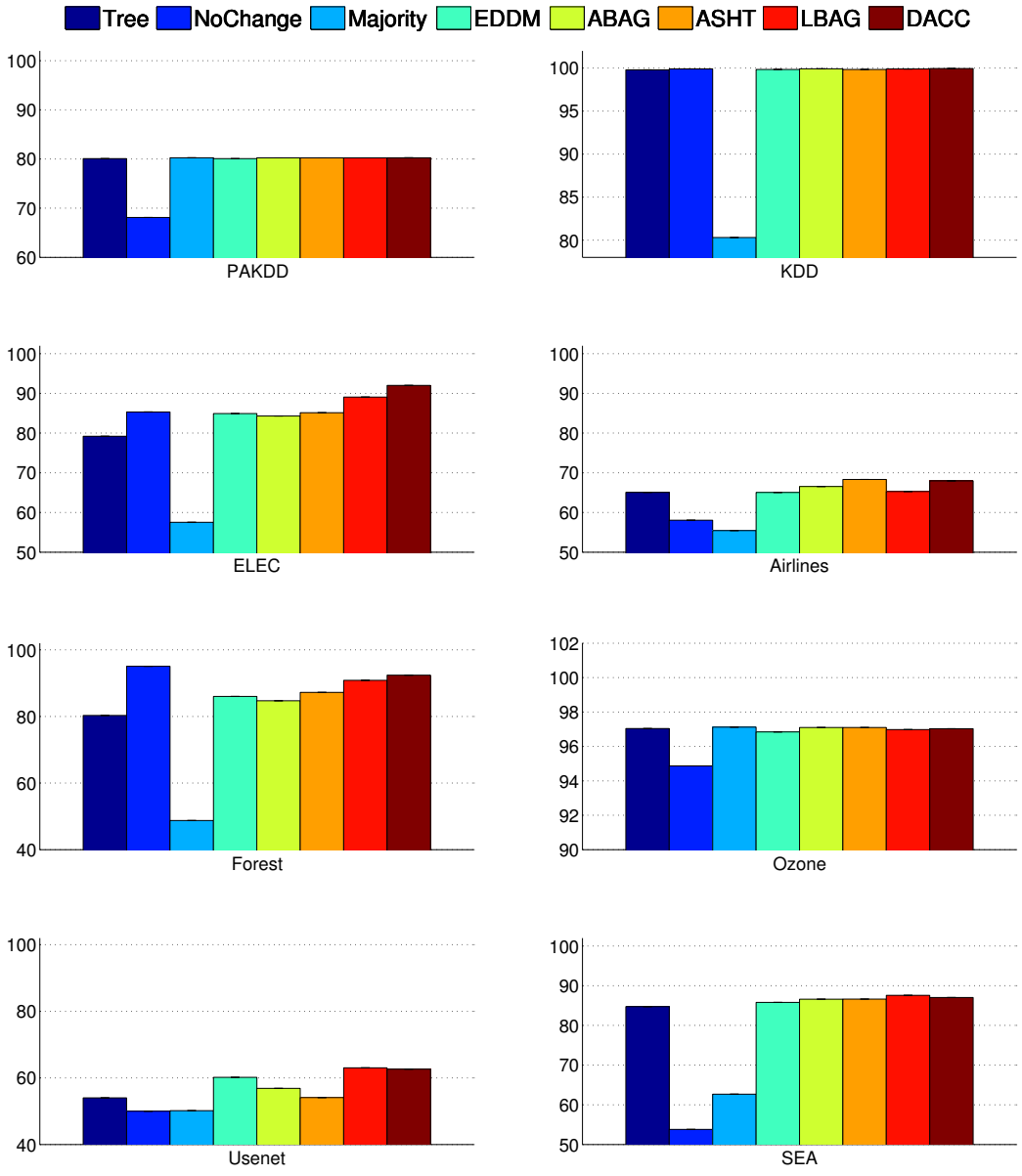


FIGURE 4.32: The mean classification accuracy (in percentage) of different online systems using Hoeffding trees as base learners.

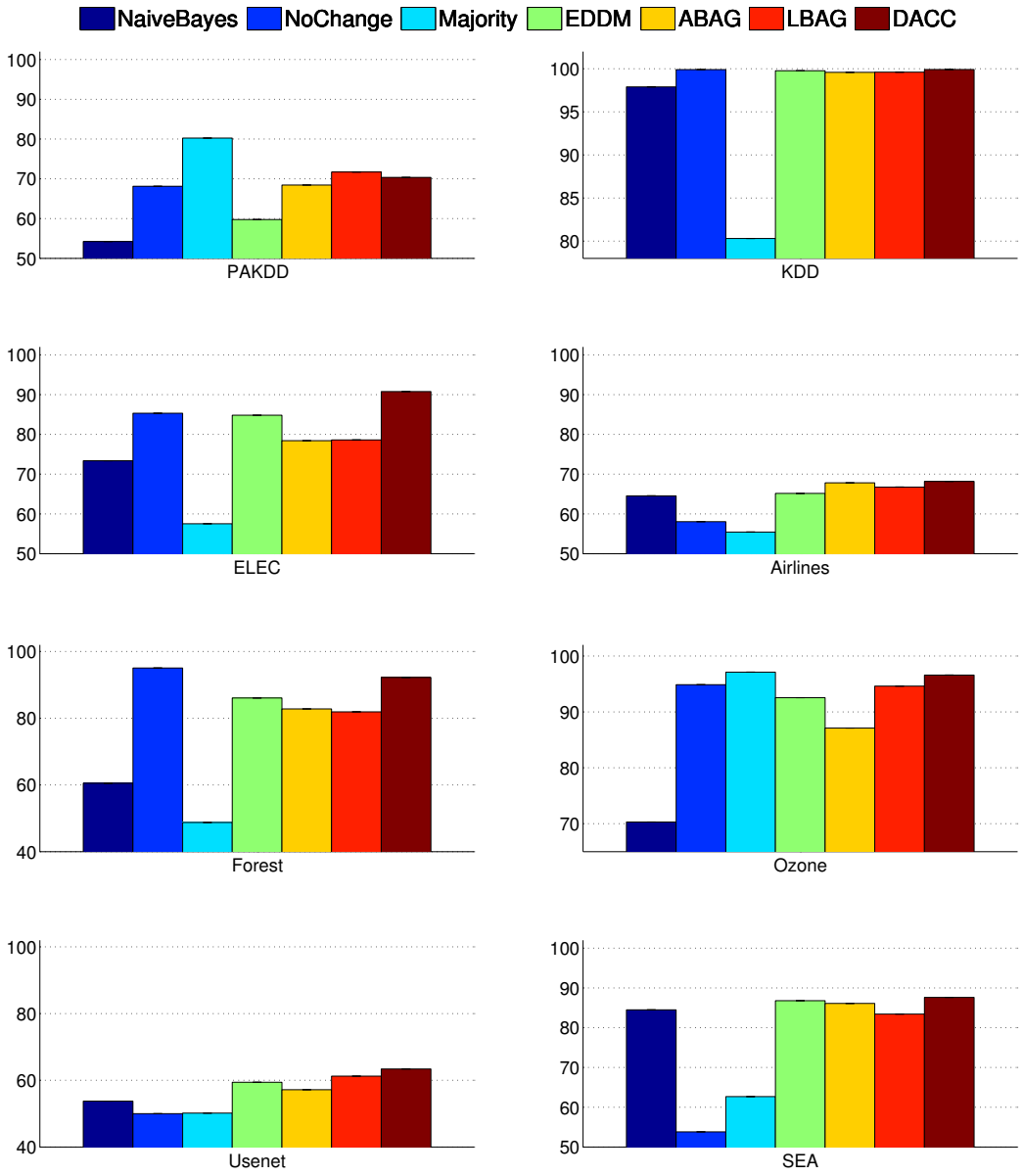


FIGURE 4.33: The mean classification accuracy (in percentage) of different online systems using Naive Bayes learners.

TABLE 4.11: The processing time (in CPU seconds) of the ensemble methods averaged over the preliminary experiments on all datasets explained in Section 4.4.3.

| **Algorithm** | **base learner** | **mean** | **std-dev** |
|---|---|---|---|
| DACC | Hoeffding decision tree | 105.32 | 163.02 |
| LBAG | Hoeffding decision tree | 236.83 | 468.51 |
| ABAG | Hoeffding decision tree | 99.23 | 170.72 |
| ASHT | Hoeffding decision tree | 62.164 | 107.08 |
| DACC | Naive Bayes | 37.04 | 61.01 |
| LBAG | Naive Bayes | 53.29 | 81.73 |
| ABAG | Naive Bayes | 50.09 | 80.89 |

classes are balanced. A possible solution is to assign weights to training examples that are inversely proportionnal to the observed frequency of the examples' class, giving more importance to minority classes.

In the SEA and Usenet datasets, artificial concept changes are simulated. We notice that DACC has the best classification accuracy when using Naive Bayes as base learners. As with Hoeffding trees, LBAG is slightly better than DACC by a margin of 0.4% for Usenet and 0.55% for SEA.

In the Electricity dataset (ELEC), concept changes are expected since power demands and supply might evolve with time. This assumption is confirmed by the classification accuracy of the single incremental learners which is outperformed by adaptive learning systems. DACC outperforms the other systems when learning from ELEC.

In the Forest dataset, concept changes also occur, as in ELEC. In addition, the examples are time-correlated according to the NoChange system. While DACC outperforms other adaptive systems, its performance is not as high as this of the simple NoChange system.

In both Ozone and Airlines, the concept evolves according to the accuracy of the incremental learners compared to this of adaptive systems. The data examples are also time-correlated according to the NoChange system. When using Naive Bayes learners, DACC outperforms other systems. With Hoeffding decision trees, the ASHT system, designed to work specifically with Hoeffding trees, gives the best results.

As for the KDD dataset, its examples are highly time-correlated with the NoChange system having an accuracy of 99.9%. None of the adaptive systems gives better results, except DACC which has a classification accuracy of 99.95% and 99.91% using Hoeffding trees and Naive Bayes learners, respectively.

According to the experimental results, DACC does not give the best results on all evaluated datasets. Nevertheless, it is always among the best performing approaches, suggesting that it can generally adapt to a large variety of problems. When DACC is not the best, its classification performance is in average 0.5% less than the best adaptive approach. The other adaptive systems, LBAG, ABAG, EDDM and ASHT have a classification performance that is in average 3.6, 4.5, 3.6, and 3.7% less than the best adaptive approach, respectively.

The LBAG system seems to be the first rival of DACC. However, its performance bounces. For instance, in the ELEC dataset, the classification accuracy of LBAG goes from 89.06% using Hoeffding trees to 78.61% using Naive Bayes learners. A main disadvantage of LBAG is its high execution time which is nearly 2.25 times the execution time of DACC when using Hoeffding trees, 1.4 times when using Naive Bayes learners (see Table 4.11).

## 4.5 Contribution

In this chapter, we presented DACC, an ensemble method for online learning in the presence of concept changes. The suggested method controls a diverse committee of experts whose memory of past information evolves dynamically with time via experts deletion and addition operations.

Unlike other ensemble methods that remove the worst expert in the committee [12, 13, 59, 85, 90], our method periodically removes experts chosen randomly from the worst half of the ensemble. We showed that this deletion strategy makes the adaptive ensemble less sensitive to its preset parameters.

Managing past information is also ruled by the way the experts' votes are combined in the committee's final decision. We suggested two combination functions: MAX and WVD depending on whether the leaning environment is noisy or noise-free. Both methods allow only the best experts to vote. The deletion strategy along with the combination function create a competitive and cooperative environment where experts compete for their life in the committee and the best ones cooperate for the final decision.

A main feature of DACC is its ability of managing different types of learning models in the same committee. This improves the overall classification accuracy as compared to the use of a single type of base learner, which is the general case for online ensemble methods [12, 13, 52, 59, 84, 85, 90].

Finally, our method deletes experts frequently without explicitly analyzing whether the underlying target concept is evolving. This has two advantages. First it overcomes the difficulties of relying on explicit drift detection systems inherent in having to detect gradual concept changes while still being robust to false alarms. Secondly, it makes the committee always ready to any upcoming concept change and thus allows it to react faster to changes than implicit approaches that generally wait until the expert's weight goes below a threshold before it is deleted [59, 85].

In the next chapter, we go beyond the passive adaptation to concept changes. We present a meta-learning mechanism that anticipates near-future concepts by analyzing how the learning models evolve in the adaptive ensemble. The anticipative and the adaptive approaches are combined in a new ensemble method called ADACC.

# Chapter 5

# Anticipating Concept Changes

Adapting to concept changes is important when considering unending data streams and long-life learning. When the environment changes, as is increasingly the case, it is necessary to rely on on-line learning with the capability to adapt to changing conditions a.k.a. concept drifts. Our ensemble method, DACC, presented in Chapter 4, adapts *passively* to concept changes without an insight to the future. This is also the case of the majority of the algorithms designed to handle changing environments. They only focus on how to *detect* changes and how to *adapt* to them.

Passive adaptation to concept changes may not be the best learning strategy. Indeed, a learner may profit from the information possibly conveyed by the very sequence of data and *anticipate* the upcoming changes, predicting *how* a concept may look like in the near future. In an evolving environment, two aspects should be considered when anticipating future concept changes: *recurrence* and *predictability*.

Recurrence means that the same concepts (or close approximations) might reappear with time, either in a cyclic manner (e.g. seasonal variations) or in an irregular manner (e.g. inflation rate, market mood). Predictability means that the concept evolves in a predictable manner, but without necessarily repeating over time. Here, "predictable" refers to an underlying prediction system, that is a learning system that, taking as input information about the past history of the concept evolution, is able to predict its (near) future. One can gain precious time and avoid costly incorrect predictions by being able to recognize a recurring situation or to anticipate the likely evolution to come along.

Anticipating concepts changes is the subject of this chapter. We introduce a second-order learning mechanism that is able to *detect* relevant states of the environment, to *recognize* recurring contexts and to *predict* likely concepts changes. We couple DACC with this meta-learning mechanism, allowing DACC to "guess" how the concept will look like in

the near future. This type of second-order learning provides means for the anticipation of, and the quick adaptation to, the underlying modification of the context. When the upcoming concept is anticipated, DACC can act proactively, preparing classification strategies in advance, enhancing its predictive performance. It is important to ensure that the anticipation mechanism does not hurt the performance of DACC's adaptative ensemble. In other words, if the anticipation mechanism fails to predict the next concept, the performance of DACC should be the same as using the mere adaptive strategy.

This chapter is organised as follows. Section 5.1 deals with the aspect of *predictability*. We present DACCv1 (Section 5.1.1), a basic approach able to anticipate predictable concept changes. The following sections then describe two extensions of the basic method: the algorithm DACCv2 (Section 5.1.2) possesses the ability to detect relevant states of the environment and predicts simple concept evolutions while DACCv3 (Section 5.1.3) can predict more complex evolutions. Then, Section 5.2 deals with the aspect of *recurrence*. A more elaborated method, DACCv4 (Section 5.2.1), stores concepts for later use in case previously observed concepts reappear. We call ADACC (*Anticipative Dynamic Adaptation to Concept Changes*) the new ensemble method that combines the adaptive ensemble of DACC with the meta-learning mechanism, dealing with both *predictability* and *recurrence*. The detailed algorithm and its realization in experimental systems are presented in Section 5.3.

## 5.1 Concept Predictability

As an illustration of concept predictability, consider a concept corresponding to a circle in a 2-dimensional input space. The instances are labeled into two classes depending on whether they lie inside or outside the circle (see Figure 2.4). Suppose now that the radius is fixed while the circle's center moves with a constant speed, creating a concept drift. It might be the case that a learning system be able to learn from such a sequence of concept changes and predicts the likely future concepts. Note however that in such a case, the same circle doesn't repeat over time and there is no recurrence of past concept thus precluding the use of Markov chains or of any method using frequency measures.

In addition to being illustrative of a simple scenario, this example shows also why an approach commonly thought about in sequence learning, that of Markov chains, is here helpless. Learning a Markov chain requires indeed that the concepts be encountered repeatedly in order to reliably estimate transition matrices. When concepts do not recur, this is impossible.

Predicting how the concept evolves with time demands handling two main issues. First, we should be able to capture and represent the past history of the stable concepts and secondly, depending on the observed changes between the consecutive concepts, we should be able to predict the next concept. Representing the past history of the stable concepts raises many questions:

- How can we *recognize* the stable concepts during the learning process?
- How can we *represent* the underlying stable concept? Can we take a "snapshot" of the current concept via the adaptative ensemble?
- The adaptative ensemble comprises diverse experts, representing rather different concepts. How will the ensemble return *one* snapshot of the current concept?
- How can we verify that the snapshot returned by the ensemble represents a *stable* concept? The ensemble might not have learnt enough training examples from the underlying target concept or it might be adapting to a new concept when the snapshot is taken.
- In the history of stable concepts, two consecutive concepts should be different. How can we ensure that there is a *concept change* between two consecutive concepts?

**Basic prediction scenario** We start our experiments by designing a predictable concept change scenario. We simulate 10 concept changes by generating a series of 11 consecutive concepts. The concept is a linear decision boundary (a line) that separates the samples in a 2-dimensional feature space into two classes 0 and 1 (see Figure 5.1). For each concept, a sequence of 500 training examples are generated as follows: 500 pairs $\mathbf{x} = (x_1, x_2)$ are generated randomly in the feature space where $x_1, x_2 \in [-1, 1]$, and the corresponding class $y$ is assigned as:

$$y = sign(a_0 + \sum_{i=1}^{2} a_i x_i)$$

where

$$sign(n) = \begin{cases} 0 & \text{if } n < 0 \\ 1 & \text{otherwise} \end{cases}$$

and $a_0$, $a_1$ and $a_2$ are the parameters related to current concept.

We show the evolution of the concept parameters in Figure 5.2. The parameters $a_0$ and $a_2$ are fixed during the whole process. It is only the parameter $a_1$ that switches from $-1$

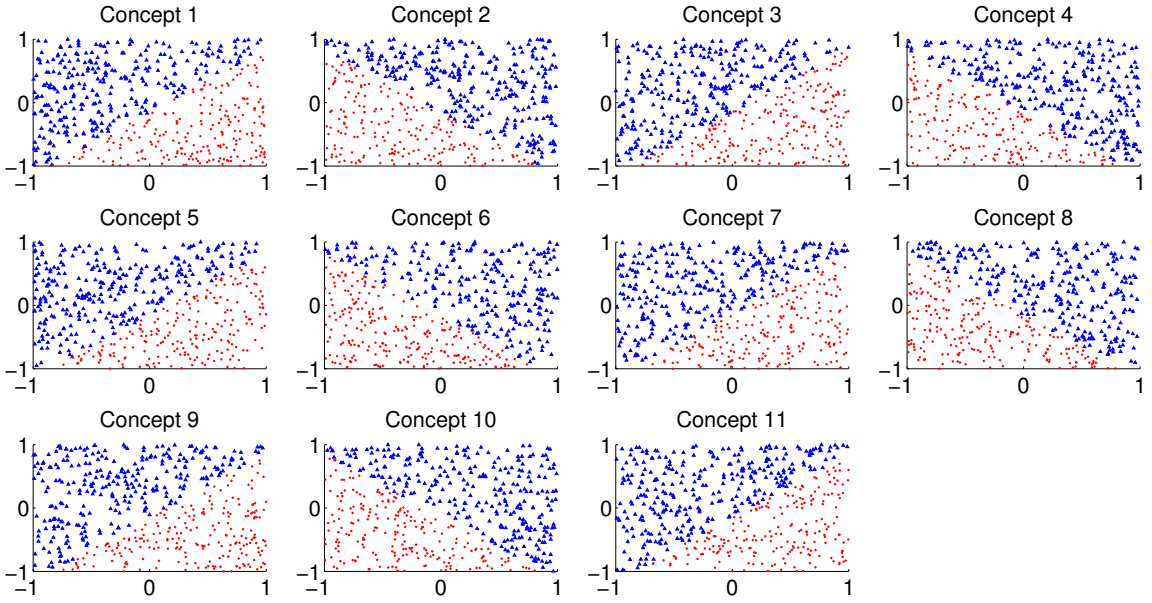


FIGURE 5.1: The data of the 11 consecutive concepts of Stream1. The data are separated into 2 classes depending on the parameters of the current concept.

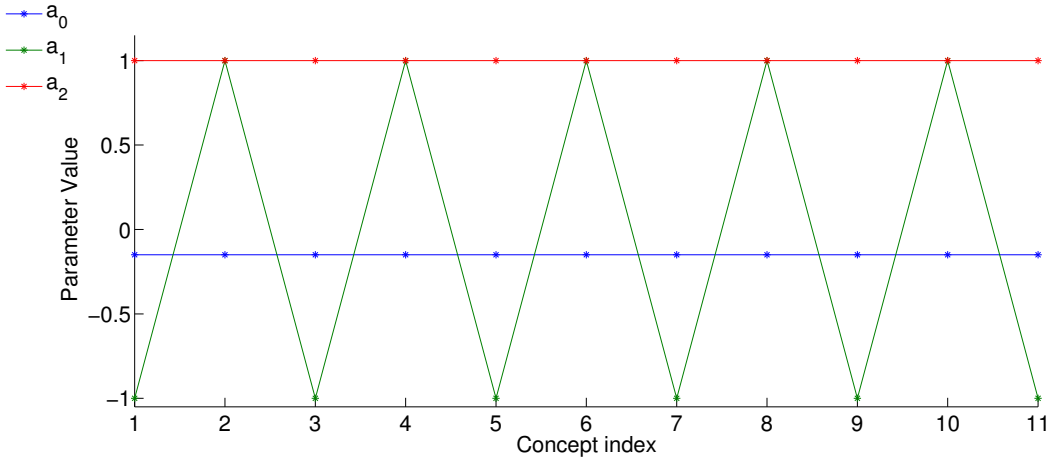


FIGURE 5.2: The parameters of the evolving concepts of Stream1.

to 1 (and vice versa) creating concept changes. The stream of data that results from concatenating the training sequences of the 11 consecutive concepts will be referred to as Stream1. It has a total of 5500 training examples.

We ran DACC on Stream1 using perceptrons [97] as learning models. The perceptron has an input layer of size 2 (the dimension of the feature space), and an output layer with one neuron that gives the class of the sample. The perceptron output is given as: $f(\mathbf{x}) = f(x_1, x_2) = sign(w_1 \times x_1 + w_2 \times x_2 + b)$, where $b$ and $w_1, w_2$ are the perceptron's parameters, with $b$ called the bias and $w_1, w_2$ the input weights. We chose perceptrons

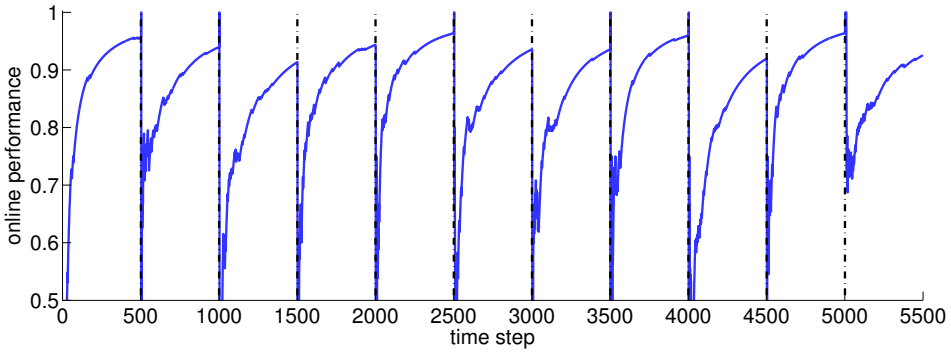


FIGURE 5.3: The online classification performance of DACC on Stream1. The online performance is reset each 500 timesteps, with each concept change. The vertical bars represent the time steps at which concept changes occur.

in order to capture the simulated evolution in the concept parameters. In fact, it is expected that after learning enough training data, the parameters $[b, w_1, w_2]$ of the perceptron mirror the values of the concept's parameters $[a_0, a_1, a_2]$.

The remaining parameters of DACC are the following:

- The committee holds a maximum of 10 experts i.e. $N = 10$.
- The experts are evaluated on their last 20 classifications i.e. $\tau_{eval} = 20$.
- At each time step, the expert with the best classification record is selected to predict the class of the incoming example i.e. $cmb = MAX$.
- The deletion strategy removes an expert every 20 time steps i.e. $\tau_{mat} = 20$.

All the experiments in this chapter were designed and executed on MATLAB unless otherwise indicated.

We show in Figure 5.3 the online classification performance of DACC on Stream1. The online performance is reset each 500 timesteps, with each concept change. We can see that DACC is able to learn each of the consecutive concepts after observing enough training data, improving its predictive accuracy with time when new concepts appear. However, as most of the other adaptation algorithms, DACC adapts blindly to the concept changes without an insight to the future.

In the following section, we present a basic method that aims to predict how concepts evolve with time.

### 5.1.1 DACCv1

We present DACCv1, a basic anticipation method that extends DACC's algorithm by predicting future concepts. DACCv1 builds the list of stable concepts encountered during the data streaming. This list provides means to analyze the changes between the consecutive concepts and anticipate the near future.

In the following, we first describe how the list of concepts is built. Then, we explain how concepts are *predicted* and *used* to classify instances from the data stream. Experimental results on Stream1 allow us to assess the main drawbacks of our basic approach.

#### 5.1.1.1 Building the history of concepts

DACCv1 makes the simple assumption that a concept change happens periodically every $\tau_{drift}$ time steps. In order to build the history of stable concepts, DACCv1 asks the adaptive ensemble to provide it with a representation of the current concept every $\tau_{drift}$ time steps. The concept returned by the ensemble is a *snapshot* (a copy) of the base learner with the highest weight in the ensemble, the one that seems to best represent the underlying target concept.

To simplify the discussion, the snapshot of a concept $C$ is represented as a vector of parameters of dimension $n$.

$$C = [c_1, c_2, ..., c_n]$$

The type of parameters depends on the learning model used to learn the target concept. For instance, if the current target concept is learnt by a learning model that is a neural network [43], the concept snapshot could be represented as the vector of the network's weight values. A pair consisting of two consecutive snapshots $\delta_i = (C_i, C_{i+1})$ is called a *concept change sample*.

At time step $k \times \tau_{drift}$, $k$ concept snapshots are provided and are stored in the list $\mathcal{M}_{LT}$:

$$\mathcal{M}_{LT} : \{C_1, C_2, C_3, \ldots, C_k\}$$

#### 5.1.1.2 The second order learning

The list of past snapshots $\mathcal{M}_{LT} = \{C_1, C_2, \ldots, C_k\}$, ordered according to the snapshots' time appearance, is the basis of the second order learning mechanism. It provides a sequence of successive models of the environment that can be used by a learning algorithm in order to predict the most likely future snapshot in the series.

In our experiments, we used Elman's neural network [21] to predict $C_{k+1}$. Elman networks are special types of recurrent neural networks that exhibit dynamic temporal behavior. Our use of Elman's networks as predictors was motivated by their generality and the good performances reported for learning tasks similar to our's [98].

An Elman network is trained on the pairs of consecutive snapshots in $\mathcal{M}_{LT}$ in order to predict the next likely snapshot. Hence, the network is trained on the pairs of change samples $\{\delta_1, \delta_2, \ldots, \delta_{k-1}\}$, learning the relationship between subsequent concepts. By running the network on $C_k$, the network returns its prediction of the next concept $\tilde{C}_{k+1}$. The predicted concept $\tilde{C}_{k+1}$ is used to classify the instances of the data stream, and this, until a new concept is predicted. Therefore, $\tilde{C}_{k+1}$ is used for classification from time step $k \times \tau_{drift} + 1$ to $(k+1) \times \tau_{drift}$.

For instance, when $\tau_{drift} = 200$, there are two snapshots at time step 400: $C_1$ and $C_2$, forming the first change sample and training example $\delta_1 = (C_1, C_2)$. The Elman network is trained on $\{\delta_1\}$ and is simulated on $C_2$ to make its first prediction, the concept $\tilde{C}_3$. The concept $\tilde{C}_3$ is used to classify the instances of time steps $[401, 600]$. At time step 600, the snapshot $C_3$ adds a new training example $\delta_2 = (C_2, C_3)$ and the Elman network is retrained on $\{\delta_1, \delta_2\}$ before predicting the next concept $\tilde{C}_4$, and so on.

Note that, in this context, only one type of learning models is considered to be present in the adaptive ensemble, in order for the snapshots to have the same dimensions when presented to the Elman network, and also to make concept change samples meaningful, capturing the evolution between the same set of parameters.

#### 5.1.1.3 Empirical results

We tested DACCv1 on Stream1 with values of $\tau_{drift}$ that are multiples of 100, ranging from 100 to 1000. The snapshot of a concept $C$ is represented by the bias and weight values of the perceptron. Hence, $C = [w_1, w_2, b]$. DACC's adaptive ensemble has the same settings as in Section 5.1

We show, in Figure 5.4, for each $\tau_{drift}$, the snapshots returned by DACCv1. We also show in Figure 5.5 for each $\tau_{drift}$, the online classification performance of DACCv1. The online performance is reset each $\tau_{drift}$ time steps.

The best case is when $\tau_{drift} = 500$ because the snapshots coincide with the concept changes in Stream1. As previously seen in Section 5.1, DACC is able to learn each of the subsequent stable concepts of Stream1. In other words, DACC captures the relationship between the input features $\mathbf{x}$ and the classes $y$ after observing enough training data from the underlying target concept. Hence, at the time of the concept change, the parameters

$[w_1, w_2, b]$ of the best expert in the ensemble are the same as the parameters $[a_0, a_1, a_2]$ of the corresponding decision boundary in Stream1. This explains why the parameters of the concept snapshots in Figure 5.4 evolve exactly as the parameters of the decision boundary in Figure 5.2.

We notice in Figure 5.5 the poor classification performance of the first predicted concept $\tilde{C}_3$ when $\tau_{drift} = 500$. However, starting $\tilde{C}_4$, the performance is clearly improved with nearly 100% of correct classifications. This means that with a minimum of three snapshots, the Elman network learns how the parameters $[w_1, w_2, b]$ evolve with time and predicts perfectly the upcoming concepts.

When $\tau_{drift} = 1000$, the snapshots are taken each 1000 time steps. Since a concept change occurs each 500 time steps, we miss a concept change between two snapshots. This explains why DACCv1 misses the concepts with negative $w_2$ values in Figure 5.4. According to the snapshots, the concept remains the same. Therefore, the Elman network will always predict the same concept. Since the underlying target concept is different than the predicted one during the first half of the prediction period $\tau_{drift}$, the classification performance decreases. The performance improves during the second half when the target concept gets back to its previous parameters.

When $\tau_{drift} = 300$, the second snapshot is taken at time step 600. At that time, the ensemble is adapting to the second concept. Thus, the second snapshot does not represent a stable concept but rather a *transition* between two concepts. Consequently, the list of snapshots $\mathcal{M}_{LT}$ is disturbed by outliers and the Elman network is not able to predict properly the upcoming concept snapshots. This is also the case for the other $\tau_{drift}$ values.

#### 5.1.1.4 Drawbacks

The main problems of DACCv1 can be summarized as follows:

- Depending on the preset parameter $\tau_{drift}$, we get different evolutions in the concept parameters. This gives place to unpredictable behaviors.
- Once the Elman network predicts a concept $\tilde{C}$, the predicted concept is used to classify the incoming instances and this, regardless of the classification performance of $\tilde{C}$, and regardless of whether DACC's committee could perform better than $\tilde{C}$.
- Fixing the parameter $\tau_{drift}$ assumes that the change happens periodically, a condition that is not necessarily satisfied in a real world scenario.

In the next section, we suggest a more sophisticated method to handle the mentionned problems.

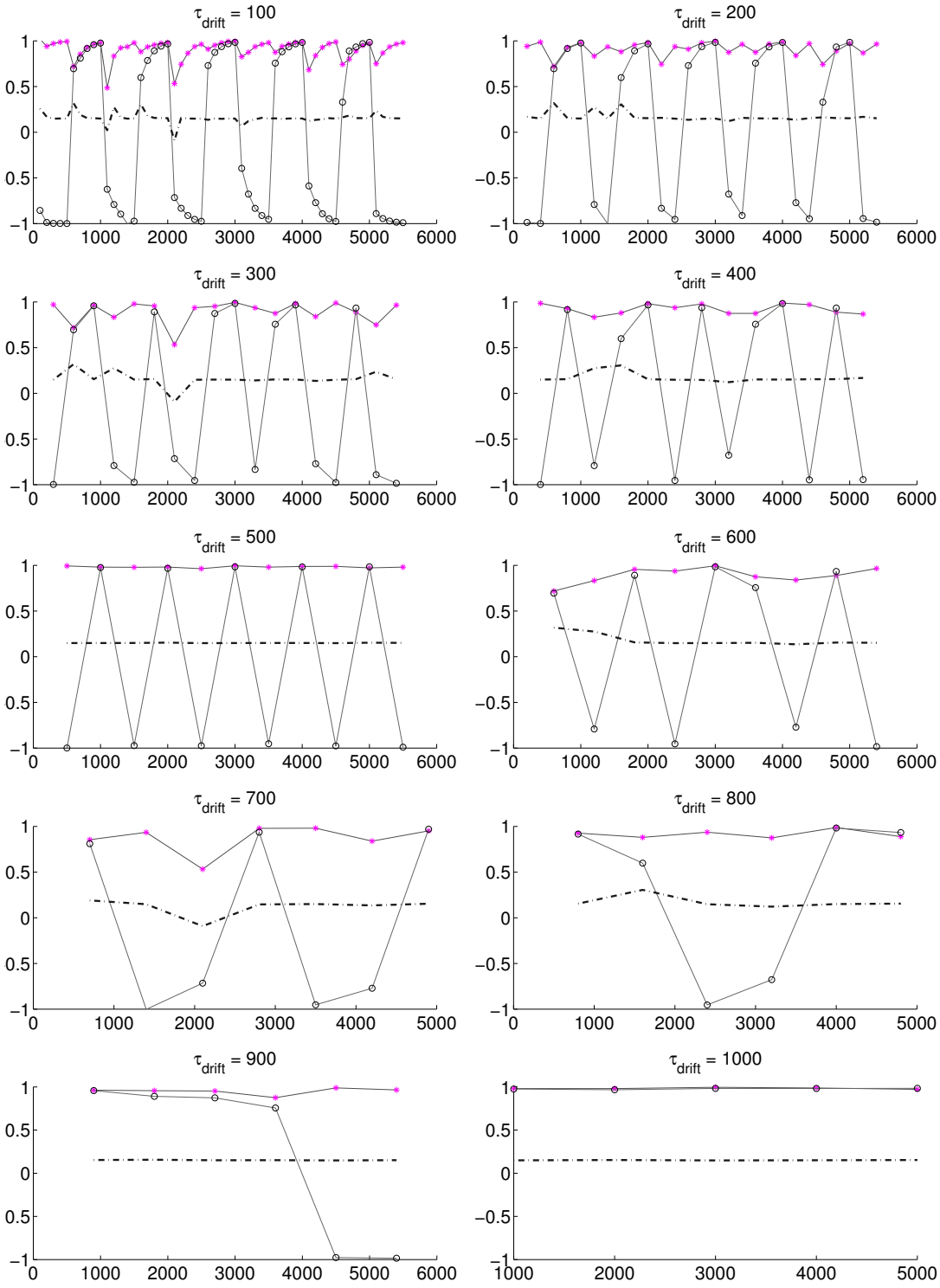


FIGURE 5.4: The snapshots of the concepts as returned by DACCv1 on Stream1, using different $\tau_{drift}$ time steps, the period separating two consecutive snapshots.

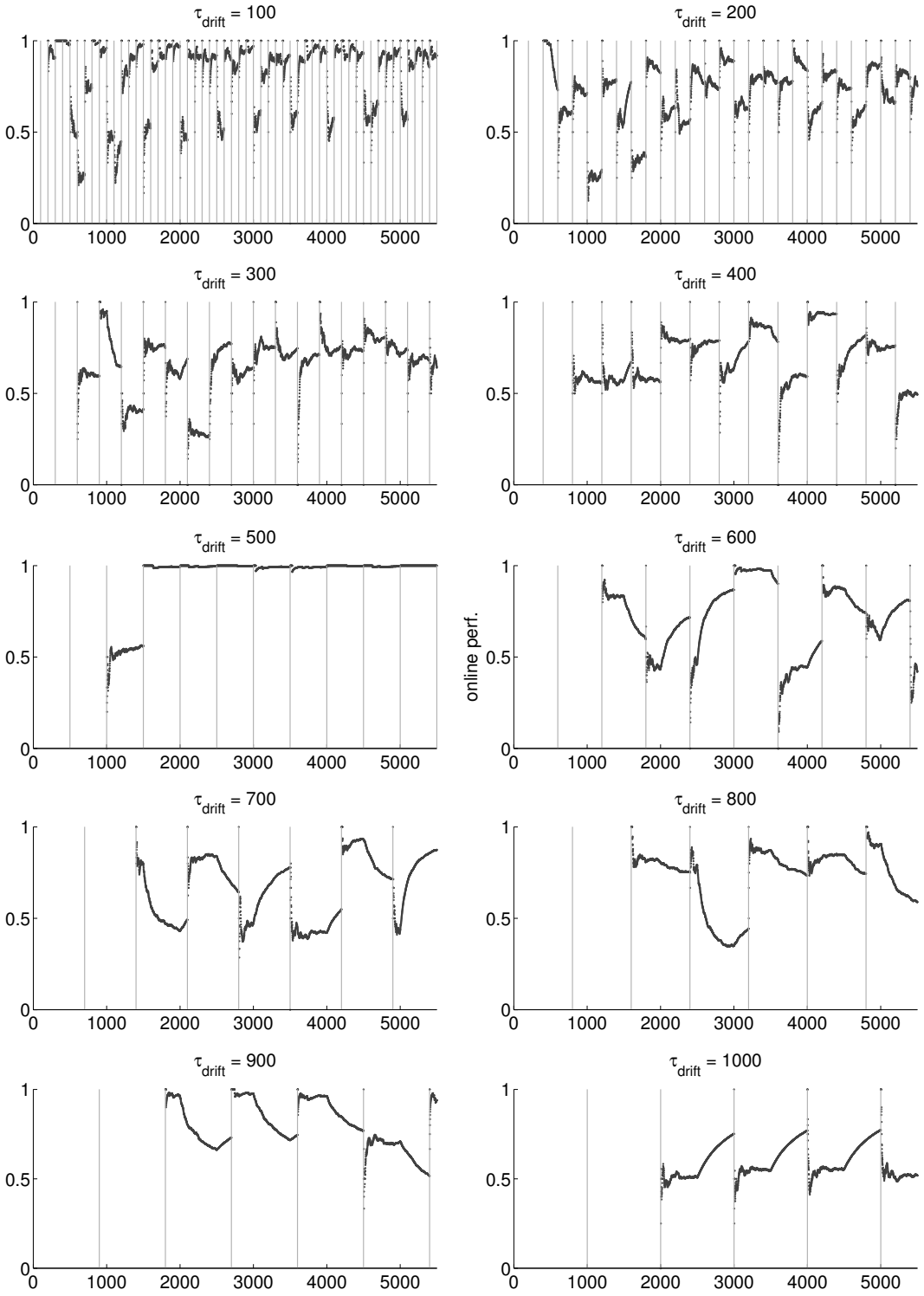


FIGURE 5.5: The online classification performance of DACCv1's predicted snapshots on Stream1, using different $\tau_{drift}$ time steps, the period separating two consecutive snapshots.

### 5.1.2 DACCv2

Analyzing how the underlying target concept evolves with time implies that the past history of stable concepts be captured. Rather than making prior assumptions about the time at which a concept changes as in DACCv1, we aim at finding an implicit way to identify periods of stability via the adaptive ensemble. When a period of stability is identified, we would take a snapshot (a copy) of the underlying stable target concept via the base learners in the ensemble. The snapshots are used to build the history of the stable concepts, providing means to analyze their evolution, and therefore predict the more likely future concept. Recognizing the stable concepts raises the following questions:

- How can we recognize whether the adaptive ensemble has "learnt" the current stable concept or not yet? Equivalently, how can we ensure that a snapshot returned by the adaptive ensemble reflects the underlying stable concept?
- DACC doesn't explicitly detect a concept change. How can we ensure that a concept change occured between two consecutive snapshots? Equivalently, how can we avoid storing snapshots of the same concept, twice in a row?

In the following, we first describe a mechanism for the recognition of the relevant states of the world before presenting DACCv2, an enhanced version of DACC that couples the adaptive ensemble with a the second-order learning mechanism that works on the identified states of the world in order to predict future concepts.

#### 5.1.2.1 Recognition of significant models of the world

In this step, we aim at recognizing the significant concepts encountered during the data streaming. Our goal is to take a snapshot of any recognized stable concept via the base learners in the adaptive ensemble. Note that we suppose therefore that the concept drift operates via a sequence of stable concepts, either in a gradual or in an abrupt manner (see Figure 2.5).

A main challenge here is to decide when to take a snapshot. Given our reliance on an ensemble method, this is equivalent to decide when the ensemble reflects the existence of an underlying relevant concept. Unlike other systems [3, 99], our method does not recognize new concepts using a drift detection system and thus it avoids the dfficulties inherent in having to detect gradual concept changes while still being robust to false alarms. Rather than making prior assumptions about the dynamics of the world, DACCv2 finds an implicit way to identify periods of stability using the adaptive ensemble as a detector.

In the adaptive ensemble, the base learners are constantly evaluated on their recent classification performance and each $\tau_{mat}$ time steps, base learners are deleted amongst the worst half of the ensemble. Deleting base learners allows the system to forget what it has learnt and therefore, makes it able to adapt to potential concept changes. As explained in Section 4.3.8, the deletion strategy causes the following behavior:

When the environment has been in a stable state for a sufficient time, the top learners (belonging to the best half of the ensemble), being exempt from deletion, are inputted with more training examples from the current concept. After observing enough training examples, they tend to give the same predictions on the instances labels. This translates into low levels of diversity among the best half of the ensemble.

When the concept changes, the former best learners see their performance deteriorate and are therefore replaced with new base learners, which increases the diversity level in the ensemble. It then takes some time until new best learners learn the new concept and their diversity decreases again.

Hence, in periods of stability, the best learners in the ensemble should converge toward the same, and near optimal, predictive error. Therefore, their diversity should be low, while their error rate should decrease toward the best achievable performance or close to it. This suggests to take into account both the diversity of the best learners in the ensemble and their error rate as an index of the stability of the environment. In our study, we use the kappa statistics $\kappa$ in order to compute the diversity. As previously explained in Section 4.3.8, this statistics measure evaluates the degree of agreement between the classification of a set of items by two classifiers. In case of complete agreement, $\kappa = 1$. If there is no agreement other than what would be expected by chance, $\kappa = 0$.

At time step $t$, given an ensemble of $N$ base learners (or hypotheses) $\left\{h_t^i\right\}_{1 \leq i \leq N}$, the stability index is computed over the last $\tau_s$ received examples as follows:

$$I_{stability} = agreement - error$$

where *agreement* and *error* are computed over the best half of the current learners in the ensemble, and are defined as:

$$agreement = \frac{\sum_{i=1}^{N/2} \sum_{\substack{j=1 \\ i \neq j}}^{N/2} \mathcal{K}_{h_t^i, h_t^j}}{\frac{N}{2} * (\frac{N}{2} - 1)} \tag{5.1}$$

and:

$$error \;=\; \frac{\sum_{j=0}^{\tau_s-1}\sum_{i=1}^{N/2} err\big(h_t^i(\mathbf{x}_{t-j}), y_{t-j}\big)}{\tau_s * \frac{N}{2}} \tag{5.2}$$

where $(\mathbf{x}_t, y_t)$ is the received example at time step $t$; $\mathcal{K}_{h_t^i,h_t^j}$ is the kappa for hypotheses $h_t^i$ and $h_t^j$; $h_t^i(\mathbf{x})$ is the prediction of $h_t^i$ regarding the class label of $\mathbf{x}$ and $err$ is an error function which returns 1 if the prediction is wrong and 0 otherwise. Hence,

$$err\big(a,b\big) = \begin{cases} 0 & \text{if } a = b \\ 1 & \text{if } a \neq b \end{cases} \tag{5.3}$$

Each point in the stability index curve, over some predefined threshold $\theta_I$, is suggestive of a stable environment and is therefore a privileged moment to take a *snapshot* (a copy) of the current best hypothesis in the ensemble of base learners, the one that seems to best represent the current state of the world.

One must however be careful not to store consecutive hypotheses that correspond to the same underlying state of the environment. Here again, the agreement statistics, in our case the kappa statistics, can be used to measure the agreement between a candidate snapshot $h_t^*$ and the preceding one $h_{t_k}^*$. For this purpose, the predictions of $h_t^*$ are then compared with the predictions of the last stored snapshot $h_{t_k}^*$ on the last $\tau_s$ received examples. If the estimated agreement is less than some predefined threshold $\theta_d$, the current candidate snapshot $h_t^*$ is considered different enough from $h_{t_k}^*$ and is therefore added to the list $\mathcal{M}_{LT}$ of snapshots representing past stationary states of the environment.

#### 5.1.2.2 The second order learning

As with DACCv1, the ordered list of past snapshots $\mathcal{M}_{LT} = \{C_1, C_2, \ldots, C_k\}$, is the basis of the second order learning mechanism. In our experiments, we also used Elman's recurrent neural networks to predict future concepts. After recognizing $k$ stable concepts, the $k-1$ change samples $\{\delta_1, ..., \delta_{k-1}\}$ form the set of training examples learnt by the Elman network in order to predict the next likely snapshot $\tilde{C}_{k+1}$ (see Section 5.1.1.2).

#### 5.1.2.3 The ensemble's final prediction

When predicting the label of an instance from the data stream, the predicted snapshot is treated on an equal footing with the base learners in the ensemble of the adaptive ensemble method. The snapshot is therefore evaluated according to the evaluation strategy used by the adaptive ensemble to evaluate the base learners. For instance, if the

combination function used by the adaptive ensemble is the MAX function, a snapshot is used for prediction if its evaluation record (weight) is the best among all candidate hypotheses from the ensemble of base learners. Unlike the base learners in the ensemble, the predicted snapshot is not modified or deleted. Its role is only to participates more or less in the ensemble's final prediction depending on its recent classification performance.

The steps of the second-order learning can be summarized as follows. Every $p$ time steps, the current stability index is computed. Then,

1. if the stability index is larger than $\theta_I$, a *candidate* snapshot $C_{cand}$ is taken.
2. if the difference between a candidate snapshot $C_{cand}$ and the last snapshot $C_{k-1}$ in $\mathcal{M}_{LT}$ is larger than $\theta_d$, the new snapshot $C_k = C_{cand}$ is added to $\mathcal{M}_{LT}$.
3. if a new snapshot $C_k$ has been added to $\mathcal{M}_{LT}$, an Elman network is trained on the change samples $\{\delta_1 = (C_1, C_2), \delta_2 = (C_2, C_3), \ldots, \delta_{k-1} = (C_{k-1}, C_k)\}$.
4. by running the Elman network on $C_k$, the network returns its prediction of the next concept $\tilde{C}_{k+1}$. The last predicted concept replaces any previously predicted concept.

DACCv2's method builds incrementally the list of stable concepts encountered during the data streaming. Each time a new stable concept is identified, the next concept is predicted (see Figure 5.7). The predicted concept is then constantly evaluated for classification purposes. As soon as it is assessed to be useful, the predicted concept participates in the data stream classification process.

#### 5.1.2.4 Empirical results

**Stream1** We evaluated DACCv2 on Stream1. The anticipative meta-learning system involves three parameters that all pertain to the detection of relevant snapshots. They are the *stability threshold* $\theta_I$, the *decision threshold* $\theta_d$ and the *duration for the evaluation* of candidate snapshots $\tau_s$. They were set respectively to $\theta_I = 0.9$, $\theta_d = 0.8$ and $\tau_s = 100$.

According to $\theta_I$, candidate snapshots are taken into account when the stability index is at least 0.9. We didn't choose a higher value fearing that higher stability indexes might not be reached. According to $\theta_d$, a candidate snapshot is considered different than the last snapshot stored in memory if their agreement measure is less than 0.8. Both the stability index and the concept equivalence measures are assessed on the last $\tau_s = 100$ predictions. The parameter $p$ (controlling the rate at which the anticipation mechanism is called) is used to reduce the additional computational cost incurred by the meta-learning system. It is set to $p = \tau_s$ in all our experiments.

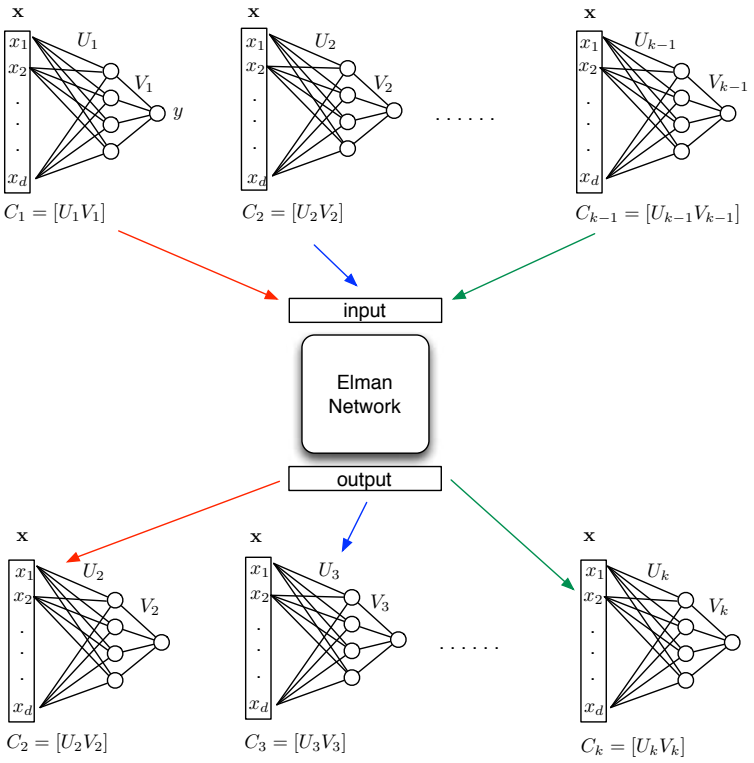


FIGURE 5.6: We show the training process of the Elman network in DACCv2 when using feed-forward neural networks as base learners in the adaptive ensemble. Accordingly, a snapshot is a feed-forward neural network represented by a vector containing its weight values. The pairs of consecutive snapshots are presented as training examples to the Elman network.

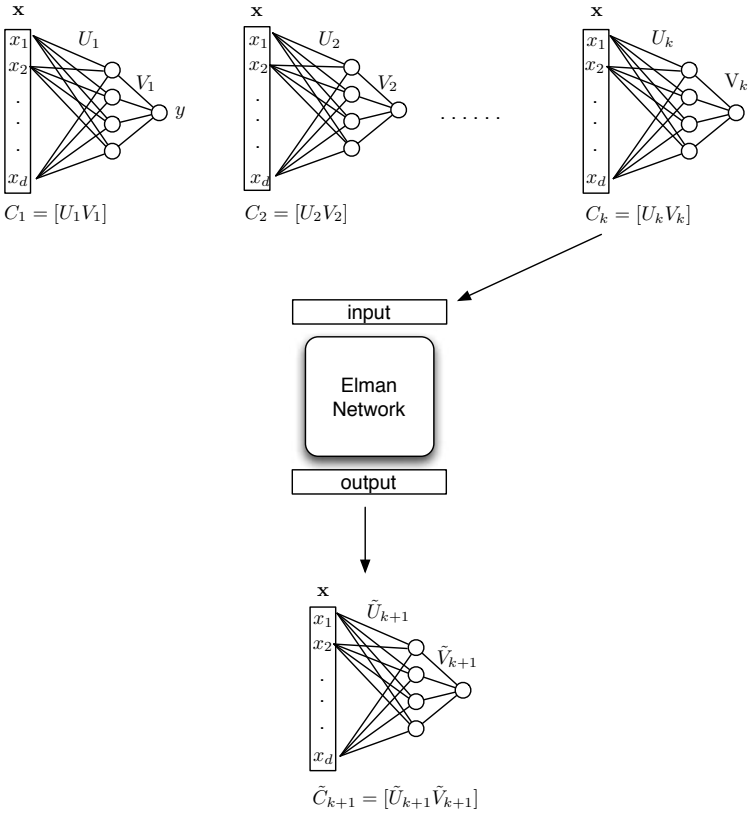


FIGURE 5.7: We show the prediction of the next snapshot using the Elman network in DACCv2 when using feed-forward neural networks as base learners in the adaptive ensemble. Accordingly, a snapshot is a feed-forward neural network represented by a vector containing its weight values. By running the Elman network on $C_k$, the network returns its prediction of the next concept (snapshot) $\tilde{C}_{k+1}$.

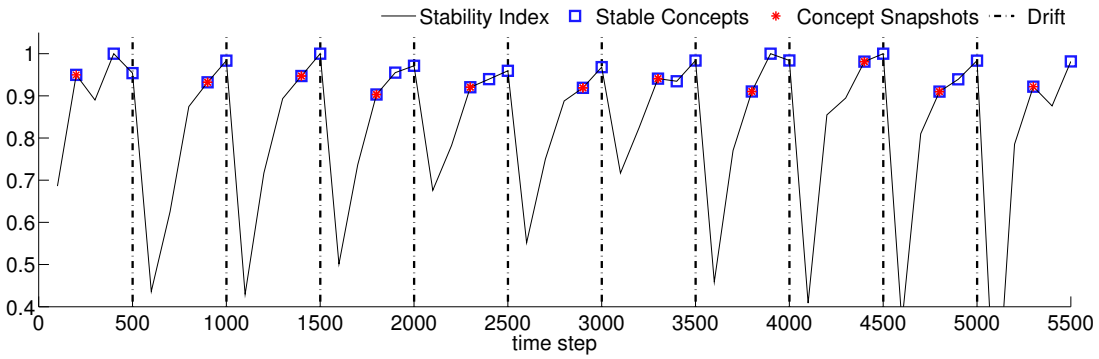


FIGURE 5.8: The time steps at which snaphots of concepts are taken by DACCv2 on Stream1.

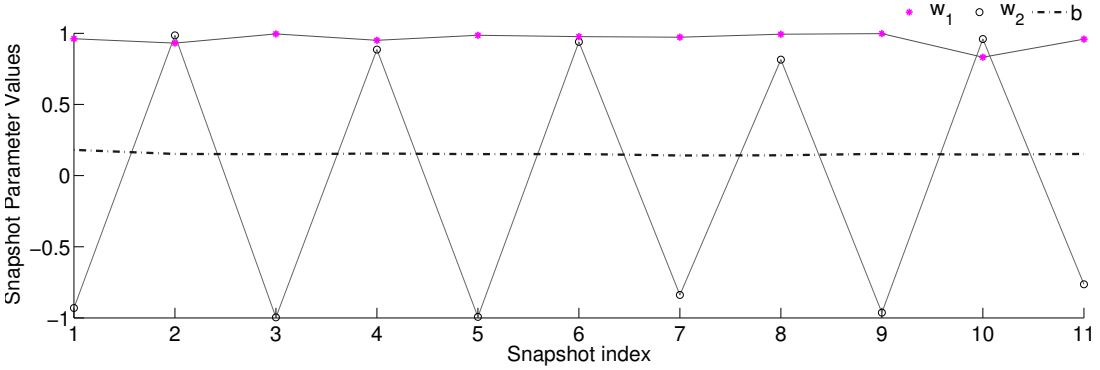


FIGURE 5.9: The snapshots of the most stable concepts of Stream1, as returned by DACCv2.

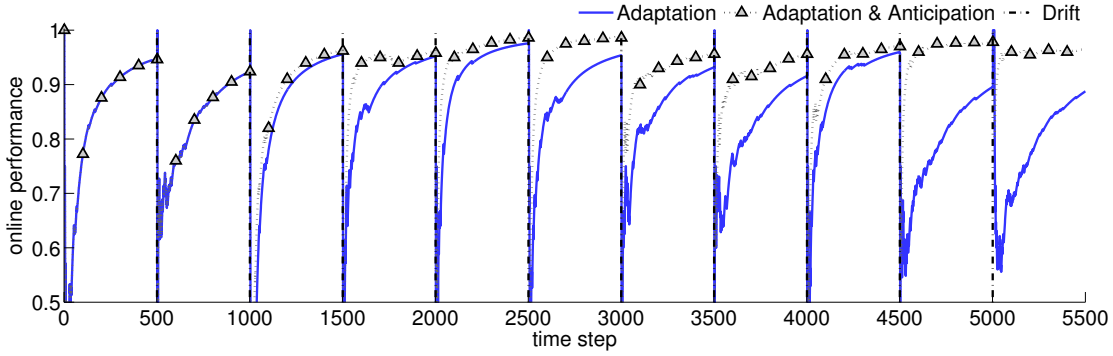


FIGURE 5.10: The online classification performance of DACC and DACCv2 on Stream1. The online performance is reset each 500 timesteps, with each concept change. The vertical bars represent the time steps at which concept changes occur.

Finally, we represent a concept snapshot by the vector of the bias and weight values $C = [w_1, w_2, b]$ of the current best base learner in the adaptive ensemble (the perceptron with the highest weight). The adaptive ensemble has the same settings as in Section 5.1.

We show in Figure 5.8 how the stability index evolves with time. Blue squares represent time steps at which candidate snapshots are taken into consideration. Candidate snapshots with red asterisks are kept as concept snapshots. Hence, two consecutive red

asterisks represent snapshots that, according to $\theta_d$, are different enough to represent different concepts. The concept snapshots are the components of the list $\mathcal{M}_{LT}$ at the end of the experiment. They are shown in Figure 5.9.

We can see how the concept snapshots in Figure 5.9 mirror the evolution of the concept's parameters in Stream1, as shown in Figure 5.2. Ideally, there would be one snapshot exactly for each encountered state during the data stream, which is the case in this experiment. By imposing the stability index to be relatively large according to $\theta_I$, we omit taking snapshots when the ensemble is still adapting to a concept change, being in a transition state between consecutive concepts, for instance at time step 600.

We show in Figure 5.10 the online classification performance of DACC and DACCv2 on the instances of Stream1. The online performance is reset each 500 time steps with each concept change. We notice that by anticipating concept changes, the classification performance is either improved or in the worst case, it is the same as the performance of the adaptive ensemble in DACC.

From Figure 5.8, we see that at time step 900, two concept snapshots are retained, and the list $\mathcal{M}_{LT}$ contains its first pair of snapshots $\mathcal{M}_{LT} = \{C_1, C_2\}$. At that point, the Elman network is trained on the first example $\{\delta_1 = (C_1, C_2)\}$ and predicts the concept $\tilde{C}_3$. According to Figure 5.10, the presence of $\tilde{C}_3$ doesn't bring a significant improvement when the third concept appears from time step 1000 to 1500. This is probably because the first predicted concept $\tilde{C}_3$ is different from the real target concept. As a result, the classification record (weight) of the predicted concept will not be high enough to allow it to be used for classification. The classification performance during this period is mostly this of the adaptive ensemble.

At time step 1400, the third snapshot $C_3$ is retained and the second change sample $\delta_2 = (C_2, C_3)$ is added to the training examples of the Elman network. The fourth concept is now predicted.

We notice that, from time step 1500 and on, the online performance is clearly improved by the presence of the predicted concepts. In fact, when concept changes occur, the classification performance of the adaptive ensemble starts decreasing while this of the predicted concept starts increasing. Thus, after a concept change, the weight of the predicted concept will be significantly higher than this of the adaptive learners, allowing it to classify the incoming instances from the stream on the behalf of the ensemble. The predicted concept will take the lead in the classification process until the base learners in the adaptive ensemble adapt to the new concept and their weight increase over again.

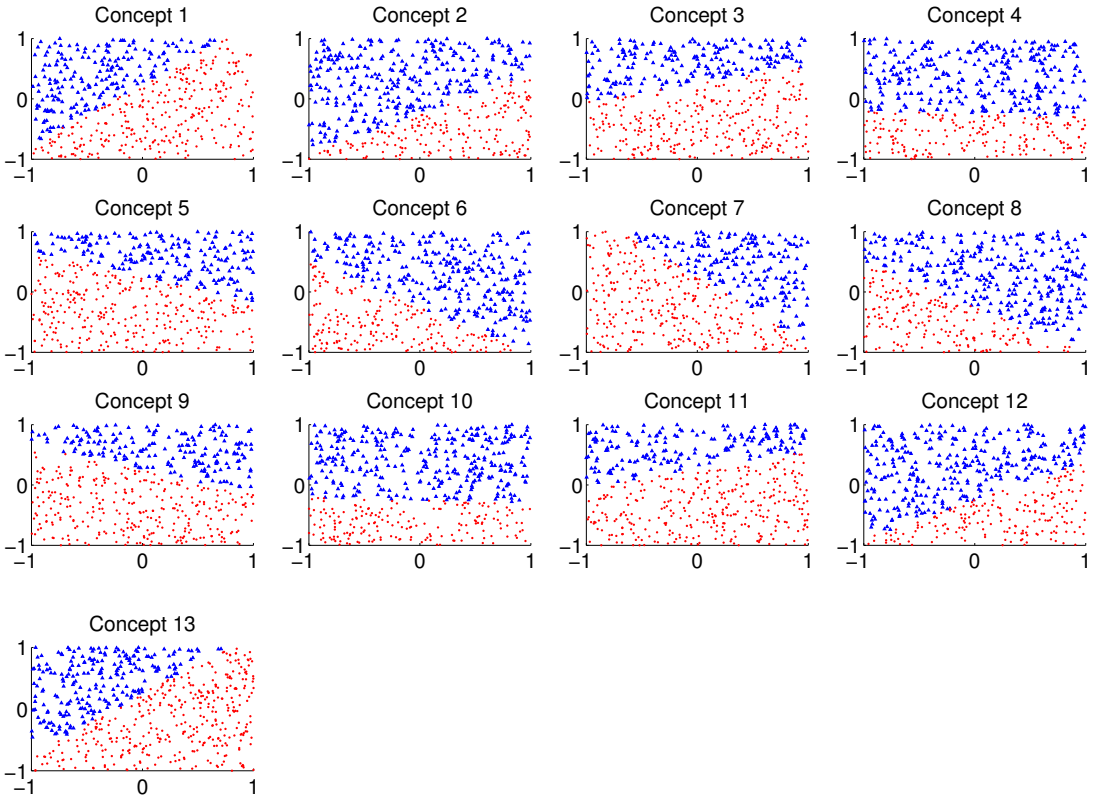


FIGURE 5.11: The data of the 13 consecutive concepts of Stream2. The data are separated into 2 classes depending on the parameters of the current concept.

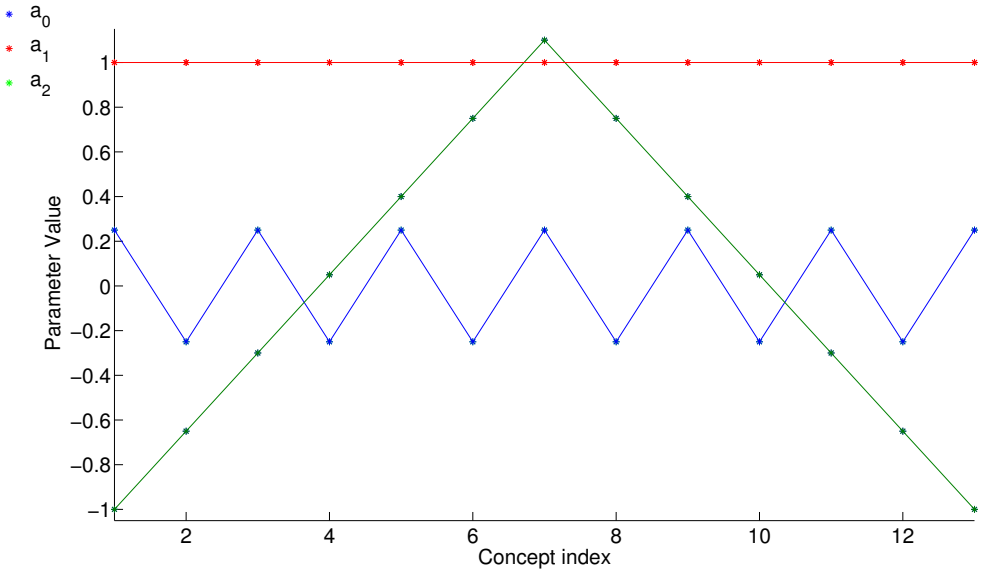


FIGURE 5.12: The parameters of the evolving concepts of Stream2.

**Stream2** We evaluated DACCv2 and DACC on another sequence of concept changes, using the same settings as with Stream1. The target concept is a linear decision boundary as in Stream1 but with a different evolution of the concept parameters. We generated a sequence of 13 concepts with 500 training examples each, as shown in Figure 5.11.

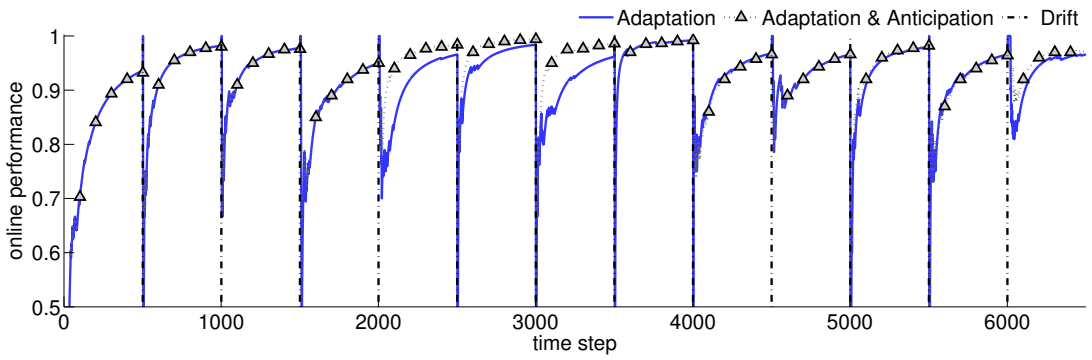


FIGURE 5.13: The online classification performance of DACC and DACCv2 on Stream2. The online performance is reset each 500 timesteps, with each concept change. The vertical bars represent the time steps at which concept changes occur.

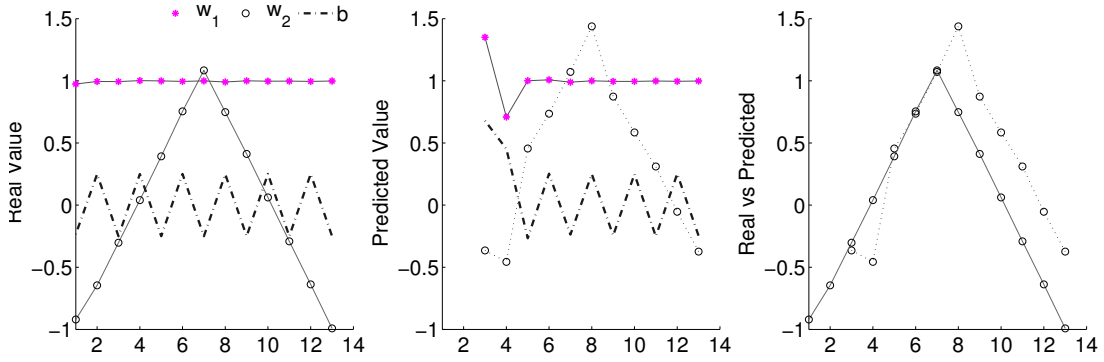


FIGURE 5.14: Left: the snapshots of the stable concepts as stored in the list $\mathcal{M}_{LT}$ by DACCv2. Middle: the predicted concepts by the Elman network. Right: a juxtaposition of both the snapshots and the predicted concepts.

The parameters of the concept evolve as shown in Figure 5.12. The new stream of data will be referred to as Stream2. The online classification performance of DACC and DACCv2 on Stream2 is shown in Figure 5.13.

We notice that anticipating future concepts improves the online performance from time steps 2000 to 3500 but the improvement disappears afterwards. To analyze the cause, we show in Figure 5.14 the snapshots of the stable concepts $\mathcal{M}_{LT} = \{C_1, C_2, \ldots, C_{13}\}$ returned by DACCv2 and the predicted concepts based on $\mathcal{M}_{LT}$, that is $\{\tilde{C}_3, \tilde{C}_4, ..., \tilde{C}_{13}\}$.

According to Figure 5.14, the list of snapshots reflects correctly how the concept evolves with time. The main issue however is that the Elman network is not able to predict the value $w_2$ (the second parameter) of the concept snapshot when $w_2$ starts decreasing. In fact, during the first half of the experiment, the Elman network, having observed the constant increase in the value of $w_2$, learns the simple evolution rule. After observing a change in the evolution of $w_2$, that is an increase followed by a decrease, the Elman network fails to predict the behavior of $w_2$.

#### 5.1.2.5 Drawbacks

We believe that anticipation can be improved by using a prediction model more sophisticated than the Elman network. If the Elman network hadn't previously observed the increase of $w_2$, he would probably have predicted the decrease in $w_2$ without difficulties. A question is then whether it is possible for the prediction system to forget the old evolution trend (the increase) and take only into consideration the new evolution trend, that is, the decrease in the parameter values.

The adaptive strategy in DACC was designed with the ability to forget previous knowledge and to adapt to *concept drifts*. By analogy, can we use DACC's algorithm to adapt to *concept change drifts*? In the following, we explain how a slighty modified version of DACC's adaptive strategy can be used to *adapt to local variations in the evolution of the concept parameters*.

### 5.1.3 DACCv3

We present DACCv3, an enhanced version of DACCv2. Rather than using one predictor (e.g. Elman network) to predict the parameters of the evolving concept, DACCv3 uses an ensemble of predictors, managed by DACC's adaptive strategy. This allows outdated predictors to be replaced with new ones, more adapted to track the current evolution of the concept parameters. It is important to note that DACCv3 comprises two different ensembles:

- The adaptive ensemble used to adapt to concept changes. This ensemble is trained on the examples received from the data stream.
- The adaptive ensemble used to predict upcoming concepts. This ensemble is trained on the concept change samples provided by the second order meta-learning. This ensemble is the subject of the following discussion.

#### 5.1.3.1 Ensemble of Predictors

The aim of the new ensemble here is to predict, based on the $k$ snapshots stored in $\mathcal{M}_{LT} : \{C_1, C_2, \ldots, C_k\}$, the next concept snapshot $C_{k+1}$.

A snapshot $C_i$ is represented by a vector of $n$ parameters:

$$C_i = [c_{(i,1)}, c_{(i,2)}, \ldots, c_{(i,n)}]$$

A predicted snapshot $\tilde{C}_i$ is represented by the following vector of parameters:

$$\tilde{C}_i = [\tilde{c}_{(i,1)}, \tilde{c}_{(i,2)}, \ldots, \tilde{c}_{(i,n)}]$$

Ideally, the predicted snapshot $\tilde{C}_i$ would mirror the still unacquired snapshot $C_i$.

As with DACCv2, the training examples provided to the ensemble of predictors are concept change samples, pairs of consecutive snapshots from $\mathcal{M}_{LT}$. When the $k$-th snapshot is acquired, the total history of concept changes is the sequence:

$$\big(\delta_1 = (C_1, C_2), \delta_2 = (C_2, C_3), \ldots, \delta_{k-1} = (C_{k-1}, C_k)\big)$$

**Building the ensemble** The well known stability-plasticity trade-off arises in this prediction scenario. If the *concept evolution* is stable, analyzing a long history of concept changes gives a better prediction accuracy. However, if the evolution trend changes suddenly, outdated history samples might be misleading, impeding the adaptation to the new evolution trend. This dilemma is handled implicitly by using the ensemble of predictors, each of which trained on a different history size.

The adaptive ensemble starts as in the classical version of DACC, by defining the *pool of types of predictors*, a set of predictors with different structures. The predictors can be neural networks with different numbers of hidden neurons and activation functions [43]. We can also have different learning models such as neural networks [43], decision trees [17], linear regression models [73], Bayes rules [95], support vector machines (SVMs) [45] and others.

The ensemble $P$ is initially empty. When the first concept change sample $\delta_1$ is observed, a pool of types of predictors $pl_1$ trained on $\delta_1$ is added to the ensemble. When the second change sample $\delta_2$ is observed, each predictor in $pl_1$ adds $\delta_2$ to its history and is retrained. In addition, a new pool of types of predictors $pl_2$ trained on $\delta_2$ only is added to the ensemble. This process continues until we reach the maximum number of pools of predictors $N_{pool}$. At this point, the pools of predictors $\{pl_1, pl_2, \ldots, pl_{N_{pool}}\}$ have a history of size: $\{N_{pool}, N_{pool} - 1, \ldots, 1\}$, respectively.

Now that the ensemble $P$ is formed, it is updated every $\tau_{mat}$ time steps according to DACC's deletion strategy, where a time step here corresponds to a new concept change sample. The deletion strategy selects randomly $m$ predictors from the worst half of the ensemble and replaces them with a new pool of predictors. To keep the ensemble size fixed, $m$ is equal to the size of the pool of predictors.

The predictors are evaluated for deletion on their recent predictive performance and weights are assigned accordingly. Just as in DACC, the weight of a predictor is inversely proportional to its prediction error on the last $\tau_{eval}$ predictions. Therefore, the higher the weight, the better the predictor. Unlike in classification tasks, predictions here are continuous values. Therefore the quality of a prediction is not assessed depending on whether its value is correct or uncorrect. The prediction error is measured here as the mean square error between the predicted snapshots and the real snapshots in $\mathcal{M}_{LT}$.

**Ensemble's prediction** Each predictor $p$ gives its prediction of the next concept snapshot $\tilde{C}^p_{k+1}$:

$$\tilde{C}^p_{k+1} = [\tilde{c}^p_{(k+1,1)}, ..., \tilde{c}^p_{(k+1,n)}] \tag{5.4}$$

where $\tilde{c}^p_{(k+1,i)}$ is the *i-th* parameter of the concept $C_{k+1}$ as predicted by the *p-th* predictor.

Unlike with classical DACC, the final prediction is formed by selecting the best prediction for each concept parameter $c_{(k+1,i)}$ independently. This is motivated by the fact that selecting the whole predictions $\tilde{C}^p_{k+1} = [\tilde{c}^p_{(k+1,1)}, ..., \tilde{c}^p_{(k+1,n)}]$ of a predictor $p$ assumes that all the parameters $c_{(k+1,i)}$ change their evolution trend at the same time. However, if a predictor $p$ is trained on a long history size then its prediction might be suitable for a parameter $c_{(k+1,g)}$ that evolves according to a stable evolution function but not for another parameter $c_{(k+1,f)}$ that just changed its evolution trend. Thus, we decide to select a predictor for each parameter of the concept. Accordingly, the final prediction of the ensemble is represented as follows:

$$\tilde{C}_{k+1} = [\tilde{c}_{(k+1,1)}, ..., \tilde{c}_{(k+1,n)}] \tag{5.5}$$

where $\forall i \in \{1, .., n\}$

$$\tilde{c}_{(k+1,i)} = \arg\min_{p \in P} \parallel c_{(k+1,i)} - \tilde{c}^p_{(k+1,i)} \parallel \tag{5.6}$$

Selecting the best predictions would be trivial if $c_{(k+1,i)}$ was known $\forall i \in \{1, .., n\}$ (see equation 5.6), but this would also make the prediction task unnecessary. We choose the best predictions by defining an evaluation function that assigns a weight $w^p_i$ to each predictor in the ensemble, and for each snapshot parameter $i$. The weight reflects the performance of a predictor $p$ in predicting the parameter $i$ of the concept snapshot, and this based on its last $\tau_{eval}$ predictions. The equation 5.6 becomes:

$$\tilde{c}_{(k+1,i)} = \tilde{c}^{p^*}_{(k+1,i)}$$

where

$$p^* = \arg\max_{p \in P} \left( w_i^p \right)$$

Weights reflect the quality of predictions. They are inversely proportional to the predictive error. Accordingly, the weight $w_i^p$ at the time the concept $k+1$ is to be predicted, is computed as follows:

$$w_i^p(k+1) = \begin{cases} \tau_{eval} / \sum_{j=0}^{\tau_{eval}-1} \left( c_{(k-j,i)} - \tilde{c}^p_{(k-j,i)} \right)^2 & \text{if } age(p) \geq \tau_{mat} \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

#### 5.1.3.2 Empirical results

In a first set of experiments, we simulated different types of evolutions in the concept parameters. We then evaluated the predictive accuracy of the ensemble using different types of predictors, more specifically, Elman neural networks [98] and polynomial regression models [35]. We showed that by mixing different types of predictors in the ensemble, we take advantage of each type of predictor and get better prediction results. We also showed that our predictors, whose history size change dynamically with time, outperform predictors trained on a fixed size window of concept change samples.

In a last experiment, we tested DACCv3 on Stream2 in order to evaluate whether DACCv3 overcomes the unability of DACCv2 to follow the evolution trend of the concept parameters.

**Experiment (1)**

In this set of experiments, we focused on the ability of the adaptive ensemble to adapt to different evolutions in the concept parameters. Therefore, the evaluation process did not involve DACCv3 algorithm in its wholeness. We generated the list $\mathcal{M}_{LT}$ ourselves, simulating a series of concepts changes.

We simulated an evolving concept with 5 parameters, moving continuously as shown in Figure 5.15. The first 4 parameters were initially set to random values. The parameter values evolved with time according to simple evolution rules that may change suddenly its evolution tendency. As for the last parameter, its value was set to the sum of the remaining parameters, multiplied by 0.5. This kind of evolution mirrors the evolution of a concept represented by a perceptron, with the first 5 parameters being the perceptron's input weights and the last parameter the bias. The bias value is set such that nearly

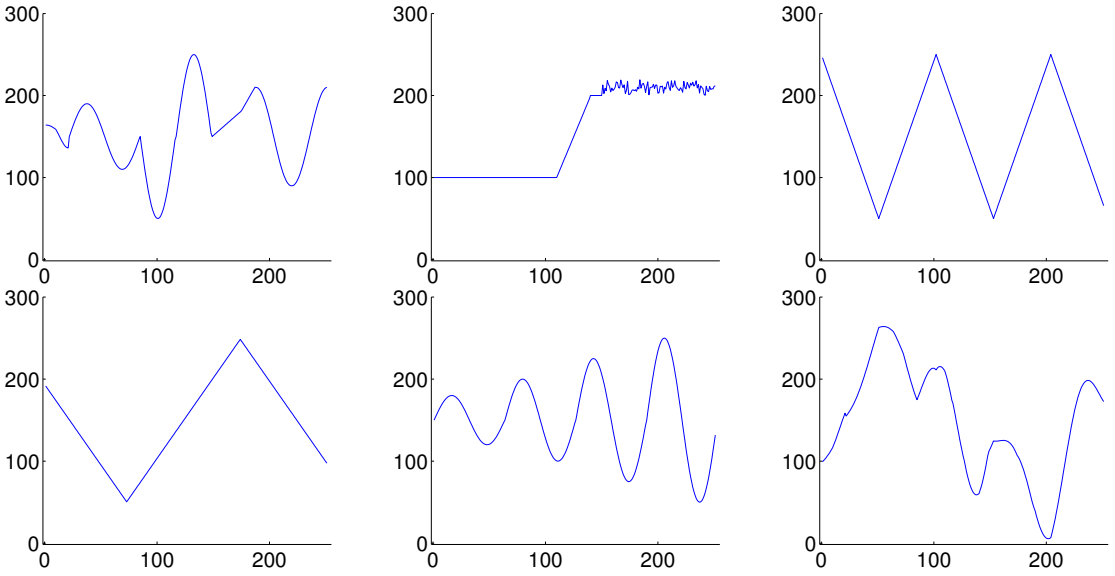


FIGURE 5.15: The evolution of the six parameters of the concept described in experiment 5.1.3.2.

half of the examples are assigned to class '0' and the other half to class '1'. A total of 250 different concepts were simulated as shown in Figure 5.15.

### Neural Networks vs Polynomial Regression

We tested the predictive ensemble using different types of predictors. We were specially interested in comparing Elman neural networks [98] and polynomial regression models [35] as predictors.

We started by testing neural network predictors. We chose a pool of type of predictors that consists of an Elman neural network. The Elman network had an input layer of size 6 (the number of parameters to be inputted to the network), a linear hidden layer with one neuron and a linear output layer with 6 neurons (the number of parameters to be predicted by the network).

We conducted three experiments where we varied the maximum number of predictors in the ensemble. In each experiment, we compared ourselves to a simple prediction scenario which considers the next concept snapshot the same as the current one. The prediction results are reported in exp. 1, 2 and 3 of Table 5.1.

We notice that, using Elman neural networks, the predictive ensemble beats the simple prediction approach in nearly 71.21% of the time. For instance in exp. 2, our prediction error is 3.28 times smaller than the error of the simple prediction scenario in 74.29% of

TABLE 5.1: The prediction results with different predictor types and ensemble sizes, using the ensemble of predictors. *Exp* is the index of the experience. 1 EL stands for one Elman neural network and 3 PR stands for three polynomial regression models with degree 1,2 and 3 respectively. During the experiments, we predict the parameters of 250 concepts. For each prediction, we compute the prediction MSE: the mean square error between the predicted values and the real values. The *S_ b_ O MSE* is the percentage of time our prediction MSE is smaller than the simple prediction MSE. The *S_ b_ O MSE ratio* is the ratio between the simple prediction MSE and our prediction MSE, when our prediction MSE is smaller than the simple prediction MSE. The *S_ O MSE ratio* is the ratio between the simple prediction MSE and our prediction MSE.

| Exp. | Pool of type of predictors | $N_{Pool}$ | S_b_O MSE Per. (%) | S_b_O MSE ratio | S_O MSE ratio |
|---|---|---|---|---|---|
| 1 | 1 EL | 8 | 65.06 | 3.19 | 0.6 |
| 2 | 1 EL | 13 | 74.29 | 3.28 | 0.85 |
| 3 | 1 EL | 20 | 74.29 | 3.26 | 0.79 |
| 4 | 3 PR | 8 | 90.76 | 2.11, 1.97 | 1.77 |
| 5 | 3 PR | 13 | 91.96 | 2.17, 2.04 | 1.83 |
| 6 | 3 PR | 20 | 91.96 | 2.1, 1.98 | 1.8 |
| 7 | 1 EL, 3 PR | 8 | 84.34 | 3.45 | 1.03 |
| 8 | 1 EL, 3 PR | 13 | 82.73 | 3.66 | 1.08 |
| 9 | 1 EL, 3 PR | 20 | 81.53 | 3.71 | 1.12 |

the time[1]. However, the *S_ O MSE ratio* is below 1 in exp. 1, 2 and 3, meaning that our prediction error is larger than the simple prediction error, in average.

Summing up: when using Elman neural networks as predictors, our predictive ensemble gives better results than the simple prediction scenario most of the time. However, our prediction error is bigger than simple prediction error, in average.

In the second set of experiments, we tested the predictive ensemble using polynomial regression models instead of Elman networks as predictors. We repeated the previous tests using a pool of type of predictors that consists of 3 polynomial predictors with degree 1, 2 and 3 respectively. The results are reported in exp. 4, 5 and 6 of Table 5.1. We notice that when we use polynomial regression, the *S_ b_ O MSE Per.* and the *S_ O MSE ratio* improve. However, the *S_ b_ O MSE ratio* decreases. We can see for example that the *S_ b_ O MSE ratio* goes from 3.26 with neural networks in exp. 3 to 2.1 with polynomial regression in exp. 6.

We sum up our remarks with the following: on the one hand, Elman neural networks beat polynomial regression models by having a smaller prediction error when they are better than the simple prediction scenario. On the other hand, polynomial regression models beat the neural networks by having a smaller prediction error on average. So, the question is: how would be the prediction results when we mix both types of predictors?

[1]We refer to the *S_ b_ O MSE ratio*.

TABLE 5.2: The prediction results using predictors with a fixed size history. *Exp* is the index of the experiment. *Predictor* is the type of predictor used in the experiment; 1 EL stands for one Elman neural network. *History Size* is the fixed history size of the predictor. The last three columns are explained in Table 5.1.

| Exp | Predictor | History Size | S_b_O Per. (%) | S_b_O MSE ratio | S_O MSE ratio |
|---|---|---|---|---|---|
| 1 | 1 EL | 3 | 82.32% | 4.4 | 0.21 |
| 2 | 1 EL | 5 | 87.95% | 3.9 | 1.7 |
| 3 | 1 EL | 9 | 70.28% | 2.44 | 1.07 |
| 4 | 1 EL | 15 | 17.67% | 1.4 | 0.29 |
| 5 | 1 EL | *growing* | 4.47% | 1.2 | 0.0108 |

In the third set of experiments, we mixed both type of predictors: the pool of type of predictors contains an Elman neural network and 3 polynomial regression models with degree 1, 2 and 3 respectively. The prediction results are reported in exp. 7, 8 and 9 of Table 5.1. We notice that, by mixing neural networks with polynomial regression models, we take advantage of both types of predictors: the *S_O MSE ratio* and the *S_b_O MSE Per.* increase compared to when we only used neural networks while the *S_b_O MSE ratio* increases compared to when we only used polynomial regression models.

These experiments show that the predictive ensemble allows us to take advantage of different types of prediction models. Thus, we don't need to have a priori knowledge of the more suitable type of predictors for the anticipation task: the predictive approach will do the work for us by automatically removing the worst predictors and replace them by a new pool of types of predictors.

### Dynamic History Size vs Fixed History Size

We repeated the former experiments using predictors trained on a fixed history size of concept change samples. We conducted five experiments using Elman networks as predictors. In the first four experiments, the history size is set to 2, 4, 8 and 15 respectively. In the fifth experiment, the history size of the predictor grows with time. The results are reported in Table 5.2.

We notice that the prediction results depend on the history size we choose. When the history size is set to 5, the prediction error brings an improvement factor of 3.9 over the simple prediction scenario in 87.95% of the time, and an improvement factor of 1.7 over the simple prediction scenario, in average. Choosing a smaller window with size 3 decreases the *S_b_O Per.* and deteriorates the *S_O MSE ratio*. Choosing a larger window size (9, 15) also deteriorates the prediction results. The last predictor with a growing window size gives the worst results. Therefore, collecting all the observed samples is not good for prediction: this assumes that the change is a static function, a condition that is rarely met in the real world.

The results suggest that using fixed window size predictors requires an a priori knowledge of the suitable window size for the prediction task; choosing the wrong window size might give catastrophic results. It also requires to choose a specific type of predictor (neural network, polynomial regression, SVM etc...) which cannot allow one to take advantage of multiple types of predictors as in our method.

Finally, when comparing the results of the fixed size history predictors with the results of our prediction approach in exp. 7, 8 and 9 of Table 5.1, we notice that whatever the size of the committee, the prediction results are close to the best prediction result we can get in the fixed size prediction scenario.

**Experiment (2)**

It was shown in Section 5.1.2.4 that DACCv2 fails to anticipate the evolution of the concept parameters when evaluated on Stream2. To test our claims that DACCv3 overcomes the limitations of DACCv2, we evaluated DACCv3 on Stream2 using an ensemble of Elman networks to predict the parameters of the evolving concept. The ensemble has the following settings:

- The ensemble holds a maximum of 10 Elman networks i.e. $N = N_{pool} = 10$.
- The Elman networks are evaluated on their last 3 predictions of the parameter values $[w_1, w_2, b]$ i.e. $\tau_{eval} = 3$.
- The deletion strategy removes an Elman network from the ensemble every 3 time steps i.e. $\tau_{mat} = 3$.

The remaining anticipation settings are the same used by DACCv2 on Stream2. Therefore, $p = \tau_s = 100$, $\theta_d = 0.8$ and $\theta_I = 0.9$

We show in Figure 5.16 the online classification performances of both DACC and DACCv3. The snapshots of the stable concepts and the predicted concepts are shown in Figure 5.17.

From Figure 5.17, we notice that, using the committee of Elman networks, we can adapt to the sudden change in the evolution of the concept's parameters which occurs at snapshot 8. This translates into accurate predictions of the concept's parameters for snapshots 11, 12 and 13. As a result, the classification performance is improved from time step 5000 and till the end of the experiment (see Figure 5.16).

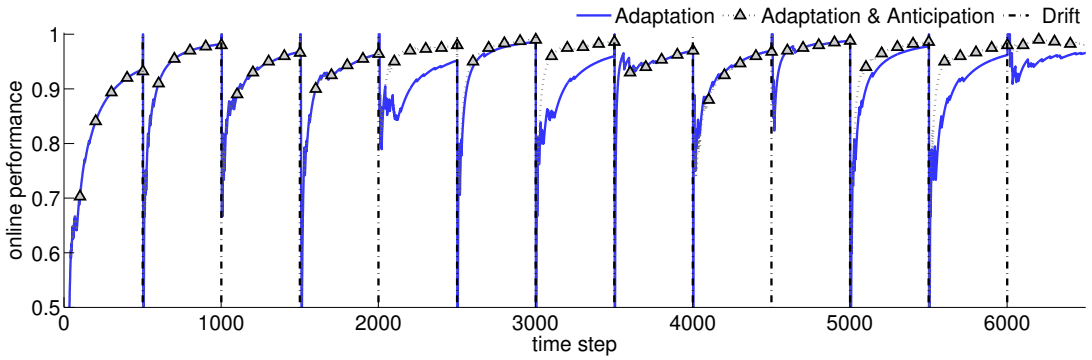


FIGURE 5.16: The online classification performance of DACC and DACCv3 on Stream2. A committee of Elman networks is used in DACCv3 to predict upcoming concepts. The online performance is reset each 500 timesteps, with each concept change. The vertical bars represent the time steps at which concept changes occur.

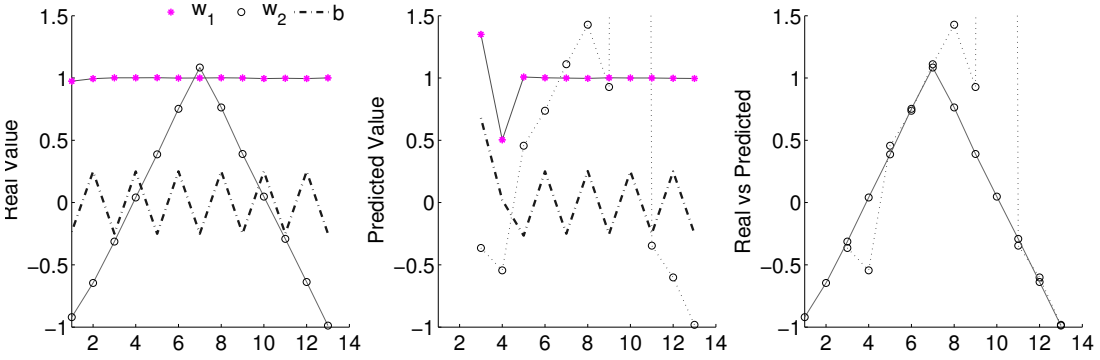


FIGURE 5.17: Left: the snapshots of the stable concepts as stored in the list $\mathcal{M}_{LT}$ by DACCv3. Middle: the predicted concepts by the committee of Elman networks. Right: a juxtaposition of both the snapshots and the predicted concepts.

## 5.2 Concept Reccurence

In Section 5.1, we worked on *predictability*, the ability to predict future concepts when a transformation rule can be identified. In this section, we work on *recurrence*, the ability to recognize recurring concepts.

In many real world scenarios, a concept might reappear over time. This problem introduces a new challenge in that apparently obsolete data or learned concepts may well reveal themselves as relevant again in the future. It would thus be profitable to be able to exploit this past knowledge as soon as it is appropriate rather than learn anew from the incoming data stream. In recent years, several proposals have been put forward to meet this challenge, particularly within the ensemble-based approach (see for instance [3, 37, 41, 52, 99]). They either work at the level of the examples themselves, by chunking them in some way, possibly organizing a hierarchy of chunks, or they work at the level of the learned concepts themselves, trying to learn significant concepts in the stream of examples while avoiding redundancy.

In the following, we show how DACCv3 can be improved to handle recurring concepts. The new version of our method, DACCv4, handles both *concept predictability* and *concept reccurence*.

### 5.2.1 DACCv4

The main idea of DACCv4 is to use the list of concept snapshots $\mathcal{M}_{LT}$ as a reference, so that if an old concept $C_{old} \in \mathcal{M}_{LT}$ reappears, $C_{old}$ can be used directly to classify the incoming instances. Thus, $\mathcal{M}_{LT}$ serves two purposes. First, it provides a sequence of successive models of the environment that can be used by a learning algorithm in order to predict the most likely future state in the series. Second, it stores a memory of past successful models of the world, models that should be repeatedly tested against current data in case a recurring concept can be recognized.

#### 5.2.1.1 The second order learning

After recognizing $k$ stable concepts, the $k-1$ change samples $\left(\delta_1, ..., \delta_{k-1}\right)$ form the set of training examples inputted to the concept prediction system (for instance, a single Elman network as in DACCv2 or an ensemble of Elman networks as in DACCv3) in order to predict the next snapshot $\tilde{C}_{k+1}$. The predicted snapshot is then temporally added to the list of snapshots $\mathcal{M}_{LT}$. It is replaced by the next snapshot $C_{k+1}$ when this one is acquired. Therefore, the list $\mathcal{M}_{LT}$ contains snapshots that represent past stationary states of the environment (useful for the recognition of recurring concepts), plus a predicted future state (useful for anticipation).

Each snapshot in the list is then treated on an equal footing with the base learners in the adaptive ensemble. The snapshots are therefore constantly evaluated according to the evaluation strategy used by DACC to evaluate the base learners in the adaptive ensemble. Hence, a snapshot in $\mathcal{M}_{LT}$ is used for prediction depending on its evaluation record (weight) and the combination function used by the adaptive ensemble. However, unlike the base learners in the adaptive ensemble, the snapshots in $\mathcal{M}_{LT}$ are not modified or deleted.

While the adaptive ensemble of base learners is of a finite constant size, the list $\mathcal{M}_{LT}$ of snapshots may a priori increase forever if new hypotheses are continually retained as worthy of storage. Fortunately, it is possible to keep this size under control by recognizing that the two roles of $\mathcal{M}_{LT}$: predictability and memory for recurring concepts, ask for two different memory management systems. Indeed, since Elman's networks are incremental learners, they do not need to keep the past history of snapshots at all. Regarding the

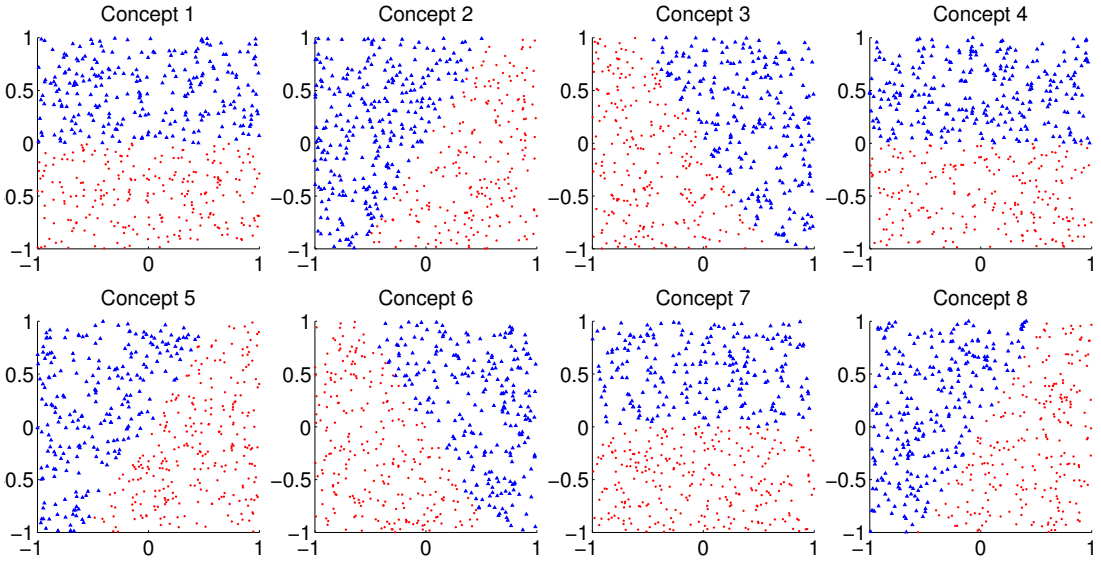


FIGURE 5.18: The data of the 8 consecutive concepts of Stream3. The data are separated into 2 classes depending on the parameters of the current concept.

memory for recurring concepts, it can be kept constant using various heuristics. One is to delete the oldest or the least reccuring snapshots from the memory. Another one, more sophisticated, would be to store prototype snapshots instead of the original ones, using a hierarchical clustering technique. In our experiments, we did not rely on such memory management schemes since there was no need for them.

#### 5.2.1.2 Empirical results

**Stream3** In order to evaluate DACCv4, we generated an artificial stream of data simulating reappearing concepts. We created a sequence of 8 concepts with 500 training examples each. The concept is a linear decision boundary that separates the examples into 2 different classes, as shown in Figure 5.18. The first 3 concepts are different from each others. They are then repeated from concept 4 until 8. This new stream of data will be referred to as Stream3.

We evaluated DACCv4 on Stream3, with the same settings used by DACCv3 on Stream2 (see Experiment 3 in Section 5.1.3.2). The time steps at which snapshots are taken by DACCv4 are shown in Figure 5.19.

According to Figure 5.19, at time step 1500, snapshots of the first 3 concepts have already been added to the list $\mathcal{M}_{LT}$. Since the concepts reappear afterwards, we expect the classification performance to increase for the remainder of the streaming process. We show in Figure 5.20 the online classification performance of both DACC and DACCv4 on Stream3.

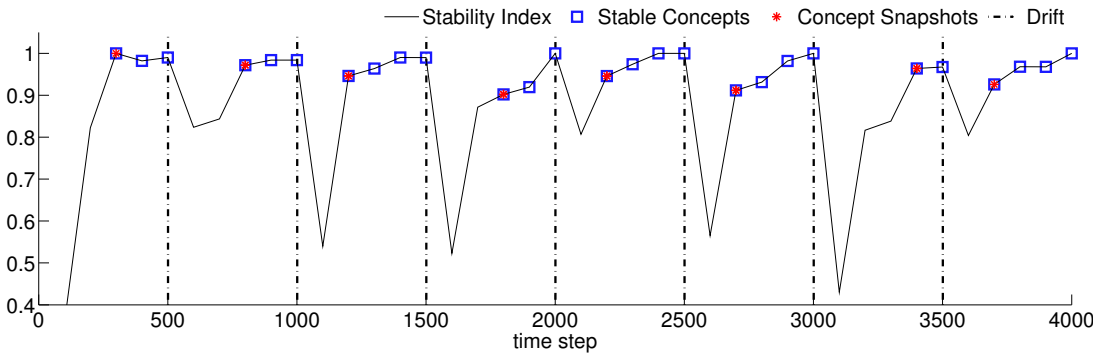


FIGURE 5.19: The time steps at which snaphots of concepts are taken by DACCv4 on Stream3.

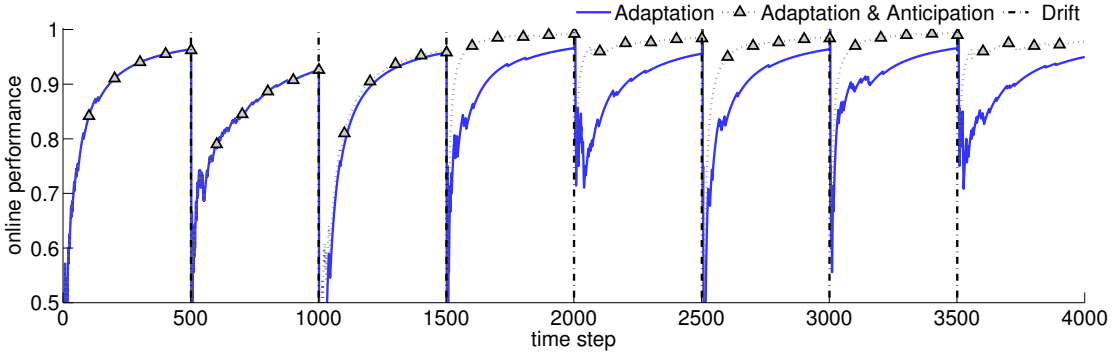


FIGURE 5.20: The online classification performance of DACC and DACCv4 on Stream3. The online performance is reset each 500 timesteps, with each concept change. The vertical bars represent the time steps at which concept changes occur.

We notice that, starting time step 1500, the classification performance is indeed improved by the presence of the snapshots of the previous concepts. In fact, when instances start being classified based on the 4th concept, the performance record of DACC's base learners starts decreasing while the performance of the corresponding snapshot starts increasing. In DACC's settings, the member with the highest predictive record on the recent $\tau_{eval}$ time steps is selected to predict the class of the incoming example. Therefore, when a concept reappears, we don't need to wait until DACC's base learners adapt to the concept to give correct predictions. The snapshot of the current concept, already present in $\mathcal{M}_{LT}$ and with a relatively high predictive record, takes the lead in classifying the incoming instances.

## 5.3 ADACC

DACCv4, the latest version of the meta-learning mechanism, couples DACC with an anticipation system that deals with both *predictability* and *recurrence*. We call this new ensemble algorithm ADACC for *Anticipative Dynamic Adaptation to Concept Changes*. ADACC deals with both the challenge of optimizing the stability-plasticity dilemma,

*adapting* to concept changes, and with the *anticipation* and *recognition* of incoming concepts. The steps and methods of ADACC are formalized in Algorithm 2 and an analysis of ADACC's computational complexity is presented in Section 5.3.1.

The experiments in Sections 5.1.3 and 5.2.1 were conducted to explain and evaluate the methodology used by ADACC to deal with predictable concept changes and reappearing contexts, respectively. The former experiments were rather simple, involving periodic and sudden concept changes only. In Sections 5.3.2 and 5.3.3, we aim at evaluating the behavior of ADACC on different types of concept drifts, with various speeds and severities, and regardless of whether drifts happen in a periodic or non-periodic way.

### 5.3.1 Computational complexity

The meta-learning mechanism introduced in ADACC increases the complexity of DACC's adaptive ensemble. According to Algorithm 2, the main operations of the meta-learning mechanism include:

1. Identifying $H$, the best half of experts in the adaptive ensemble (line 9)
2. Computing the average kappa value of the pairs of experts belonging to $H$. This value is computed over the last $\tau_s$ received examples (line 10)
3. Computing the average classification error of the former experts on the last $\tau_s$ received examples (line 11)
4. Computing the kappa value $\mathcal{K}_{C_k,h_t^*}$ of a candidate snapshot $h_t^*$ and the last stored snapshot $C_k$ in the list $\mathcal{M}_{LT}$ on the last $\tau_s$ received examples (line 19)

Step (1) has order of $\mathrm{O}(N \times \log N)$ complexity, as explained in Section 4.3.7. As for step (2), the kappa value $\mathcal{K}_{i,j}$ of classifiers $i$ and $j$ is the same as the kappa value $\mathcal{K}_{j,i}$ of classifiers $j$ and $i$. In addition, the kappa values $\mathcal{K}_{i,i}$ of pairs of identical classifiers are not computed since the agreement measure should be evaluated on different classifiers. Hence, a total of $0.5 \times (N/2) \times (N/2-1)$ kappa values are computed. Since kappa values are computed over the last $\tau_s$ classifications, the complexity of step (2) is $\mathrm{O}(0.5 \times (N/2) \times (N/2-1) \times \tau_s)$. Step (3) entails computing the classification error of the $N/2$ experts over the last $\tau_s$ classifications and hence has order of $\mathrm{O}(N/2 \times \tau_s)$ complexity. Finally, step (4) is executed only when a stable environment is identified. It adds a complexity of order $\mathrm{O}(\tau_s)$.

Overall, the complexity of the meta-learning mechanism is $\mathrm{O}(N \times \log N + 0.5 \times (N/2) \times (N/2-1) \times \tau_s + N/2 \times \tau_s + \tau_s) = \mathbf{O(}N^2\tau_s\mathbf{)}$.

**Algorithm 2:** Selection of snapshots by ADACC.

**input** : the stability threshold $\theta_I$
the concept equivalence threshold $\theta_d$
the evaluation window $\tau_s$
the period $p$ between two consecutive calls of the meta-learning system

**begin**
  $E_0 \leftarrow \emptyset$; /* Ensemble of experts */
  $\mathcal{M}_{LT} \leftarrow \emptyset$; /* List of snapshots */
  $k \leftarrow 1$;
  **for** $t = 1$ **to** $\infty$ **do**
    /* ---------- */
    /* Adaptation */
    /* ---------- */
    $(\mathbf{x}_t, y_t)$ is the current training instance;
    $[E_t, \tilde{y}_t] \leftarrow AdaptationEnsemble(E_{t-1}, \mathbf{x}_t, y_t)$;
    /* ---------- */
    /* Anticipation */
    /* ---------- */
    **if** $t \bmod p = 0$ **then**
      $H = \{h_t^i\}_{i=1}^{N/2}$ is the best half of experts in $E_t$;
      $$agr = \frac{1}{\frac{N}{2} * (\frac{N}{2} - 1)} \sum_{i=1}^{N/2} \sum_{\substack{j=1 \\ i \neq j}}^{N/2} \mathcal{K}_{h_t^i, h_t^j};$$
      $$error = \frac{1}{\tau_s * \frac{N}{2}} \sum_{j=0}^{\tau_s - 1} \sum_{i=1}^{N/2} err\big(h_t^i(\mathbf{x}_{t-j}), y_{t-j}\big);$$
      $I_{stability} = agr - error$;
      /* Detect Stable Concept */
      **if** $I_{stability} \geq \theta_I$ **then**
        $h_t^* = snapshot(E_t)$;
        /* Detect New Concept */
        **if** $isEmpty(\mathcal{M}_{LT})$ **then**
          $C_k = h_t^*$;
          $\mathcal{M}_{LT} = add(\mathcal{M}_{LT}, C_k)$;
        **end**
        **else if** $\mathcal{K}_{C_k, h_t^*} \leq \theta_d$ **then**
          $k = k + 1$;
          $C_k = h_t^*$;
          $\mathcal{M}_{LT} = replace(\mathcal{M}_{LT}, \tilde{C}_k, C_k)$;
          $\tilde{C}_{k+1} = predictNextConcept(\mathcal{M}_{LT})$;
          $\mathcal{M}_{LT} = add(\mathcal{M}_{LT}, \tilde{C}_{k+1})$;
        **end**
      **end**
    **end**
  **end**
**end**

### 5.3.2 Empirical results (1)

The aim of the experiments was threefold. First, to test the performance of *the snapshot mechanism* and specially its ability to detect both abrupt and gradual concept changes and store the relevant target concepts with no, or limited, redundancy. Secondly, to examine the gain, if any, brought by *the anticipation scheme* compared to the mere adaptation mechanism. Obviously, this depends on the ability to anticipate the next state of the environment, and therefore on the underlying structure (if any) of the sequence of changes [7]. Thirdly, to test the mechanism for *the recognition of recurring concepts* and the gain it can bring.

In the worst case, where it is not possible to anticipate the next concept and when no recurring concept arises, the prediction performance of the system should fall back to the performance of its adaptation mechanism. In these cases, no snapshot in $\mathcal{M}_{LT}$ does outperform the best base learners and the resulting behavior is the one of the adaptive system alone.

#### 5.3.2.1 Experiments and datasets

We conducted experiments on artificial and real datasets. The artificial sets were used to simulate *recurrent* and *predictable concept changes* while controlling the timing of the change, its speed (abrupt, or more or less gradual) and its severity (amount of change) [69]. The real dataset comes from video sequences taken with a mobile robot wandering in and out of rooms in a laboratory, creating *recurring contexts*.

In the *artificial datasets*, the input space $\mathcal{X}$ is $d$-dimensional and the target concept is a linear decision boundary (a hyperplane) described by the relation $y(\mathbf{x}) = sign(\sum_{i=1}^{d} w_i x_i + w_0)$. The experiments were carried out on streams with 7,150 time steps and hence data points.

In each stream, 12 concept changes were simulated by changing the weights $\{w_i\}_{i=0}^{d}$ of the hyperplane. The first 7 concepts evolved through the successive addition or substraction of constant values (differing according to the experiments) to the weights. The idea was to look at the capacity of the anticipative mechanism to identify this regularity and therefore to predict likely future concepts. The last 6 concepts were recurring concepts, that is concepts already encountered in the past data stream (see Figure 5.21).

Three artificial data streams were generated, each involved a different level of severity in its concept changes: *low* or *medium* or *high* respectively involving changes in 1, 5 and 9 parameters out of the 11 that define the target concept, with respectively approximately

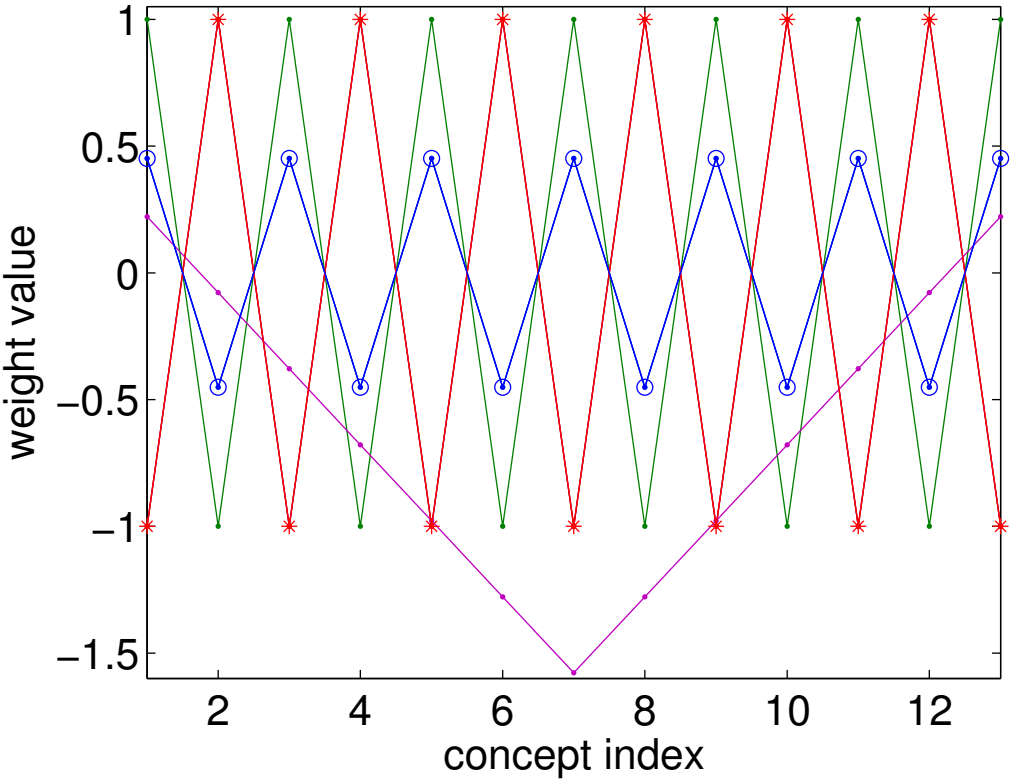


FIGURE 5.21: A typical evolution in the weight values of the hyperplane used in the artificial datasets.

3%, 60% and 84% of the input space changing class between successive concepts. In each stream, the changes happened either suddenly or gradually, in a linear manner, between successive target concepts.

The transition between consecutive concepts took from 0 to 200 time steps and changes would start happening every 400 to 700 time steps. We did not observe any effect of the dimension up to more than one hundred and therefore we only report results for the 10-dimensional case. The base learners in the adaptation ensemble were perceptrons with 10 input units and one output unit, involving 11 weights (10 + 1 for the bias). One Elman network was used to predict future snapshots (as in DACCv2). The Elman's networks took as input the 11 weights of a snapshot and gave as output the 11 weights of the next predicted snapshot.

The *real dataset* was issued from the COLD database of the Saarbrücken laboratory [79], a benchmark for vision-based localization systems. It contains sequences of images recorded by a mobile robot under different variations of illumination and weather: sunny, cloudy and night. We worked on the dataset captured in sunny conditions. The images were labeled into one of four classes: *corridor*, *one-person office*, *printer area* and *classroom*, and the total length of the data sequence was 753. The robot visited the rooms

in the following order: *corridor*, *bathroom*, *corridor*, *one-person office*, *corridor*, *printer* and *corridor*. It stayed in the same room between 45 and 284 time steps. Images were first pre-processed into a 128-dimensional space using the Self-Organizing Map described in [42]. In the experiments, we used decision trees (as implemented in Matlab) as base learners in the adaptation ensemble.

One important goal of the experiments was to compare the performances achieved with the combined anticipative and adaptive mechanism, with the ones of a purely adaptive mechanism.

The anticipative meta-learning system itself involves three parameters that all pertain to the detection of relevant snapshots. They are the *stability threshold* $\theta_I$, the *decision threshold* $\theta_d$ and the *duration for the evaluation* of candidate snapshots $\tau_s$. They were set respectively to $\theta_I = 0.9$ and $\theta_d = 0.8$ while $\tau_s = 100$ was chosen for artificial data streams and $\tau_s = 25$ for robotics in order to cope with a faster dynamics. In the base version with no sophisticated management of the snapshot list $\mathcal{M}_{LT}$, there are no additional parameters. The parameter $p$ (controlling the rate at which the anticipation mechanism is called) is used to reduce the additional computational cost incurred by the meta-learning system. It is set to $p = \tau_s$ in all datasets.

The remaining parameters concern the adaptive mechanism, and we tried to optimize these in order not to unfairly attribute gains to the anticipative process. The parameters for the ensemble method for *adaptive online learning* include the size of the ensemble $N$, the maturity age $\tau_{mat}$ and the evaluation size $\tau_{eval}$. After extensive experiments, they were set as follows.

The ensemble comprised $N = 10$ base learners for the artificial datasets ($N = 15$ for the robotics data). In order to be compared, base learners were evaluated on the most recent $\tau_{eval} = 20$ data points (time steps) ($\tau_{eval} = 15$ for the robot). The duration for maturity $\tau_{mat}$ was equally set to 20 time steps ($\tau_{mat} = 10$ for the robot).

#### 5.3.2.2 Evaluation measures and methodology

In the experiments, we evaluated *the snapshots* stored by the system with respect to the known target concepts. Ideally, there would be one snapshot exactly for each encountered state during the data stream. For instance, in Figure 5.22, candidate snapshots are indicated with small squares and the retained ones appear as red squares.

We also evaluated the *gain in prediction errors* resulting from the use of the anticipation mechanism over the use of the adaptation scheme alone. Likewise, we measured the gain (if any) due to the recognition of a recurring concept. The gain is simply the number of

TABLE 5.3: Summary of the experiments and the measured gains in prediction errors wrt. an adaptive only strategy.

| **Stream** | | **Adaptation** | | | **Anticipation** | **Total gain** | | **Dues to predictability** | | **Dues to recurrence** | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| name | size | base learner | mean error | std-dev | predictor | mean | std-dev | mean | std-dev | mean | std-dev |
| 10-D Low | 7,150 | perceptron | **107.2** | 7.7 | Elman net | **1.9** | 1.7 | 0.0 | 0.0 | 1.9 | 1.7 |
| 10-D Med. | 7,150 | perceptron | **784.4** | 32.2 | Elman net | **317.7** | 25.4 | 70.8 | 9.7 | 246.9 | 21.2 |
| 10-D High | 7,150 | perceptron | **937.4** | 54.4 | Elman net | **393.9** | 46.5 | 120 | 18.1 | 273.9 | 34.3 |
| Robot | 753 | decision tree | **43.0** | 2.6 | - | **9.0** | 1.9 | - | - | 9.0 | 1.9 |

errors of prediction that were avoided with respect to the use of the adaptive strategy only (see Table 5.3 below).

Finally, the graphs in Figure 5.23 report at each time step the current online predictive performance. One can then observe, for instance, that the gains due to the anticipation mechanism start to show after the fourth concept change for the stream with medium severity level (see Figure 5.23, the second picture down from the top).

#### 5.3.2.3 Empirical results

Table 5.3 sums up the experimental results, averaged over 10 experiments. The table shows the mean predictive error of the adaptive learning strategy, and the gain of using the anticipation mechanism, in both predictability and recurrence. The gain is measured as the difference between the number of prediction errors made by the adaptive ensemble and the number of prediction errors made by the anticipation mechanism. In the artificial data streams, we highlight the gain brought by the *predictability* of the first 7 concept changes, and the gain brought by the last 6 *recurring* concepts.

Figure 5.23 illustrates the mechanism for the selection of snapshots on one data stream and it shows the evolutions of the prediction performance over 10 repeated experiments according to the severity of the concept changes.

**Detection of concept changes and selection of snapshots**

As can be seen in Figure 5.23, the value of the stability index closely mirrors the concept changes. As soon as the appearance of a new concept is detected by the system and the corresponding candidate snapshot sufficiently differs from the previously stored ones, it is stored away in $\mathcal{M}_{LT}$. That policy enables the fast detection of novel target concept.

In our experiments, 100 % of all new concepts (260 altogether) that were introduced in the data streams with high and medium severity levels triggered the storage of a new snapshot. However, in the data stream with *low severity* level, since the consecutive concepts are quite similar, few snapshots were retained (see Figure 5.22). There was

no redundancy (no more than one snapshot per concept) in the artificial data streams. Some snapshot redundancy appeared for the robotic data (see Figure 5.22) because of the variation within each concept (e.g. in the bathroom the robot's camera points to different parts of the room which may induce several snapshots).

### Second order learning

For concept changes of *low severity* (Figure 5.23, third picture down from the top), the adaptive strategy is able to follow the variation of the environment as soon as enough candidate hypotheses are good enough, which happens at the end of the first concept (circa 400 time steps). Therefore, the anticipation strategy does not bring an advantage there. The situation is significantly altered, however, when the concept changes are of *medium* or *high severity*.

In our experiments, even though the concept changes occur at varying dates and with varying speed, the anticipation mechanism is able to predict relevant foreseeable target concepts that, in turn, are quickly recognized as the best for labeling the incoming examples. This brings significant gains in the online performance starting already after the 3rd change of concept for the data stream with *high severity* level, and the 4th change of concept for the data stream with *medium severity* level. The gain increases thereafter with each new concept change.

Table 5.3 shows that the gain in the number or labeling errors attains $317.7/784.4 = 40.5\%$ for concept changes of *medium severity*, and 42% in the case of *high severity*. These gains are impressive in face of a difficult learning task. It is unlikely that they could be obtained without a second order learning mechanism working over the adaptive one.

Table 5.3 distinguishes furthermore between the gain due to the predictability and the gain due to the fast recognition of a recurring concept. Predictability brings significant gain in the medium and high severity settings for the artificial datasets. In the case of the robotics data, the gain is totally due to the fast recognition of recurring concepts which outperforms the adaptation mechanism.

As expected, there is never a negative gain. As noted earlier, because the ensemble methods is based on a continual competition between base learners from the adaptive mechanism and base learners from the anticipative one, second order learning can never be detrimental to the overall prediction performance as compared to the adaptive only policy.

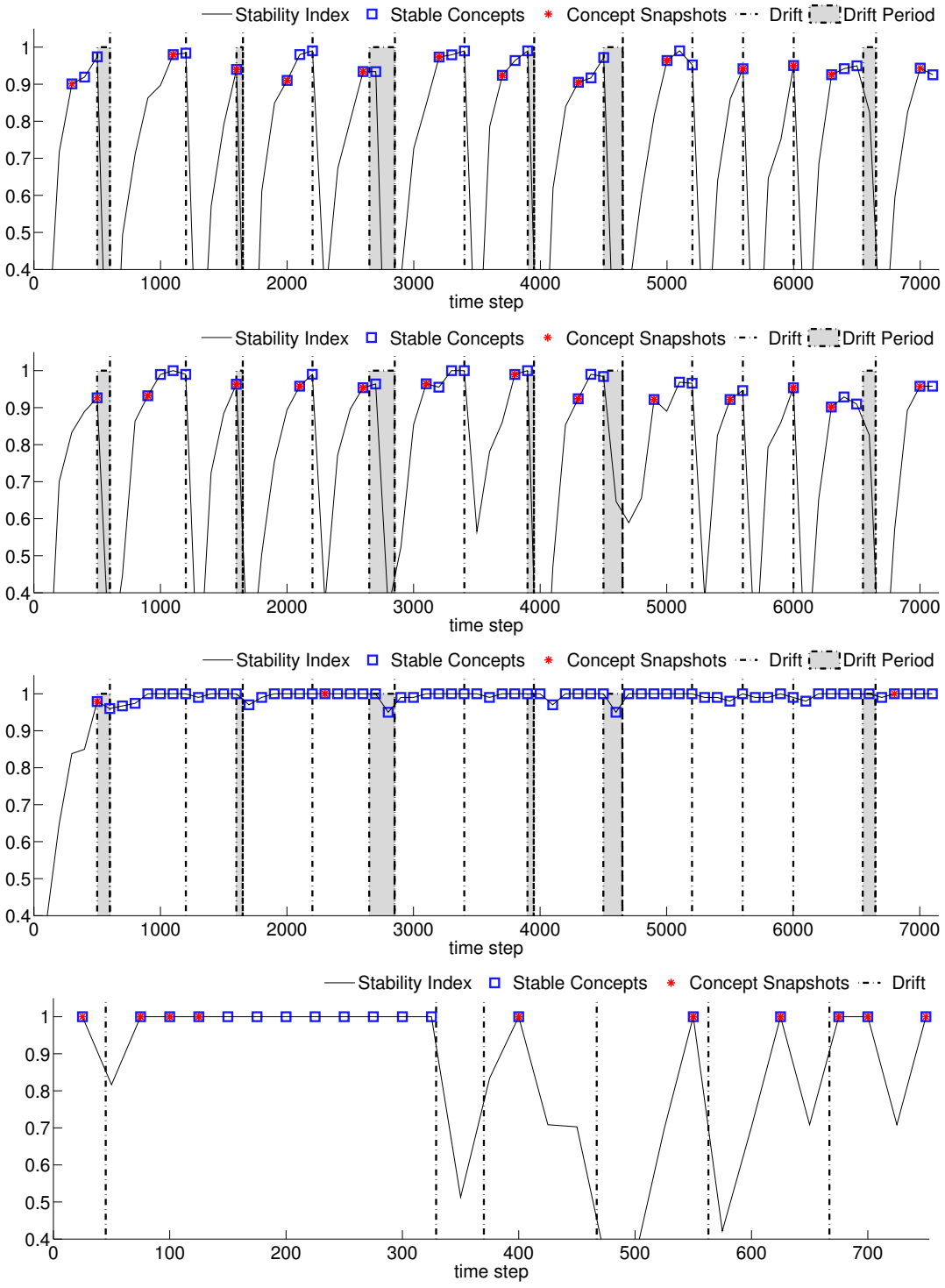


FIGURE 5.22: The evolution of the stability index for 10-dimensional artificial data streams (top three figures) and the robot data stream (last figure). We show the time steps where candidate snapshots are considered (small squares), and when they are retained (red squares). A candidate snapshot must have a stability index larger than $\theta_i$, and in order to be retained, it must differ from the last retained snapshot, according to a decision threshold $\theta_d$. First three plots: plot (*top*) corresponds to high severity concept change, plot (*middle*) to medium severity, and plot (*bottom*) to low severity.

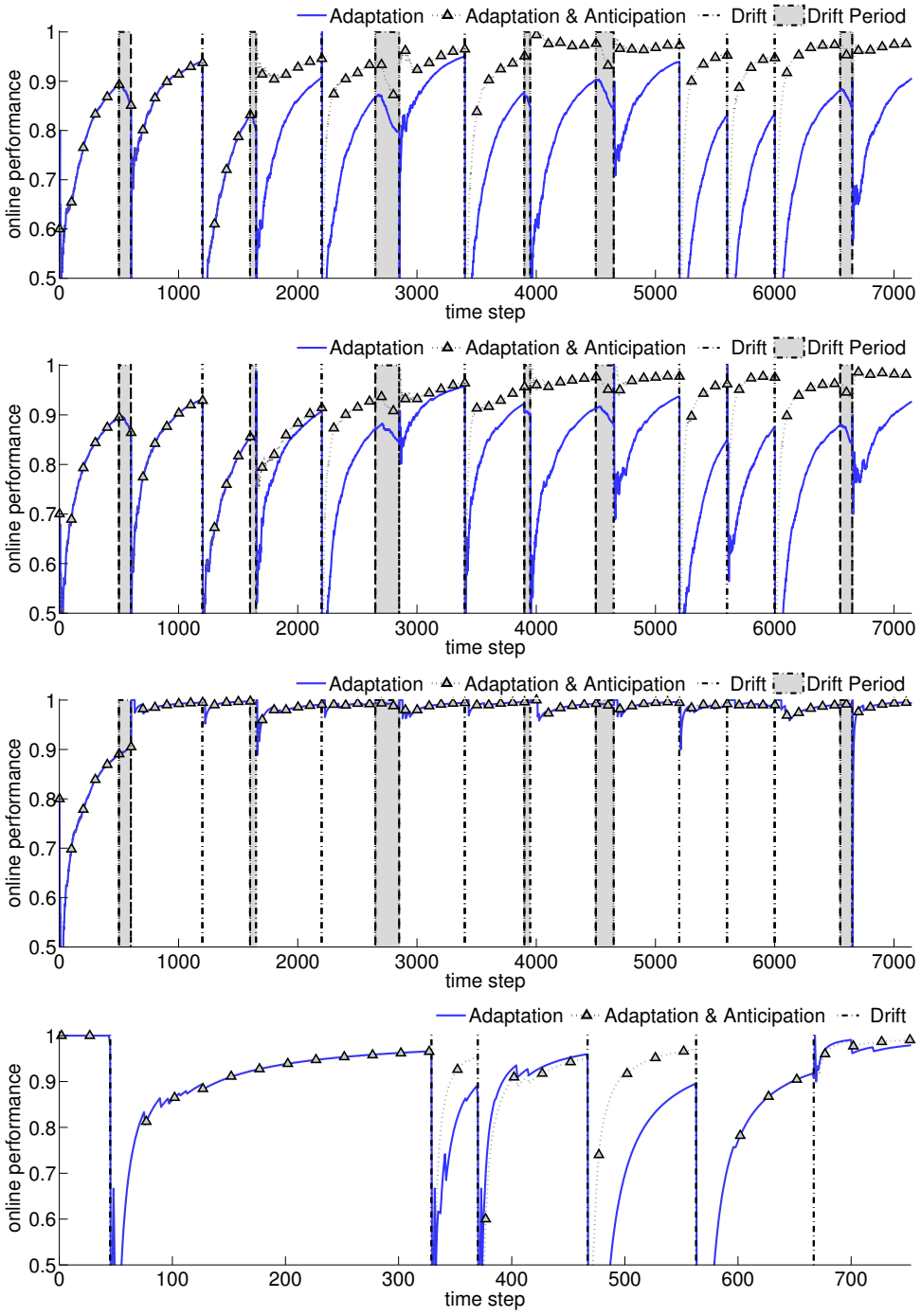


FIGURE 5.23: The online predictive performance for 10-dimensional artificial data streams (top three figures) and the robot data stream (last figure), using the adaptive learning strategy (continuous, blue, line) and with the second order learning taking place. The experiments are averaged over 10 repeated simulations. The beginning/end of concept changes are indicated as vertical dotted lines. In case of gradual concept changes, the transition period between consecutive concepts is colored in gray. The online predictive performance is reset once a transition is complete. First three plots: plot (*top*) corresponds to high severity concept changes, plot (*middle*) to medium severity, and plot (*bottom*) to low severity.

### 5.3.3 Empirical results (2)

The aim of this second set of experiments was to compare ADACC with state-of-the-art systems. Different categories of predictive systems were evaluated: single classifier methods, an ensemble method that adapt implicitly to concept drifts, an ensemble method that adapt to drifts by explicitly detecting changes in the environment and an ensemble method that handles recurring concepts. The methods were confronted with artificial and real benchmark datasets. To the best of our knowledge, there exist no benchmark dataset simulating predictable concept drifts. Therefore, only the case of recurrence is evaluated.

#### 5.3.3.1 Experiments and datasets

We evaluated ADACC against the following methods:

- **Simple incremental classifier (SIC)**: a single incremental classifier.
- **Moving window (MW)**: a single classifier that learns incrementally learns over the last $w$ instances.
- **Weighted examples (WE)**: a single classifier that gives larger weights to recent examples in order to gradually forget the outdated information.
- **Dynamic Weighted Majority (DWM)** [59]: an ensemble method that adapts implicitly to concept drift (i.e. without using a drift detection system)
- **Leveraging Bagging (LBAG)** [12]: a version of online bagging that uses the ADWIN method [9] to detect concept drifts.
- **Early Drift Detection Method (EDDM)** [6]: a classifier with a drift detection system.
- **Conceptual Clustering and Prediction (CCP)** [52] : an ensemble method able to handle recurring concepts.
- **DACC** the mere adaptive side of the ADACC approach.

DWM, LBAG, EDDM and CCP are explained in Chapter 3 and DACC in Chapter 4.

We carried out experiments on two artificial datasets (STAGGER and ELIST) and one real dataset (SPAM). These datasets are well-known benchmarks (see for instance [13, 52, 59, 96]). The datasets are presented in see Section 2.3.5. For STAGGER, we generated 10,000 training instances for each concept chosen uniformly from the instance space. To

simulate recurring contexts, we concatenated the original sequence with a copy of it, creating a stream of size 60,000.

Incremental Naive Bayes classifiers were used as base learners, because they naturally learn incrementally and are often used in studies of on-line learning.

LBAG and EDDM algorithms are available in the MOA (*Massive Online Analysis*) API[2]. We implemented all remaining algorithms on top of MOA, except for CCP and DWM whose results reported below are taken from the work of Katakis et al. in [52]. The parameters of DWM were set to $\rho = 0.5$, $\theta = 0.01$, $p = 1$ and those of CCP were set to $b = 50$, $c_{max} = 10$ and $\theta = 4$ for ELIST and $\theta = 2.5$ for SPAM. Both DWM and CCP were not evaluated on STAGGER in [52] and thus no results are reported on this dataset. We evaluated MW with three different window sizes: 50, 100 and 200 (retaining the best one: 100). WE was tested with the weighting formula $w(n) = w(n-1)+n^2$ where $w(n)$ is the weight of the $n$-th example. The threshold values of EDDM are automatically set by MOA. In LBAG, we tuned the parameters $\lambda$ and $\gamma$ according to preliminary experiments with $\lambda \in \{6, 10, 20\}$ and $\gamma \in \{0.002, 0.01, 0.1\}$. We converged on $\lambda = 20, \gamma = 0.002$ for ELIST and STAGGER and on $\lambda = 20, \gamma = 0.01$ for SPAM. Error correcting codes can be used for LBAG to add more diversity in the ensemble, but this does not improve the accuracy and is thus not used.

The parameters of ADACC, our *anticipative meta-learning* approach, were set to $\theta_I = 0.8$, $\theta_d = 0.7$ and $\tau_s = 100$ in all experiments. The anticipation mechanism is called every $p = 100$ time steps. Finally, the parameters of the *adaptive learning* mechanism, DACC, were optimized after preliminary experiments with $\tau_{eval}, \tau_{mat} \in \{10, 20, 50, 100\}$. They were set and $\tau_{eval} = \tau_{mat} = 20$ were used for all datasets. The ensemble size was fixed to 20 for DACC and LBAG.

#### 5.3.3.2 Evaluation measures and methodology

In the experiments, we evaluated the accuracy (predictive performance), precision, recall and run time of each approach.

In a document retrieval task, *precision* is the fraction of retrieved documents that are relevant [49], while *recall* (or sensitivity) is the fraction of relevant documents that are retrieved [18]. Relevant documents are generally labeled to class "1" while the non-relevant documents are labeled to class "-1" or "0". In a more general formulation, the precision can be seen as the number of instances correctly labeled to class "1" by the predictive system out of all the examples belonging to class "1", while the precision is

[2]http://sourceforge.net/projects/moa-datastream/

the number of instances correctly labeled to class "1" by the predictive system out of all the examples labeled to class "1" by the predictive system. These two measures are calculated as follows:

$$precision = \frac{tp}{tp + fn}$$

$$recall = \frac{tp}{tp + fp}$$

where $tp$ are the true positive instances i.e. the instances labeled correctly to class "1" by the predictive system, $tn$ are the true negative instances i.e. instances labeled correctly to class "0", $fp$ ($fn$) are the false positive (negative) instances i.e. labeled uncorrectly to the positive (negative) class.

We also studied the impact of the two threshold values: the threshold that decides of the stability of the environment $\theta_I$ and the threshold that determines the concept equivalence $\theta_d$. Their impact on both the memory of snapshots and the predictive performance of ADACC is reported.

#### 5.3.3.3 Empirical results

Table 5.4, 5.5 and 5.6 report the results on the ELIST, SPAM and STAGGER datasets averaged over 10 runs showing the accuracy, precision, recall and the run time in CPU seconds. All experiments were executed on an Intel Core i5 CPU at 2.4 GHz with 4.0 GB of RAM. The execution time of DWM and CCP are not given since they were tested in [52] on a different machine.

In all cases, ADACC yields the best accuracy. Figure 5.24 (top) shows a moving average of the accuracy of DACC, ADACC, LBAG and EDDM over sliding windows of 1,000 instances. DACC adapts faster to concept drifts than EDDM and LBAG, probably because of the frequent removal and addition of classifiers (every 20 time steps) which makes it ready to any upcoming change. However, despite the very good performance of DACC, ADACC still tops it by recognizing recurring concepts starting at time step 30,000. This comes at the expense of the execution time which is multiplied by a factor of 1.5 to 3. However, the amount of computation can be reduced by increasing the value of $p$, the period separating two calls of the meta-learning mechanism. In STAGGER, the run time of ADACC is reduced to 2.8 CPU seconds when $p = 1,000$ time steps (instead of 100). Note that the classification performance is not hurt as long as $p$ is small enough to take snapshots of all encountered concepts (i.e. $p < 10,000$ for STAGGER).

TABLE 5.4: The accuracy, precision, and recall (in %) along with the execution time (in CPU seconds) of the different approaches on the ELIST dataset using Naive Bayes classifiers as base learners.

| | **ELIST** | | | |
|---|---|---|---|---|
| Algorithm | Acc. | Precis. | Recall | Time |
| SIC | 54.2 | 50.7 | 69.3 | 0.53 |
| MW | 74.7 | 70.6 | 78.4 | 0.48 |
| WE | 66.9 | 64.9 | 63.9 | 0.53 |
| DWM | 43.8 | 47 | 42.5 | - |
| CCP | 77.5 | 79.7 | 77.6 | - |
| EDDM | 75.6 | 72.9 | 75.9 | 1.15 |
| LBAG | 58.5 | 54.4 | 68.3 | 15.0 |
| DACC | 76.2 | 73.8 | 75.9 | 9.52 |
| ADACC | **77.5** | **75.2** | **77.2** | 13.6 |

TABLE 5.5: The accuracy, precision, and recall (in %) along with the execution time (in CPU seconds) of the different approaches on the SPAM dataset using Naive Bayes classifiers as base learners.

| | **SPAM** | | | |
|---|---|---|---|---|
| Algorithm | Acc. | Precis. | Recall | Time |
| SIC | 90.7 | 94.2 | 93.2 | 1.27 |
| MW | 90.7 | 90.6 | 97.5 | 1.28 |
| WE | 92.8 | 95.2 | 95.0 | 1.28 |
| DWM | 91.8 | 84.8 | 83.1 | - |
| CCP | 92.3 | 85.7 | 83.9 | - |
| EDDM | 90.8 | 92.0 | 95.9 | 1.68 |
| LBAG | 91.8 | 95.6 | 93.3 | 14.60 |
| DACC | 94.7 | 95.1 | 97.8 | 11.9 |
| ADACC | **94.9** | **95.6** | **97.6** | 18.92 |

TABLE 5.6: The accuracy, precision, and recall (in %) along with the execution time (in CPU seconds) of the different approaches on the STAGGER dataset using Naive Bayes classifiers as base learners.

| | **STAGGER** | | | |
|---|---|---|---|---|
| Algorithm | Acc. | Precis. | Recall | Time |
| SIC | 64.5 | 62.4 | 89.5 | 0.35 |
| MW | 98.8 | 98.4 | 99.3 | 0.39 |
| WE | 78.5 | 77.6 | 85.9 | 0.4 |
| EDDM | 99.7 | 99.7 | 99.8 | 0.6 |
| LBAG | 89.9 | 85.7 | 98.1 | 1.51 |
| DACC | 99.9 | 99.9 | 99.9 | 1.06 |
| ADACC | **99.9** | **99.9** | **99.9** | 3.22 |

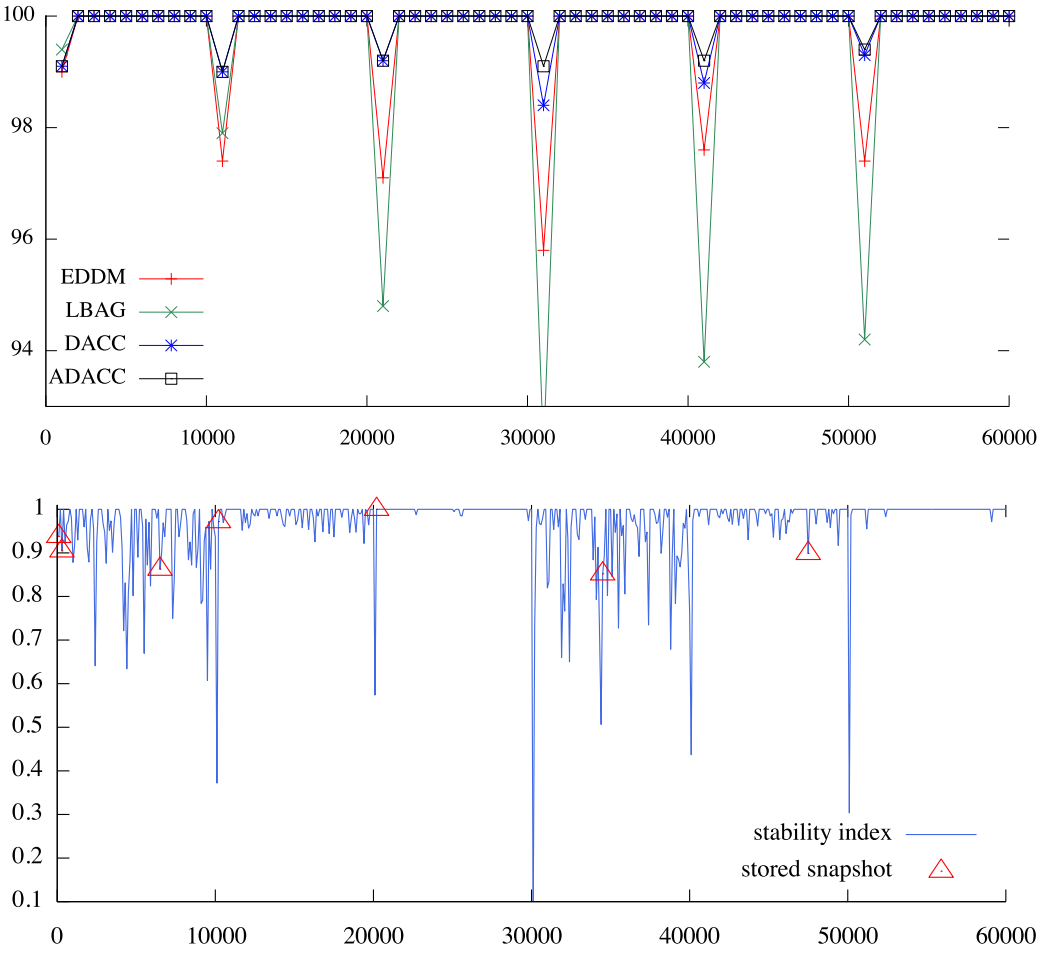


FIGURE 5.24: (Top) The classification accuracy of ADACC, DACC, LBAG and EDDM on STAGGER, averaged over sliding windows of size 1,000. (Bottom) The evolution of the stability index on STAGGER.

Figure 5.24 shows the stability index on the STAGGER dataset and highlights when snapshots are stored. All data points above $\theta_I = 0.8$ are candidate snapshots (a total of 575) but only 5 are kept as relevant in the list $\mathcal{M}_{LT}$. The unstable behavior of the stability index from time step 1 to 10,000 reflects the difficulty of learning the first concept. Three different snapshots of the first concept are stored during this period, capturing different subspaces. Only one additional snapshot is stored for the first concept when it reappears (time steps 30,000 to 40,000) confirming that redundant snapshots are avoided by ADACC. The second concept is learnt more easily and only two distinct snapshots are taken. Learning the third concept corresponds to the highest stability index, suggesting a rather easy learning task. Only one representative snapshot of the third concept is stored at the end of the experiment. Regarding the other streams, a total of 8 snapshots were stored for SPAM and 4 for ELIST.

TABLE 5.7: The effect of the conceptual equivalence threshold on ADACC with $\theta_I = 0.8$.

| $\theta_d$ | # candidates | # stored | Accuracy |
|---|---|---|---|
| 0.5 | 575 | 4 | 99.918 |
| 0.6 | 575 | 5 | 99.915 |
| 0.7 | 575 | 7 | 99.916 |
| 0.8 | 575 | 9 | 99.918 |
| 0.9 | 575 | 12 | 99.918 |

TABLE 5.8: The effect of the stability index threshold on ADACC with $\theta_d = 0.7$.

| $\theta_I$ | # candidates | # stored | Accuracy |
|---|---|---|---|
| 0.5 | 596 | 9 | 99.908 |
| 0.6 | 594 | 9 | 99.908 |
| 0.7 | 587 | 9 | 99.91 |
| 0.8 | 575 | 7 | 99.916 |
| 0.9 | 540 | 4 | 99.916 |

In our experiments, the threshold values for stability and conceptual equivalence were fixed. We varied their values on the STAGGER dataset to study their effect on the number of candidate snapshots, the number of stored snapshots and the classification performance of ADACC. The results are shown in Tables 5.7 and 5.8. Smaller stability thresholds increase the number of candidate snapshots and thus of the stored ones but very much less significantly. Only 5 additional snapshots are stored when the threshold switches from 0.9 to 0.5. When varying the concept equivalence threshold, the number of candidates doesn't change since it is only related to the stability threshold value. The stored snapshots however evolve. Larger threshold values entice larger numbers of stored snapshots. Remarkably, changing both threshold values may impact the accuracy of the anticipative mechanism (ADACC) but never to the extent of being worse than the mere adaptive scheme (DACC).

The comparisons with state-of-art methods were the last experiments we conducted for this thesis. We wished to go further with our evaluation of the ADACC system, comparing its performance with other systems dealing with recurrence [3, 37, 41, 99]. However, for the sake of computational speed, the sole recurrence handling system we evaluated ADACC against was CCP since it was already tested in [52] on benchmark datasets available for the MOA API. We keep the comparisons with ADACC's other concurrents as a future perspective for this work.

## 5.4 Contribution

In this chapter, we presented ADACC, a combination of a meta-learning mechanism and an adaptive ensemble of learners. The meta-learning mechanism (a) detects regularities (concepts) that are significant and new, (b) recognizes when to use past knowledge when old concepts reappear and (c) analyses how concepts evolve with time in the aim of detecting regularities in the evolution function to predict near-future likely concepts.

Unlike RePro (see Chapter 3) and other systems handling concept recurrence [3, 99], our method does not recognize new concepts using a drift detection system and thus it overcomes the difficulties inherent in having to detect gradual concept changes while still being robust to false alarms.

A main advantage of our meta-learning system is its ability to detect new concepts using a stability index measure, without relying on a priori knowledge on *when* the concept is expected to be stable. In PreDet and CCP systems for instance (see Chapter 3), each batch of $m$ time steps represents a new stable concept. This scenario entails two main difficulties. First, the batch size $m$ should be predefined. It should be high enough in order to be able to learn a stable representation and small enough to ensure that no drift occured inside the batch. Secondly, fixing the value of $m$ assumes that concepts evolve periodically, which is not a general rule in data streams.

Another advantage of ADACC lies in the combination of adaptation and anticipation on the *instance level*. Thus, each new example from the data stream can be labeled by the adaptive ensemble, the anticipated concept of the meta-learning mechanism or by both, according to the combination function. Other systems combine adaptation and anticipation on a *batch level* [52, 99] where the anticipated concept is used to classify the next $m$ instances. The main disadvantage is that, if the anticipated concept is not accurate, the next $m$ examples of the stream get labelled by unadapted models. Thus a wrong decision affects the classification accuracy on a batch of examples, in comparison with only one example in ADACC.

Finally, the meta-learning mechanism involves few parameters to set. As shown in the experimental results, the parameter values may slightly affect the classification accuracy of ADACC but never to the extent of being worse than the mere adaptive ensemble.

# Chapter 6

# Conclusion and Perspectives

In this concluding chapter, we summarize the major contributions of this thesis. We present the basic ideas behind the research methodology we employed, along with the main properties of the adaptive approach (DACC) and the meta-learning mechanism in (ADACC). We then discuss the areas of improvements and suggest possible perspectives. We end our discussion by making a link between this work and two fields of machine learning: **(a)** the *theory of online learning* and **(b)** *domain adaptation* also known as *transfer learning*.

## 6.1 DACC

In Chapter 4, we presented DACC, an online ensemble method designed to learn from data streams with the ability to *adapt to concept changes*.

### 6.1.1 Methodology

In a first step, we studied online ensemble methods that can learn under concept changes. We analyzed two main strategies used by these methods to forget outdated knowledge in an evolving environment: **(a)** deleting learners with poor predictive performance, according to a preset threshold value, and **(b)** deleting periodically the relatively worst learner in the ensemble. In both cases, deleted learners are replaced with new ones, trained on the most recent examples. While old learners are reliable in stable environments, youngest learners tend to be valuable during changing times.

The analysis showed that both strategies require prior knowledge of the dynamics of the environment in order to choose a threshold value for the strategy **(a)** or the deletion

frequency for the strategy **(b)**. Failing to choose adapted values can lead to unwanted results, either affecting the stability of the ensemble, or its plasticity towards potential concept changes.

This work was motivated by the need to mitigate the effect of prior knowledge. We focused on strategy **(b)** to avoid finding an appropriate threshold. Nevertheless, choosing the deletion frequency $f_{del}$ for strategy **(b)** does not come without difficulties. A high value for $f_{del}$ prones the newest learners in the ensemble to be deleted (if they did not have time to learn enough of the regularities in the environment) which impedes the ability of the system to adapt to concept changes. At the same time, $f_{del}$ cannot be too small otherwise the system looses any plasticity.

The question was then whether it was possible to remove learners frequently from the ensemble, in anticipation for change, but without removing the youngest experts when a concept change actually happens. To solve this problem, we decided to enlarge the candidates for deletion. Hence, instead of removing the worst learner in the ensemble, a learner is randomly selected from the $ds$ worst learners and is then forced to retire. This increases the expectation of young learners to survive deletion when $f_{del}$ is high. Finally, the last step was to choose the value for $ds$. While larger values increase the expectation of young learners to survive, they also remove "good" learners from the ensemble, affecting the stability of the system. According to experimental results, a deletion size that is equal to half the size of the ensemble offers a good trade-off.

The enhanced deletion strategy was the base for a new online ensemble method, called DACC.

### 6.1.2 Properties

DACC adapts to concept changes using a dynamic and diverse committee of experts (learners). The committee is *dynamic* since it constantly updates itself by removing the lowest performing experts and adding new experts. The committee is also *diverse* in the sense that it can be comprised of different types of experts (neural networks, polynomial regression models, SVMs etc...) and the experts also observe different windows of training data. The diversity in the window sizes allows our method to deal *implicitly* with the stability-plasticity dilemma. This advantage cannot be obtained with a single expert with a fixed history size.

Unlike traditional ensemble methods where the committee consists of weak experts that learn from the same concept and cooperate to give the final prediction [77], DACC is a mix of a *competitive* and a *cooperative* strategy between experts that might have been

trained on data coming from different concepts. The experts compete for their life in the ensemble and also to be heard in the voting process. The best learners cooperate by voting for the ensemble's final prediction.

The experimental results show that our approach overcomes many of the common difficulties encountered in the current ensemble methods that are designed to adapt to concept drifts. The main advantages of DACC include:

#### 6.1.2.1 Minimal a priori knowledge of the dynamics of the environment

Unlike many approaches, DACC doesn't rely on a threshold value to decide whether an expert is adapted to the current concept or whether it is outdated and should therefore be discarded. This redeems the algorithm from the burden of choosing the adapted threshold value, which obviously depends on the drift properties such as the speed or the severity.

In most classical methods, the deletion strategy removes the worst expert in the ensemble. Generally, the newest expert is prone to be deleted if it did not have time (according to a parameter $\tau_{mat}$) to learn enough of the regularities in the environment. DACC, however, removes expert(s) randomly from the worst half of the ensemble, which gives the opportunity for new promising experts to improve even when the parameter $\tau_{mat}$ is not finely tuned.

DACC, being less sensitive to its preset parameters, can adapt to a variety of concept changes and problems using the same fixed parameters values.

#### 6.1.2.2 Knowledge transfer

DACC doesn't remove its experts at once when a drift happens. A period of $\tau_{mat}$ time steps separate two consecutive deletion operations which ensures that the committee needs at least $N * \tau_{mat}$ time steps to be reset. This allows the committee to transfer knowledge from the old concept to the new one afer the drift, and if the concept change is continuous, DACC can still predict on the old concept instances during the drift.

#### 6.1.2.3 Fast adaptation

DACC removes experts constantly from the ensemble, regardless of whether the concept is stable or changing. In some cases, this leads to a faster adaptation to a concept change compared to approaches that wait until the weight of an expert goes below a threshold or that the explicit drift detection systems "feels" the change.

#### 6.1.2.4 Dynamic levels of diversity

The levels of diversity in the committee evolve dynamically with time. Generally, when the concept is stable, the diversity of the best experts decreases as they converge towards the near maximal classification accuracy. The diversity of the worst half of experts however is generally high since they are constantly removed in anticipation for change. When the concept evolves, the diversity increases among all experts as old well-performing experts become outdated and are replaced with new ones. The diversity then gets back to the former case (concept stability).

The implicit levels of diversity in DACC allows low diverse experts to predict well on the current stable concept, and makes highly diverse experts ready to any upcoming change. Monitoring the evolution of the concept diversity provides means to detect states of stability by the anticipative meta-learning system (ADACC).

### 6.1.3 Strengths, weaknesses and perspectives

A large number of online methods rely on a threshold value to adapt to concept changes. The threshold value is generally used to decide whether an expert is outdated and hence should be deleted from the ensemble, or to detect concept changes explicitly by monitoring, for instance, the ensemble's classification performance. In all cases, the threshold value plays a key role in the ability of the system to detect and adapt to upcoming changes. In DACC, experts are removed frequently from the ensemble, without analyzing whether the underlying target concept is stable or changing. Thus, in contrast to other ensemble methods, DACC is not faced with the problem of not adapting to a concept change as there always exist young experts in the ensemble that are ready to learn the new concept.

Thanks to the frequent experts' deletions, DACC reacts faster to concept drifts than other ensemble methods adapting implicitly to changes. Nevertheless, DACC's reactivity is not as fast as this of systems using explicit drift mechanisms when the concept drift is *sudden* and *severe*, which is generally the easiest case for a drift detection system. One possible future direction is to find ways to combine an implicit adaptive ensemble with an ensemble using a drift detection system, in order to profit from the advantages of both approach while mitigating their drawbacks.

When learning from examples of the data stream, DACC gives the same importance to examples of each class, therefore making the assumption that classes are balanced. According to the empirical results, this leads to difficulties when learning in the presence of *imbalanced classes*. A possible solution is to oversample the minority class(es) and/or

subsampling the majority class(es) in order to overcome the class imbalance. This can be achieved, for instance, by assigning weights to training examples that are inversely proportionnal to the observed frequency of the examples' class, giving more importance to minority classes.

Finally, it was shown in the empirical results that DACC's execution time compares favorably to other ensemble methods. However, since the experts in the ensemble are updated and evaluated independently from each other, DACC can be parallelized in order to speed up its run time.

## 6.2 ADACC

In Chapter 5, we presented ADACC, a meta-learning mechanism that *anticipates concept changes*, relying on the adaptive ensemble of DACC.

### 6.2.1 Methodology

Our goal was to take advantage of the changes in the environment as means to anticipate near future characteristics, allowing us to act pro-actively to changes instead of adapting passively as with the majority of online learning methods.

We worked on what seemed to be two advantageous cases:

- *recurrence:* re-using previously learnt concepts when they reappear
- *predictability:* analyzing the evolution of the underlying target concept in order to predict the likely future concepts.

Both recurrence and predictability imply that the past history of stable concepts be captured. Relying on a drift detection mechanism was a possible way to identify the different concepts. However, the former systems are known to have difficulties to detect gradual concept changes while still being robust to false alarms, and thus were avoided.

During our experimental study on DACC, we noticed that the diversity levels of the adaptive ensemble evolve with time depending on whether the environment is stable or changing. Periods of stability are characterized with small levels of diversity and also small error rates in contrast with periods of change where outdated experts are generally removed, increasing the diversity in the ensemble.

Accordingly, we defined a stability index that allows us to identify periods of stability and memorize "snapshots" of the stable concepts learnt by the adaptive ensemble. These snapshots are used as training samples to a second order learning system that predicts future concepts. The snapshots also represent the memory of the past and thus can turn to be useful when old concepts reappear.

### 6.2.2 Properties

The main contribution of ADACC lies in:

- the use of a stability measure that monitors the ensemble of adaptive learners
- the long-term memory of past useful concepts.

Snapshots of the relevant states of the world are stored and re-used when old contexts reappear. The evolution of the stored snapshots is also analyzed to predict likely future states of the environments (i.e. future concepts). The meta-learning mechanism is completely embedded in the natural functioning of the adaptive ensemble method with few parameters to set.

The empirical evaluation explored various conditions for evolving data streams. It showed that as soon as the concept changes are significant (medium or high severity), second order learning yields substantial gains in prediction performance over a mere adaptation policy. Furthermore, second order learning can only improve and never deteriorate the prediction performance. Experiments on real and artificial benchmark datasets and comparisons with various online learning systems show that ADACC brings improvement in the classification performance of DACC, outperforming all compared systems.

### 6.2.3 Strengths, weaknesses and perspectives

ADACC is a general framework to endow adaptive online learning systems based on an ensemble approach with second order learning capacity.

The second order learning keeps in memory a list of the stable concepts encountered so far, or what we call "snapshots" of the stable concepts. The memorized snapshots depend mainly on two preset parameters: a stability threshold $\theta_I$ and a concept equivalence threshold $\theta_d$. By setting a high value for $\theta_I$, we might miss a snapshot in the list. Hence, when choosing the parameter values, we make sure that $\theta_I$ is not very high. Then, the impact of the preset values will be on the size of the list as redundant snapshots might be stored. When dealing with concept recurrence, the presence of redundant snapshots does

not harm the predictive performance of ADACC. However, when dealing with concept predictability, redundant snapshots will be considered as noisy samples when analyzing the sequence of supposedly different concepts in the list. This means that the threshold values do not need to be finely tuned for *concept recurrence* in contrast with *concept predictability* that may require prior knowledge of the evolving environment to avoid noise. However, as previously mentionned, ADACC can only bring gain to the adaptive strategy. Hence, even when the parameters are not adapted to the learning problem, the worst scenario for ADACC is to have the same predictive accuracy as the adaptive ensemble.

All of our experiments regarding concept predictability were applied on artificial datasets to simulate regular concept changes. As future work, it would be interesting to look for real-life scenarios where the underlying target concept evolves in a regular way, according to a particular evolution function, and thus can benefit from the concept prediction approach. It would also be interesting to predict not only the near future concept changes but also the changes on the long-term.

An important improvement of ADACC is to find ways to keep constant the size of the long term memory of the memorized snapshots. As previously mentionned in Chapter 5, a promising avenue is to store prototypes of snapshots instead of the original ones, using a hierarchical clustering technique.

Finally, a theoretical study of the system's accuracy under different types of concept drifts are possible extensions of this work.

## 6.3 Links with the Theory of Online Learning

The *classical theory of supervised machine learning* assumes constrained learning scenarios where the training data are assumed to be independent and identically distributed (*i.i.d.*) and the learning environment is static, that is, the distribution of the training data $P(\mathbf{x}, y)$ is supposedly stable [94]. Constraining the learning conditions allows one to assess the performance of specific learning algorithms relative to a loss function (see Equation 2.1). However, the restrictions imposed are not always present when learning from data streams which makes the classical theoretical framework unadapted to the context of online learning.

The *theory of online learning* as studied by Cesa-Bianchi [20] makes no assumptions about the learning environment. In contrast to the theory of classical machine learning, the training examples are not assumed to be *i.i.d.*. They can be time-dependent and their distribution $P(\mathbf{x}, y)$ can evolve with time creating concept changes. Nevertheless,

enlarging the scope of the learning problem limits the use of a real risk or loss measure as in classical machine learning, and thus does not make it possible to give theoretical bounds on the performance of specific learning algorithms.

In this work, we made benign assumptions about the dynamics of the environment such as it evolves gradually or it evolves with sudden changes but interspersed with stationary states. Hence, we assumed a sequence of predictions that are not random as in the theory of online learning but also not as constrained as in classical machine learning. Developing a middle ground theory in between the very general online theory and the very restricted classical theory remains an important research area. It is important to note that some theoretical studies were conducted in this domain. Generally, some restrictions are imposed on the type of admissible concept change, such as the rate of change [63] or the severity of change [46]. The main disadvantage is that these very special cases don't usually occur.

## 6.4 Links with Domain Adaptation and Transfer Learning

In classical supervised machine learning, it is assumed that the training examples (used to learn a predictive model) and the testing examples (used to evaluate the learnt model) are issued from the same static distribution [94]. However, models are generally built from some fixed source domain but are then tested on different target domains. For instance, in face detection systems, the images used as training examples can be captured in a particular position, orientation and lighting conditions. When tested on video sequences, face detection systems will probably be confronted with a different distribution of images, with arbitrary poses and lighting conditions.

*Domain adaptation* or *transfer learning* aims to build predictive models or classifiers that are robust to mismatched distributions, relying on the presence of some examples from the target domain [68]. The examples can either be labeled, leading to a semi-supervised learning problem, or unlabeled, leading to a unsupervised learning problem.

In online machine learning, the distribution of the examples in the feature space $P(\mathbf{x})$ can also evolve with time, creating what is called a *virtual concept drift* [88] or a *covariate shift* [8]. However, covariate shift in online learning differs from the problem of domain adaptation or transfer learning in two major points.

First, in domain adaptation, we know *when* the change in the distribution will happen, which is when the predictive model is used on testing examples. In online learning, we have no a priori knowledge of the time of change: a covariate shift may happen at any time or may not happen at all. In addition, the changes in distributions may happen

gradually, in contrast with domain adaptation where the change is abrupt, happening at the time of testing. Secondly, domain adaptation relies on the presence of examples from the target (testing) distribution. In online learning, as there is no a priori information on when the covariate shift might happen, we can't have samples of examples belonging to the target distribution before a covariate shift is detected.

Our work can be seen as an *adaptive transfer learning*, that is, domain adaptation without prior knowledge of the examples from the target domain. New domains are *detected* implicitly by evaluating the experts in the adaptive ensemble relative to their predictive performance on recent streaming data. Then, *adaptation* is realized by suppressing relatively bad experts, considered as unadapted to the new domain, and by replacing them with new experts, trained only on examples from the new domain.

# Bibliography


[1] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine learning*, 6(1):37–66, 1991.

[2] J. H. Ahrens and U. Dieter. Computer methods for sampling from gamma, beta, poisson and bionomial distributions. *Computing*, 12(3):223–246, 1974.

[3] C. Alippi, G. Boracchi, and M. Roveri. Just-in-time classifiers for recurrent concepts. *IEEE transactions on neural networks and learning systems*, 24(4):620–634, 2013.

[4] L. Arge. The buffer tree: A new technique for optimal I/O-algorithms. In *Algorithms and Data Structures, Lecture Notes in Computer Science*, volume 955, pages 334–345, 1995.

[5] Arthur Asuncion and David J Newman. Uci machine learning repository, 2007.

[6] M. Baena-García, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldà, and R. Morales-Bueno. Early drift detection method. *Fourth International Workshop on Knowledge Discovery from Data Streams*, 2006.

[7] P.L. Bartlett, S. Ben-David, and S.R. Kulkarni. Learning changing concepts by exploiting the structure of change. *Machine Learning*, 41(2):153–174, 2000.

[8] S. Bickel, M. Brückner, and T. Scheffer. Discriminative learning under covariate shift. *The Journal of Machine Learning Research*, 10:2137–2155, 2009.

[9] A. Bifet. Adaptive learning and mining for data streams and frequent patterns. *ACM SIGKDD Explorations Newsletter*, 11(1):55–56, 2009.

[10] A. Bifet and E. Frank. Sentiment knowledge discovery in twitter streaming data. In *Discovery Science*, pages 1–15. Springer, 2010.

[11] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. Moa: Massive online analysis. *The Journal of Machine Learning Research*, 99:1601–1604, 2010.

[12] A. Bifet, G. Holmes, and B. Pfahringer. Leveraging bagging for evolving data streams. In *Machine Learning and Knowledge Discovery in Databases*, pages 135–150. Springer, 2010.

[13] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data mining*, pages 139–148, 2009.

[14] C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*, volume 1. Springer New York, 2006.

[15] M. Bottcher, M. Spott, and R. Kruse. Predicting future decision trees from evolving data. In *ICDM'08. Eighth IEEE International Conference on Data Mining*, pages 33–42, 2008.

[16] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[17] L. Breiman, J. H. Friedman, R. Olshen, C. J. Stone, L. Breiman, W. Hoeffding, R. J. Serfling, J. H. Friedman, O. Hall, P. Buhlmann, et al. Classification and regression trees. *Machine Learning*, 19:293–325, 1984.

[18] M. K. Buckland and F. C. Gey. The relationship between recall and precision. *JASIS*, 45(1):12–19, 1994.

[19] J. Carletta. Assessing agreement on classification tasks: the kappa statistic. *Computational linguistics*, 22(2):249–254, 1996.

[20] N. Cesa-Bianchi. *Prediction, learning, and games*. Cambridge University Press, 2006.

[21] Y. Cheng, W. Qi, and W. Cai. Dynamic properties of elman and modified elman neural network. In *Machine Learning and Cybernetics, 2002. Proceedings. 2002 International Conference on*, volume 2, pages 637–640. IEEE, 2002.

[22] A. Cornuéjols. On-line learning: where are we so far? In *Ubiquitous knowledge discovery*, pages 129–147. Springer, 2010.

[23] Airlines dataset. http://moa.cms.waikato.ac.nz/datasets/.

[24] PAKDD 2009 dataset. http://sede.neurotech.com.br/PAKDD2009/.

[25] S. J. Delany, P. Cunningham, A. Tsymbal, and L. Coyle. A case-based technique for tracking concept drift in spam filtering. *Knowledge-Based Systems*, 18(4):187–195, 2005.

[26] C. P. Diehl and G. Cauwenberghs. Svm incremental learning, adaptation and optimization. In *Proceedings of the International Joint Conference on Neural Networks*, volume 4, pages 2685–2690. IEEE, 2003.

[27] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923, 1998.

[28] T. G. Dietterich. Ensemble methods in machine learning. In *Multiple classifier systems*, pages 1–15. Springer, 2000.

[29] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *The Journal of Artificial Intellige Research*, 2:263–286, 1995.

[30] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD International Conference on Knowledge discovery and data mining*, pages 71–80, 2000.

[31] R. Elwell and R. Polikar. Incremental learning of concept drift in nonstationary environments. *Neural Networks, IEEE Transactions on*, 22(10):1517–1531, 2011.

[32] W. Fan, I. Davidson, B. Zadrozny, and P. S. Yu. An improved categorization of classifier's sensitivity on sample selection bias. In *Data Mining, Fifth IEEE International Conference on*, pages 4–pp. IEEE, 2005.

[33] Y. Freund, R. Schapire, and N. Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771–780):1612, 1999.

[34] J. H. Friedman. On bias, variance, 0/1—loss, and the curse-of-dimensionality. *Data mining and knowledge discovery*, 1(1):55–77, 1997.

[35] A. R. Gallant and W. A. Fuller. Fitting segmented polynomial regression models whose join points have to be estimated. *Journal of the American Statistical Association*, 68(341):144–147, 1973.

[36] J. Gama. *Knowledge discovery from data streams*. CRC Press, 2010.

[37] J. Gama and P. Kosina. Tracking recurring concepts with meta-learners. In *Progress in Artificial Intelligence*, pages 423–434. Springer, 2009.

[38] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *Advances in Artificial Intelligence–SBIA 2004*, pages 286–295. Springer, 2004.

[39] J. Gama, R. Rocha, and P. Medas. Accurate decision trees for mining high-speed data streams. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 523–528. ACM, 2003.

[40] J. Gama, R. Sebastião, and P. P. Rodrigues. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 329–338. ACM, 2009.

[41] J. B. Gomes, P. A. Sousa, and E. Menasalvas. Tracking recurrent concepts using context. *Intelligent Data Analysis*, 16(5):803–825, 2012.

[42] H. Guillaume, M. Dubois, P. Tarroux, and E. Frenoux. Temporal bag-of-words: A generative model for visual place recognition using temporal integration. In *Proceedings of the International Conference on Computer Vision Theory and Applications*, 2011.

[43] M. T. Hagan, H. B. Demuth, M. H. Beale, et al. *Neural network design*. Pws Pub. Boston, 1996.

[44] M. Harries and N. S. Wales. Splice-2 comparative evaluation: Electricity pricing. Technical report, Technical report, The University of South Wales, 1999.

[45] M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, and B. Scholkopf. Support vector machines. *Intelligent Systems and their Applications, IEEE*, 13(4):18–28, 1998.

[46] David P Helmbold and Philip M Long. Tracking drifting concepts by minimizing disagreements. *Machine Learning*, 14(1):27–45, 1994.

[47] G. Jaber, A. Cornuéjols, and P. Tarroux. Predicting concept changes using a committee of experts. In *ICONIP'11 International Conference on Neural Information Processing*, pages 580–588, Shanghai, China, 2011.

[48] G. Jaber, A. Cornuéjols, and P. Tarroux. Reacting to concept changes using a committee of experts. In *Proc. of Atelier CIDN (Classifications incrémentales et méthodes de détection de nouveauté) at EGC-2012*, pages 31–47, 2012.

[49] T. Joachims. *Text categorization with support vector machines: Learning with many relevant features*. Springer, 1998.

[50] A. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, 14:841, 2002.

[51] G. N. Karystinos and D. A. Pados. On overfitting, generalization, and randomly expanded training sets. *Neural Networks, IEEE Transactions on*, 11(5):1050–1057, 2000.

[52] I. Katakis, G. Tsoumakas, and I. Vlahavas. Tracking recurring contexts using ensemble classifiers: an application to email filtering. *Knowledge and Information Systems*, 22(3):371–391, 2010.

[53] M. Kearns and D. Ron. Algorithmic stability and sanity-check bounds for leave-one-out cross-validation. *Neural Computation*, 11(6):1427–1453, 1999.

[54] R. Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis*, 8(3):281–300, 2004.

[55] R. Klinkenberg and T. Joachims. Detecting concept drift with support vector machines. In *ICML*, pages 487–494, 2000.

[56] R. Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, volume 14, pages 1137–1145, 1995.

[57] R. Kohavi, D. H. Wolpert, et al. Bias plus variance decomposition for zero-one loss functions. In *ICML*, pages 275–283, 1996.

[58] J. Z. Kolter and M. A. Maloof. Using additive expert ensembles to cope with concept drift. In *Proceedings of the 22nd international conference on Machine learning*, pages 449–456. ACM, 2005.

[59] J. Z. Kolter and M. A. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *The Journal of Machine Learning Research*, 8:2755–2790, 2007.

[60] P. S. A. Krogh. Learning with ensembles: How over-fitting can be useful. In *Proceedings of the 1995 Conference*, volume 8, page 190. The MIT Press, 1996.

[61] M. Kubat. Flexible concept learning in real-time systems. *Journal of Intelligent and Robotic Systems*, 8(2):155–171, 1993.

[62] A. Kuh, T. Petsche, and R. L. Rivest. Learning time-varying concepts. In *Proceedings of the 1990 conference on Advances in neural information processing systems 3*, pages 183–189. Morgan Kaufmann Publishers Inc., 1990.

[63] Anthony Kuh, Thomas Petsche, and Ronald L Rivest. Learning time-varying concepts. In *Proceedings of the 1990 conference on Advances in neural information processing systems 3*, pages 183–189. Morgan Kaufmann Publishers Inc., 1990.

[64] L. I. Kuncheva. Classifier ensembles for changing environments. In *Multiple classifier systems*, pages 1–15. Springer, 2004.

[65] C. E. Leiserson, R. L. Rivest, C. Stein, and T. H. Cormen. *Introduction to algorithms*. The MIT press, 2001.

[66] Y. Liu, X. Yao, and T. Higuchi. Evolutionary ensembles with negative correlation learning. *Evolutionary Computation, IEEE Transactions on*, 4(4):380–387, 2000.

[67] D. D. Margineantu and T. G. Dietterich. Pruning adaptive boosting. In *ICML*, volume 97, pages 211–218. Citeseer, 1997.

[68] A. Margolis. A literature review of domain adaptation with unlabeled data. *University of Washington, http://ssli. ee. washington. edu/˜ amargoli/review_ Mar23. pdf*, 2011.

[69] L. Minku, A. P. White, and X. Yao. The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering*, 22(5):730–742, 2010.

[70] L. Minku and X. Yao. DDD: A new ensemble approach for dealing with concept drift. *IEEE transaction on knowledge and data engineering*, pages 619–633, 2012.

[71] C. Z. Mooney, R. D. Duval, and R. Duvall. *Bootstrapping: A nonparametric approach to statistical inference*. Number 94-95. Sage, 1993.

[72] R. Nallapati. Discriminative models for information retrieval. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 64–71. ACM, 2004.

[73] J. Neter, W. Wasserman, M. H. Kutner, et al. *Applied linear statistical models*, volume 4. Irwin Chicago, 1996.

[74] Usenet news filtering dataset. http://www.liaad.up.pt/kdus/products/datasets-for-concept-drift.

[75] J. R. Norris. *Markov chains*. Number 2008. Cambridge university press, 1998.

[76] N. C. Oza and S. Russell. Experimental comparisons of online and batch versions of bagging and boosting. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 359–364. ACM, 2001.

[77] N. C. Oza and S. Russell. Online bagging and boosting. In *Artificial Intelligence and Statistics 2001*, 2001.

[78] R. Polikar. Ensemble based systems in decision making. *Circuits and Systems Magazine, IEEE*, 6(3):21–45, 2006.

[79] A. Pronobis and B. Caputo. Cold: Cosy localization database. In *The International Journal of Robotics Research, 28(5)*, 2009.

[80] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[81] S. Ruping. Incremental learning with support vector machines. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 641–642, San Jose, CA , USA, 2001.

[82] R. E. Schapire. The boosting approach to machine learning: An overview. *Lecture Notes in Statistics*, pages 149–172, 2003.

[83] J. C. Schlimmer and R. H. Granger. Beyond incremental processing: Tracking concept drift. In *AAAI*, pages 502–507, 1986.

[84] M. Scholz and R. Klinkenberg. An ensemble classifier for drifting concepts. In *Proceedings of the Second International Workshop on Knowledge Discovery in Data Streams*, pages 53–64. Porto, Portugal, 2005.

[85] K. O. Stanley. Learning concept drift with a committee of decision trees. *UT-AI-TR-03-302, Department of Computer Sciences, University of Texas at Austin, USA*, 2003.

[86] G. Stoltz. Agrégation séquentielle de prédicteurs: méthodologie générale et applications à la prévision de la qualité de l'air et à celle de la consommation électrique. *Journal de la Société Française de Statistique*, 151(2):66–106, 2010.

[87] W. N. Street and Y. Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 377–382. ACM, 2001.

[88] N. A. Syed, H. Liu, and K. K. Sung. Handling concept drifts in incremental learning with support vector machines. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 317–321. ACM, 1999.

[89] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications*, 2009. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[90] A. Tsymbal, M. Pechenizkiy, P. Cunningham, and S. Puuronen. Dynamic integration of classifiers for handling concept drift. *Information Fusion*, 9(1):56–68, 2008.

[91] Alexey Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 2004.

[92] P. E. Utgoff, N. C. Berkman, and J. A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29(1):5–44, 1997.

[93] V. N. Vapnik. An overview of statistical learning theory. *Neural Networks, IEEE Transactions on*, 10(5):988–999, 1999.

[94] V. N. Vapnik. *The nature of statistical learning theory*. Springer, 2000.

[95] B. Vidakovic. Nonlinear wavelet shrinkage with bayes rules and bayes factors. *Journal of the American Statistical Association*, 93(441):173–179, 1998.

[96] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, 1996.

[97] B. Widrow and M. A. Lehr. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. volume 78, pages 1415–1442. IEEE, 1990.

[98] W. H. Wilson. A comparison of alternatives for recurrent networks. In *Proceedings of the sixth Australian Conference on Neurals Networks ACNN*, volume 93, pages 189–192, 1993.

[99] Y. Yang, X. Wu, and X. Zhu. Mining in anticipation for concept change: Proactive-reactive prediction in data streams. *Data mining and knowledge discovery*, 13(3):261–289, 2006.

[100] H. Zhang. The optimality of naive bayes. pages 562–567. The AAAI Press, 2004.

[101] I. Zliobaite. Learning under concept drift: an overview. Technical report, Vilnius University, Faculty of Mathematics and Informatics, 2009.