Méthodes d'ensembles pour l'apprentissage multi-tâche avec des tâches hétérogènes et sans restrictions

Jean-Baptiste Faddoul

Université de Lille

Laboratoire d'Informatique Fondamentale de Lille

THÈSE

présentée pour obtenir le grade de

Docteur en Sciences
Spécialité: Informatique

Par

Jean Baptiste FADDOUL

Soutenue le 18/06/2012

Sujet : Ensemble Methods to Learn Multiple Heterogenous Tasks without Restrictions.

Dirigée par Rémi Gilleron. Co-dirigée par Boris Chidlovskii et Fabien Torre

Travaux réalisés au sein de
Xerox Research Center Europe et INRIA Lille Nord Europe

Jury

Boris CHIDLOVSKII	Centre de recherche Xerox Europe	Encadrant
Patrick GALLINARI	Professeur à l'université de Paris6	Rapporteur
Rémi GILLERON	Professeur à l'université de Lille3	Directeur
Marc SEBBAN	Professeur à l'université de Saint-Étienne	Rapporteur
Fabien TORRE	Maître de Conférences à l'université de Lille3	Encadrant

I would like to dedicate this thesis to Anna, the girl who I met at the beginning of my thesis, and thought that It would be only for a beer with friends, but It turned out that it will be for a life partnership....

Acknowledgements

I would like to express my utmost gratitude to my advisors: Rémi Gilleron, Boris Chidlovskii and Fabien Torre. Along the period of my PhD they were with me all the time, always teaching me new things, giving me ideas and ensuring that my research is sound and rigorous. In addition to their invaluable scientific help, they were next to me when I faced some life problems, I really appreciate their help which I really needed.

I would like to thank the members of the jury: Prof. Patrick Gallinari and Prof. Marc Sebban for accepting to be part of my thesis defense and for their remarks and questions.

I would like to acknowledge the direct and inderect help thanks to the interesting scientific discussions and exchange of ideas with Stéphane Clinchant, Francis Maes, Nadi Tomeh, Craig Saunders and Andrew McCallum.

Finally, during a thesis we need a positive energy which can not be here without the support great friends and colleagues around us, to those friends I would like to say 'Big Thank' for every great moment spent with you, for every advise and moral support you have offered me. Thank You Thierry Jacquin and Jean-Pierre Chanod.

Résumée

Apprendre des tâches simultanément peut améliorer le performance de prédiction par rapport à l'apprentissage de ces tâches de manière indépendante. dans cette thèse, nous considérons l'apprentissage multi-tâche lorsque le nombre de tâches est grand. En outre, nous détendons des restrictions imposées sur les tâches. Ces restrictions peuvent trouvées dans les méthodes de l'état de l'art. Plus précisément on trouve les restrictions suivantes : l'imposition de la même espace d' étiquette sur les tâches, l'exigence des mêmes examples d'apprentissage entre tâches et / ou supposant une hypothèse de corrélation globale entre tâches.

Nous proposons des nouveaux classificateurs multi-tâches qui relaxent les restrictions précédentes. Nos classificateurs sont considérés en fonction de la théorie de l'apprentissage PAC des classifieurs faibles, donc, afin de parvenir à un faible taux d'erreur de classification, un ensemble de ces classifieurs faibles doivent être appris. Ce cadre est appelé l'apprentissage d'ensembles, dans lequel nous proposons un algorithme d'apprentissage multi-tâche inspirée de l'algorithme Adaboost pour seule tâche. Différentes variantes sont proposées également, à savoir, les forêts aléatoires pour le multi-tâche, c'est une méthode d'apprentissage d'ensemble, mais fondée sur le principe statistique d'échantillonnage Bootstrap.

Dans la première approche, les classifieurs faibles que nous considérons sont des *stumps* de décision 2-niveau pour différentes tâches. Un clas-

sificateur faible assigne une classe à chaque occurrence de deux tâches et s'abstenir de voter sur d'autres tâches. Les classifieurs faibles permettent de gérer les dépendances entre les tâches sur l'espace d'apprentissage. Nous introduisons différents apprenants efficaces pour apprendre ces classifieurs. Nous considérons ensuite Adaboost avec les classifieurs faibles qui peuvent s'abstenir et de l'adapter à la configuration de l'apprentissage multi-tâche. Dans une 'etude empirique, nous comparons les apprenants faibles.

Dans la seconde approche, nous développons l'environnement Adaboost multi-tâches, avec des arbres de décision comme classifieurs faibles. D'abord nous adaptons l'arbre de décision bien connue au réglage multi-tâches. Nous révisons la règle du gain d'information pour l'apprentissage des arbres de décision pour l'adapter au multi-tâche. Nous utilisons cette fonctionnalité pour développer un nouveau critère pour l'apprentissage des arbres de décision multi-tâches. Le critère guide de la construction de l'arbre par l'apprentissage des règles de décision à partir des données de tâches différentes, et représentant différents degrés de relations entre les tâches. Ensuite, nous modifions Adaboost pour pouvoir combiner un ensemble des arbres de décision multi-tâches.

Enfin, nous donnons une validation expérimentale qui montre que approche sur-performe des méthodes existants et permet d'apprendre des nouvelles configurations de tâches qui ne correspondent pas aux méthodes de l'état de l'art.

Abstract

Learning multiple related tasks jointly by exploiting their underlying shared knowledge can improve the predictive performance on every task compared to learning them individually. In this thesis, we address the problem of *multi-task learning (MTL)* when the tasks are heterogenous: they do not share the same labels (eventually with different number of labels), they do not require shared examples. In addition, no prior assumption about the relatedness pattern between tasks is made.

Our contribution to multi-task learning lies in the framework of ensemble learning where the learned function consists normally of an ensemble of "*weak*" hypothesis aggregated together by an ensemble learning algorithm (*Boosting*, *Bagging*, etc.). We propose two approaches to cope with heterogenous tasks without making prior assumptions about the relatedness patterns. For each approach, we devise novel multi-task weak hypothesis along with their learning algorithms then we adapt a boosting algorithm to the multi-task setting.

In the first approach, the weak classifiers we consider are 2-level decision stumps for different tasks. A weak classifier assigns a class to each instance on two tasks and abstain on other tasks. The weak classifiers allow to handle dependencies between tasks on the instance space. We introduce different efficient weak learners. We then consider Adaboost with weak classifiers which can abstain and adapt it

to multi-task learning. In an empirical study, we compare the weak learners and we study the influence of the number of boosting rounds.

In the second approach, we develop the multi-task Adaboost environment with Multi-Task Decision Trees as weak classifiers. We first adapt the well known decision tree learning to the multi-task setting. We revise the information gain rule for learning decision trees in the multi-task setting. We use this feature to develop a novel criterion for learning Multi-Task Decision Trees. The criterion guides the tree construction by learning the decision rules from data of different tasks, and representing different degrees of task relatedness. We then modify MT-Adaboost to combine Multi-task Decision Trees as weak learners.

We experimentally validate the advantage of our approaches; we report results of experiments conducted on several multi-task datasets, including the Enron email set and Spam Filtering collection.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Multi-task learning Caruana [1997] aims at improving the performance of related tasks by learning a model representing the common knowledge across the tasks. Traditionally, the existing techniques assume that tasks share the same instance and label space Pan and Yang [2008], in the case of classification Evgeniou and Pontil [2004]; Xue et al. [2007], regression Archembeau et al. [2011]; Dai et al. [2007], ranking Chapelle et al. [2010a] and feature learning Argyriou et al. [2006a].

However, in many natural settings these assumptions are not satisfied. A known example is the automatic categorization of Web pages into hierarchical directories, like DMOZ or Yahoo! Liu et al. [2005]. When building a categorizer for the Yahoo! directory, it is desirable to take into account DMOZ web directory, and vice versa. The two tasks are clearly related, but their label sets are not identical. Moreover, both ontologies can evolve with time when new categories are added to the directories and some old categories die naturally due to lack of interest.

Multi-task learning with no label correspondence was considered in Novi Quadrianto [2010], where the problem is formulated as learning the maximum entropy estimator $H(Y|X)$ for each task while maximizing the mutual information $-H(Y, Y')$ among the label sets Y and Y' of different tasks. Their approach relies on the hypothesis of the global correlation between tasks in the

whole learning space. Tests on the real world datasets show however that this global relatedness assumption turns to be too strong. Indeed, task relatedness may show up different degrees or even different signs in different regions of the learning space. It is therefore important that the multi-task learner copes with the varying relatedness of tasks, learns its different degrees and accommodates the inductive bias accordingly.

We are interested in the multi-task learning where label sets are close but differ from one task to another and the number of classes might be different across tasks. A motivating example is the automatic classification of e-mails in personal inboxes Mantrach and Renders [2012]. Similarly to the case of Yahoo! and DMOZ web directories, categories used in two e-mail inboxes may be related but not identical. For example, people may use *Family* or *Home* categories for personal e-mails and *Finance* or *Budget* for e-mails relevant to financial issues. The application becomes particularly critical when inboxes are owned by people from the same organization; they may share the same messages but classify them according to personal category names. We therefore expect that learning all tasks simultaneously can benefit to the classification model for each task.

Our contribution to multi-task learning lies in the framework of ensemble learning where the learned function consists normally of an ensemble of "*weak*" hypothesis aggregated together by an ensemble learning algorithm (*Boosting*, *Bagging*, etc.). We propose two multi-task ensemble learning approaches for tasks with different label sets which makes no assumption on global relatedness. For each one, we devise novel multi-task weak hypothesis along with their learning algorithms then we adapt a boosting algorithm to the multi-task setting.

In the first approach Faddoul et al. [2010] we propose a method for multi-task learning for tasks with different label sets which makes no assumption on global relatedness. For this purpose, we developed a multi-task learning algorithm *(MT-Adaboost)* which extends Adaptive boosting *(Adaboost)* Freund and Schapire [1996]; Schapire and Singer [1999] to the multi-task setting. The boosting tech-

nique is used to generate and combine multiple (weak) classifiers to improve the predictive accuracy. According to the boosting principle, a smart re-weighting of examples from different tasks without label correspondences can grasp the local relatedness of tasks.

As weak classifiers, we consider multi-task decision trees with only two levels (`2T-stumps`). When an instance is considered, the `2T-stump` assigns a label for at most two tasks and abstains for all other tasks. Thus, we consider the abstention not as an exception, but as the first class behavior for a weak classifier because abstaining on some tasks is a more natural choice that enforcing a weak classifier to make predictions for all tasks. We introduce and compare different weak learners for `2T-stumps`. We consider Adaboost with abstention as introduced in Schapire and Singer [1999]. We adapt it to the multi task setting and show convergence for training error. We consider different weighting schemes for Adaboost and compare the weighting schemes when the number of tasks increases. Last, we design experimental studies to compare the weak learners and to show the influence of the number of boosting rounds.

The method however suffers from some limitations. The algorithm which learns a multi-task stump level-by-level, is based on a heuristic choosing at the root the best N stumps (where the training error is the lowest); then for each it forwards recursively to the next levels to learn the remaining tasks. In this kind of a cascade classification on tasks, it learns at each node a classifier for a task taking into account the other tasks' classifiers in the node's ancestors. The intuition behind is as follows, If the tasks are related then learning one task would provide information which makes learning the others easier. Unfortunately, such a sequential design of multi-task stumps might perform poorly when its greedy algorithm fails to capture task relatedness. In addition, multi-task stumps are binary classifiers, and their extension to multiple multi-class tasks requires additional efforts. We have realized this extension by the adaptation of *Adaboost.MH* algorithm and a multi-class modification to decision stumps.

In the second approach, we first propose *Multi-Task Decision Tree (`MT-DT`)* as a multi-task weak classifier. We revisit the well known *C4.5* decision tree learning and adapt it to the multi-task setting. Decision trees are naturally multi-class classifiers, thus `MT-DT` can learn multiple multi-class classification tasks. Our main contribution is in proving that `MT-DT` can benefit from an improved information gain criterion due to the multi-tasking. Unlike multi-task stumps, the criterion used to learn the nodes makes use of the data from several tasks at each node. Second, we proceed by plugging the `MT-DT` in the boosting framework; we modify MT-Adaboost to cope with the multi-class problems accordingly. Our modification of MT-Adaboost adapts their *Adaboost.M1* algorithm.

The ability of `MT-DTs` to make use of the data from several tasks at each node is advantageous in capturing tasks relatedness. But, at the same time, it could cause some limitations -when the number of tasks becomes very large- for two reasons. First, the higher computational cost of learning the tree. Second, it becomes more difficult to learn a single tree that fits all tasks.

Abstaining on some tasks is a more natural choice than enforcing a weak classifier to make predictions for all tasks, especially when we have a large number of tasks. In such a setting, clustering the tasks by their relatedness is a preferable choice. Therefore, `2T-stumps` can be advantageous for learning large number of tasks since they predict on two tasks and abstain on the rest, then learning an ensemble of them by a boosting algorithm will implicitly induce a clustering on the tasks.

The thesis is organized as follows. In the next chapter we will go briefly through the statistical learning theory and the different approaches to supervised classification. Then we narrow the scope and review transfer learning approaches. Actually, multi-task learning can be seen as a transfer learning approach. We finally, present the prior-art on multi-task learning. In Chapter 3, we present our weak classifiers, `2T-stump` and `MT-DT`. Chapter 4 is dedicated to present and analyze our learning algorithms, the weak learners and the boosters. To this purpose

we start the chapter by an introduction of ensemble learning. In Chapter 5 we validate empirically our algorithms on synthetic data sets generated from Bayesian networks that model the task relatedness. Also, we experiment with real large scale data sets like email spam filtering and Enron emails classification. Finally, we conclude the manuscript and give our perspectives for the future works.

Chapter 2

Context and Related Work

Contents

This chapter spans the background knowledge needed to understand the rest of the thesis. Computational machine learning is the large domain which contains the presented work. To be more specific, our work on multi-task learning has its theoretical foundations in the statistical machine learning theory. This theory provides a framework and tools to design and analyze new machine learning methods. We are interested in the setting of supervised learning, where a teacher (human being) provides the learning machine with annotated data that serve as the training sample from which the learning model is induced. To this regard, we start the actual chapter by a section on supervised learning and how it was approached by statistical learning theory. We then present different basic supervised learning problems in the literature; they are important to understand and compare multi-task learning approaches.

Multi-task learning can be viewed as a special case of transfer learning; a domain inspired from human ability to share and transfer knowledge across learning tasks. For instance, a human finds it easy to learn juggling with clubs after having learned juggling with balls. Nevertheless, transfer learning is uni-directional, which means that there is a task that we want to improve its performance by transferring knowledge acquired by other tasks. However, in multi-task learning there is no such a notion. All tasks are equal and the goal is to improve each one by sharing knowledge with others. In order to understand multi-task learning from transfer learning perspective, we dedicate a section on transfer learning. We will give a unified definition of transfer learning, then we go through different transfer learning categories before presenting the approaches used to cope with transfer learning problems. We conclude the section by addressing a critical issue for transfer learning named *negative transfer*. In couple of words, it is when transfer learning gives the opposite to what it is expected; harming the performance of the task of interest, instead of improving it.

2.1 Supervised Learning

Supervised learning is the task of learning (inferring) a function (hypothesis) $h : \mathcal{X} \to \mathcal{Y}$, from a dataset (training set) $S = \{(x, y); x \in \mathcal{X}, y \in \mathcal{Y}\}$. A learning algorithm A (Learner) makes use of the training data to infer h from a space of functions $\mathcal{H}$. Provided with an input (x), the function should be able to predict correctly the desired output, even on unseen instances, in other words, it should be able to generalize over unseen situations. The desired function could have arbitrary output on the unseen instances, which means that if we are allowed to chose any function without restrictions on the function space $\mathcal{H}$, we can always find many functions which perform equally good on the training set but have different output on unseen instances. Thus, we cannot know if we have learned a good function or not. In such case, we will easily be able to fit perfectly the training set with a function which might perform badly on the unseen instances, this phenomena is called *overfitting*. So, making such restrictions is necessary to perform learning. In the terminology of machine learning they are called *Inductive Bias* Mitchell [1980]. On the other hand, the more the learner is biased the less variance it has, and so it might not be able to learn a function which fits the training set, in this case we are faced with *underfitting*. Figure 2.1 shows a dummy example of classification problem learned with different function families, it explains the tradeoff to be considered between learning complex functions (variance) and learning simple functions (bias). This tradeoff is a crucial issue for learning. Generally speaking, supervised problems share three main steps:

1. *Preprocessing*: define the input space $\mathcal{X}$, then project the examples in this space. Depending on the learning methods, different kind of representations may be used: vectors, attribute-value lists, first-order logic, relational representations and so forth. The statistical machine learning methods we use in this thesis mainly use vectorial representations $\mathcal{X} \subset R^d$ which are functions that map input objects to vectors. The components of the vectors are

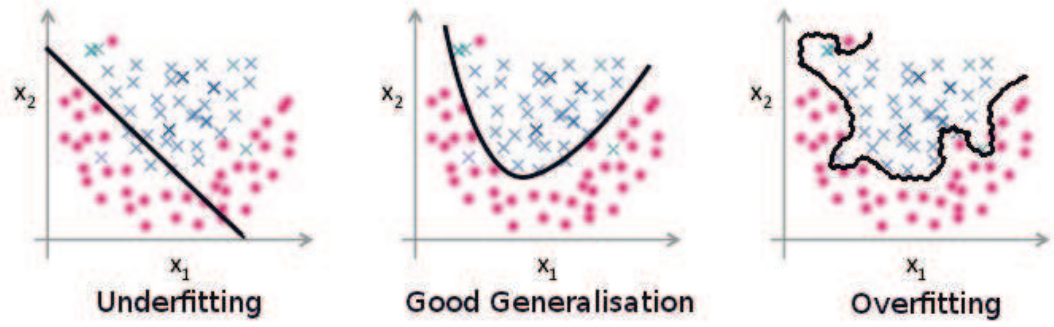


Figure 2.1: **Overfitting and Underfitting in binary classification.** From the left to the right we allow more and more complex functions to be learned. Left: the function family is too simple to fit the training data. Right: the function family is too complex, the learned function is tailored to the training data. Middle: a good compromise between underfitting and overfitting.

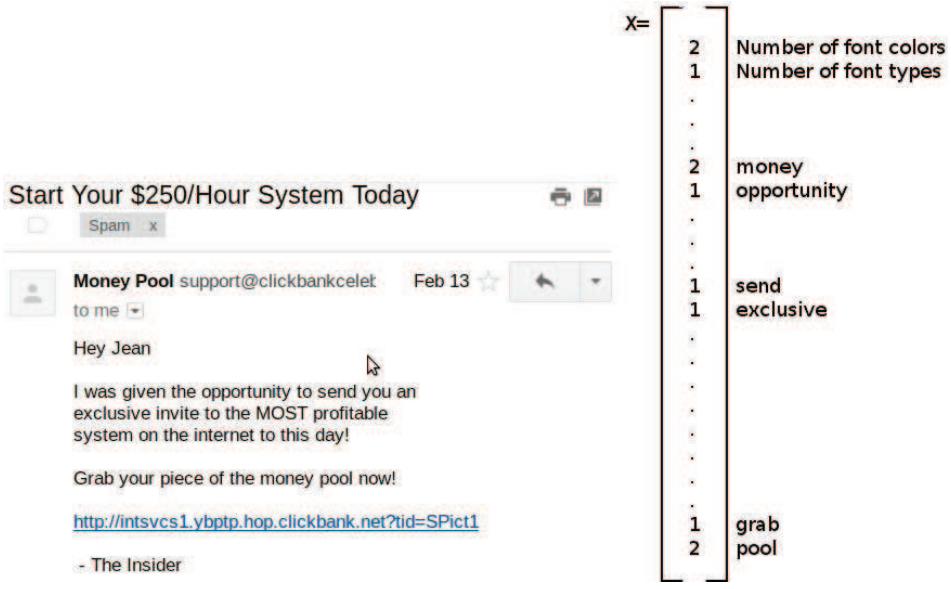


Figure 2.2: **Vectorial representation of an email.** The image on the left is a screenshot of a spam email and on the right its corresponding feature vector (x) is presented.

often called features and may describe any aspect of the input objects. For example, when dealing with textual documents, a common representation is to have one feature per possible word whose value is the word frequency (*bag-of-words*). Figure 2.2 illustrates an eventual vectorial representation for textual emails.

Choosing the appropriate features for a given learning problem is a difficult task and usually requires expertise of the domain. From a practical point of view, it is often observed that the quality of learning crucially depends on the choice of features. There must be enough features to accurately describe the input objects, but not too many features, since it may lead to costly and noisy learning.

2. *Training*: learn a mapping function h, which is able to predict (almost) correctly the label of a given example x. Thus, the learned function is expected to predict the labels on unseen examples with minimum number of mistakes. In Section 2.1.1 we talk formally about training within the statistical learning theory scope.

3. *Prediction*: use the learned function to predict the labels of unlabeled inputs. Prediction depends on the learning method, in some methods it is deterministic (*e.g.* decision trees), in others it is stochastic (*e.g. Markov Chains Monte Carlo*). Prediction cost also varies according to the method used, in methods like *KNN (Nearest Neighbor)* the whole computational complexity lies at prediction time rather than training time. Whereas, in methods like decision trees the training part is the most computationally demanding.

2.1.1 Theoretical Framework

Supervised learning problems can be formalized within the framework provided by *Statistical Learning Theory* Valiant [1984]; Vapnik [1999]. This framework

is based on the principle of risk minimization. A learning problem is defined through a joint distribution over input and output $\mathcal{D}_{\mathcal{X}\times\mathcal{Y}}$ and a loss function $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$. $l(y, \hat{y})$ quantifies how bad it is to predict $\hat{y} = h(x)$ instead of y.

Expected Risk (True Risk) The expectation of loss over the distribution $\mathcal{D}_{\mathcal{X}\times\mathcal{Y}}$ is called the *expected risk* and it is defined by:

$$R(h) = \mathbb{E}_{D_{\mathcal{X}\times\mathcal{Y}}}\{l(h(x), y)\} \tag{2.1}$$

Given a function space $\mathcal{H}$, supervised learning is the problem of selection the function $h \in \mathcal{H}$ that minimizes the expected risk:

$$h^* = \underset{h\in\mathcal{H}}{\operatorname{argmin}}\, R(h) \tag{2.2}$$

Depending on the application various loss functions may be defined. The simplest one is the so called $0\backslash 1$ loss, which is equal to zero if the prediction is correct and one otherwise:

$$l^{0\backslash 1}(y, \hat{y}) = \mathbf{1}\{y \neq \hat{y}\} \tag{2.3}$$

where $\mathbf{1}\{a\}$ is the indicator function whose value is one if a is true and zero otherwise. For classification problems with $0\backslash 1$ loss, the expected risk minimization can be rewritten in the following way:

$$\begin{aligned} \underset{h\in\mathcal{H}}{\operatorname{argmin}}\, R(h) &= \underset{h\in\mathcal{H}}{\operatorname{argmin}}\, \mathbb{E}_{D_{\mathcal{X}\times\mathcal{Y}}}\{l(h(x), y)\} \\ &= \underset{h\in\mathcal{H}}{\operatorname{argmin}}\, \mathbb{E}_{D_{\mathcal{X}\times\mathcal{Y}}}\{\mathbf{1}\{h(x) \neq y\}\} \\ &= \underset{h\in\mathcal{H}}{\operatorname{argmax}}\, P[h(x) = y | (x, y) \sim \mathcal{D}_{\mathcal{X}\times\mathcal{Y}}] \end{aligned} \tag{2.4}$$

i.e. the best function is the one maximizing the probability of classifying examples correctly.

Empirical Risk Since the distribution $\mathcal{D}_{\mathcal{X}\times\mathcal{Y}}$ is unknown, so the expected risk cannot be computed. However, we usually have a set of training examples $S = \{(x_i, y_i); i \in 1 \dots n\}\}$. This set is assumed to be independently and identically drawn *i.i.d.* (independently and identically distributed) from $\mathcal{D}_{\mathcal{X}\times\mathcal{Y}}$. The expected risk can be then approximated with the *empirical risk* on the training set:

$$\hat{R}(h) = \frac{1}{n}\sum_{i=1}^{n} l(h(x_i), y_i) \tag{2.5}$$

Selecting the function h^* that minimizes the empirical risk is known as the principle of *empirical risk minimization*:

$$h^* = \underset{h\in\mathcal{H}}{\operatorname{argmin}}\, \hat{R}(h) \approx \underset{h\in\mathcal{H}}{\operatorname{argmin}}\, R(h) \tag{2.6}$$

Statistical learning theory studies the relation between empirical risk minimization and excepted risk minimization, it defines the assumptions under which minimizing the empirical risk ensures -with certain probability- minimizing the true risk.

Structured (Regularized) Risk Generalization is crucial for machine learning. In order to improve it, one should avoid overfitting. A way to control overfitting is inspired from the *Ockham*'s razor principle. This principle is cited as *All other things being equal, the simplest solution is the best.* In other words, among the functions h whose empirical risks are not significantly different, we should choose the simplest one. One widely used way to induce such a function simplicity is to add a *regularization term* $\Omega(.)$. It is a function returns high scores for complex models and low scores for simple ones. Learning then aims at finding a function, which is a good compromise between small empirical risk and simplicity. Finding such a function is known as the *structured empirical risk minimization* principle.

$$h^* = \underset{h\in\mathcal{H}}{\operatorname{argmin}}\, \hat{R}(h) + \lambda\Omega(h) \tag{2.7}$$

where λ is a parameter that gives the control on the tradeoff between the minimization of the empirical risk and the minimization of the regularizer.

Loss Functions The 0\1 loss function might be hard to optimize directly, because it is not continuous and thus not derivable either. In practice many methods minimize an alternative loss that has better mathematical properties. Those alternative loss functions are often upper bounds of the original loss. In figure 2.3, we find the plots of some of the most common loss functions Maes [2009]. Following is a brief description of those functions:

- The *Perceptron loss* penalizes errors linearly *w.r.t.* socres. For examples, if the correct class is $y = +1$, a score of -3 leads to a penalty of 3 and a score of -1 leads to a penalty of 1. The perceptron loss corresponds to the problem solved by the early Rosenblatt's Perceptron algorithm (1957) and it is defined as follows:

$$l(m) = \begin{cases} -m & \text{if } m \leq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

  where $m = h(x).y$ a score reflecting the confidence of the prediction. It is positive in case of correct prediction and negative otherwise.

- The *Large-margin (hinge) loss* enforces a maximum *margin* between the positive examples and the negative examples. Enforcing such a margin leads to stronger theoretical guarantees Vapnik [1999]. This loss is optimized by the well-known Support Vector Machines Cortes and Vapnik [1995] and it is defined as follows:

$$l(m) = \begin{cases} 1 - m & \text{if } m \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

- The *Exponential loss* penalizes errors exponentially *w.r.t.* the negative scores.

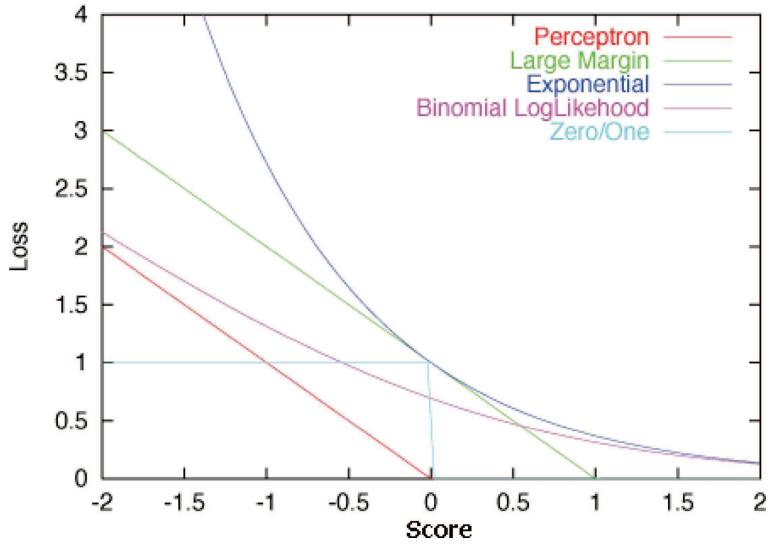


Figure 2.3: **Common loss functions used to upper bound the $0\backslash 1$ loss.** Each curve represents a loss function for binary classification, such a function depends on the score $y.h(x)$ that should be positive.

Boosting Schapire and Singer [1999] can be seen as a solution of this risk minimization problem corresponding to exponential loss. This loss is defined as follows:

$$l(m) = \exp(-m) \tag{2.10}$$

- The *Log-binomial loss* can be thought as a continuously derivable approximation fo the large-margin loss. This loss has some strong theoretical motivations and is minimized by maximum entropy classifiers Guiasu and Shenitzer [1985]. It is defined as follows for binary classification:

$$l(m) = \log(1 + \exp(-m)) \tag{2.11}$$

Figure 2.4 shows how various losses penalize the training examples on a binary classification task. The $0\backslash 1$ loss penalizes all the errors in the same way, whereas the other loss functions take the score into account: the further examples are from

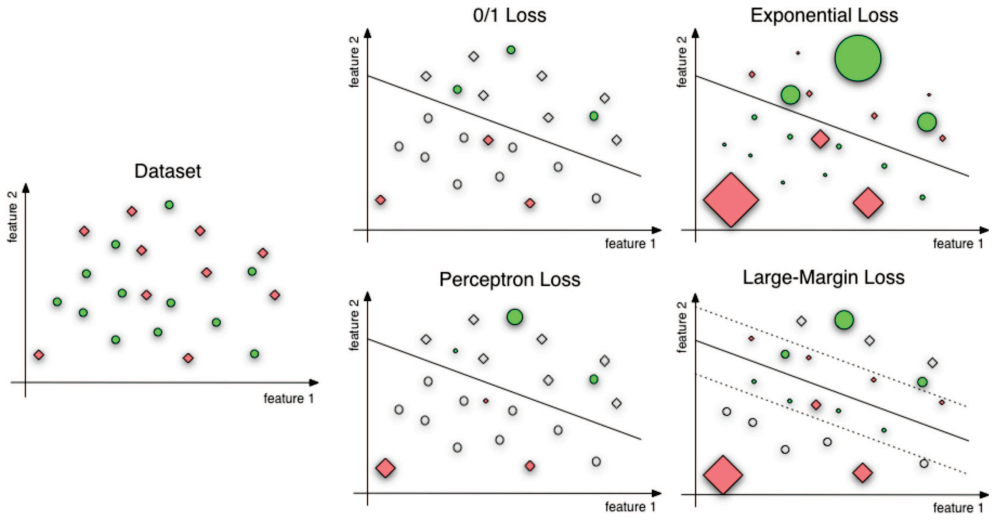


Figure 2.4: **Losses illustrated on a binary classification task.** Empty shapes correspond to examples with a 0 loss. Otherwise, the size of the filled shapes reflects the amount of loss the examples suffer from. In this example, ideally, all the green circle examples should be below the line and the red square examples should be above

being correctly predicted, the more loss they suffer from. The exponential loss increases exponentially in function of the score, whereas the Perceptron and large-margin losses only grow linearly. As soon as the examples are correctly classified in the Perceptron, their loss becomes null. Instead, the large-margin loss considers that an example that is near from the separator should be considered as an error.

What Loss Function is Suitable ? Let us explain this point through a simple example. Suppose we want to fit a function for predicting if it will rain or not. The input x will be the sky: `CLEAR`, `CLOUDY`, or `MIXED`. The output y will be either, `RAIN` (when it rains) or `NO` (when it does not). The loss is a function $l : \{RAIN, NO\}^2 \rightarrow \mathbb{R}$. It is not only the mathematical properties of the loss function which matter, in addition, the loss function depends on the priorities of the user. For example, if you hate getting wet, but you do not mind carrying an umbrella on a clear day, you might use a loss function like the one in Table 2.1.

$y/h(x)$	RAIN	NO
RAIN	0	1
NO	10	0

Table 2.1: The loss function is case you hate to be wet more than carrying an umbrella in a clear day

$y/h(x)$	RAIN	NO
RAIN	0	1
NO	1	0

Table 2.2: The loss function is case you hate to carry an umbrella when it is not needed

However, someone who usually carries a lot of things, he might not like to carry an umbrella when it is not needed, so he may use a loss function as the one in Table 2.2. For a given distribution $\mathcal{D}$, those two losses will yield in different learned functions, each of which suits the user defined priorities.

How to ensure (theoretically) a good learning ? The generalization is the key factor to evaluate the efficiency of a learning algorithm. Generalization is when the algorithm does not only minimize the empirical risk, but it does in addition minimize the true risk. In this context the following factors are crucial:

- *Consistency*: for any given function h, as we get more and more data, we should expect that the empirical risk tends to the true risk.
- The loss function: as explained previously, the mathematical properties as we well as the priorities of the user should be taken into consideration when we choose the loss function. In addition to that, the loss function depends also on the nature of the algorithm, some algorithms are designed to work well with certain losses (SVM with hinge loss, boosting with exponential loss, etc.).
- The function class $\mathcal{H}$. Roughly speaking, if the size of $\mathcal{H}$ is large, and the

functions are complex, the approximation would be worsened (overfitting). On the other hand, if we choose $\mathcal{H}$ to be small with simple functions, we would worsen the value of the minimum true risk (underfitting).

In the following section we will present a fundamental work in the theory of statistical machine learning which answers the previous issues.

2.1.1.1 PAC Learning in a Nutshell

The work of this thesis is done within the PAC (*Probabilistically and Approximately Correct*) learning framework Valiant [1984]. As mentioned before, studying the relation between the true and empirical risk is central in statistical learning. The goal is always to minimize the true risk (generalization), which is practically not possible since we have only an *i.i.d.* sample of the data (training data). With this sample we want to minimize the empirical risk while ensuring that by doing so, we also minimize the true risk.

PAC learning formed a fundamental brick in the statistical learning theory by introducing the computational complexity theory concepts to machine learning, those concepts are used to define the *PAC-Learnable* notion as follows.

We have a concept class C defined on an input space $\mathcal{X}$ -with dimension n- and an output space $\mathcal{Y}$, a distribution over $\mathcal{X} \times \mathcal{Y}$, an *i.i.d.* training sample S with length m, and a learning algorithm L which uses a function class $\mathcal{H}$. We will first give two basic definitions.

For a concept $c \in C$ and parameters ϵ *s.t.* $0 < \epsilon < 1/2$ and δ *s.t.* $0 < \delta < 1/2$. A function $h \in \mathcal{H}$ learned by L on S is called:

- *approximately correct* if the true risk $R(h)$ is smaller than ϵ:

$$R(h) < \epsilon$$

- *probabilistically correct* if the probability P of being approximately correct

is at least $(1-\delta)$:

$$P[R(h) < \epsilon] \geq 1-\delta \Leftrightarrow P[R(h) \geq \epsilon] < \delta$$

Finally, C is PAC-Learnable, if for all $c \in C$, distributions $\mathcal{D}$, ϵ, δ, it exists a learner L able to learn a probabilistically and approximately correct function $h \in \mathcal{H}$ with a polynomial time [1] in $|C|$, $1/\epsilon$, $1/\delta$, n.

To derive the bounds from PAC learning, we first formalize the consistency by an approximate upper bound on the true risk, *i.e.*

$$R(h) \leq \hat{R}(h) + \epsilon. \tag{2.12}$$

We start form the special case where $\hat{R}(h) = 0 \Rightarrow R(h) \leq \epsilon$. We want to calculate the probability that learning is not feasible (not PAC-Learnable), in other words, the probability that there exists a function $h \in \mathcal{H}$ which has null empirical risk, but its true risk $R(h) > \epsilon$. We call such a function a *dangerous function*. The probability that a certain function h makes an error on one training example is $R(h)$. Thus, the probability of not making an error is

$$1 - R(h) \leq 1 - \epsilon$$

The probability that h predicts perfectly all the m training examples:

$$\leq (1-\epsilon)^m$$

The probability that <u>at least one</u> $h \in \mathcal{H}$ predicts perfectly all the m training examples while having a true risk $> \epsilon$:

$$P[(\exists h \in \mathcal{H}) s.t. (\hat{R}(h) = 0) \wedge (R(h) > \epsilon)] \leq |\mathcal{H}|(1-\epsilon)^m$$

[1] Polynomial in the number of training examples and processing time per example

$$\leq |\mathcal{H}|e^{-m\epsilon}$$

$$|\mathcal{H}|e^{-m\epsilon} \leq \delta$$

$$m \geq \frac{1}{\epsilon}[\ln|\mathcal{H}| + \ln\frac{1}{\delta}]$$

This is a lower bound on the number of examples needed to make the risk of ending up with a dangerous function less than δ. This bound depends on the size of the function class $|\mathcal{H}|$, if this size grows linearly with the dimensionality of the data the concept class is PAC-Learnable. However, if $|\mathcal{H}|$ grows exponentially with n, the concept class is not PAC-Learnable.

For the more general case ($\hat{R}(h) \neq 0$), we can derive the bounds using Hoeffding's inequality:

$$P[(\exists h \in \mathcal{H}) s.t. (R(h) > \epsilon + \hat{R}(h))] \leq |\mathcal{H}|e^{-2m\epsilon^2}$$

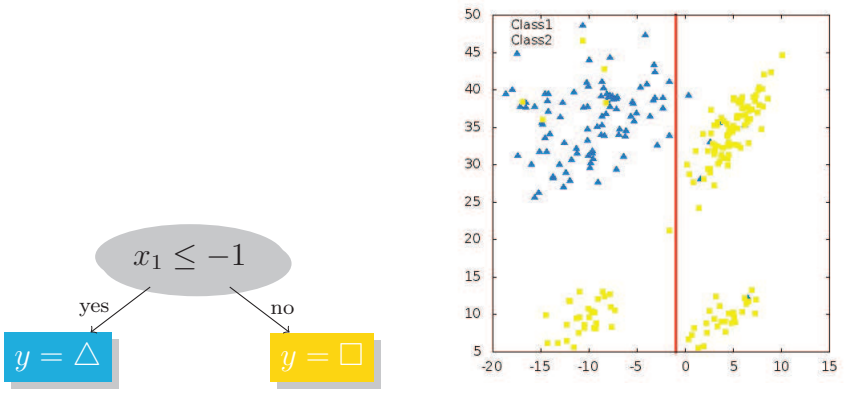
The lower bound on the number of examples:

$$m \geq \frac{1}{2\epsilon^2}[\ln|\mathcal{H}| + \ln\frac{1}{\delta}].$$

We finally deduce the following bound:

$$R(h) \leq \hat{R}(h) + \sqrt{\frac{\ln|\mathcal{H}| + \ln\frac{1}{\delta}}{2m}}$$

This bound tells that the more we have training examples, the tighter it is, and thus we get a better generalization. It also depends on the size of the function class $\mathcal{H}$. Smaller function class tightens the bounds but on the other hand limit the expressibility of the learned function. That is again, the compromise between variance and bias.



(a) The decision stump test and leafs (b) Four $2D$ mixture of Gaussian for a binary classification task

Figure 2.5: An example of a decision stump

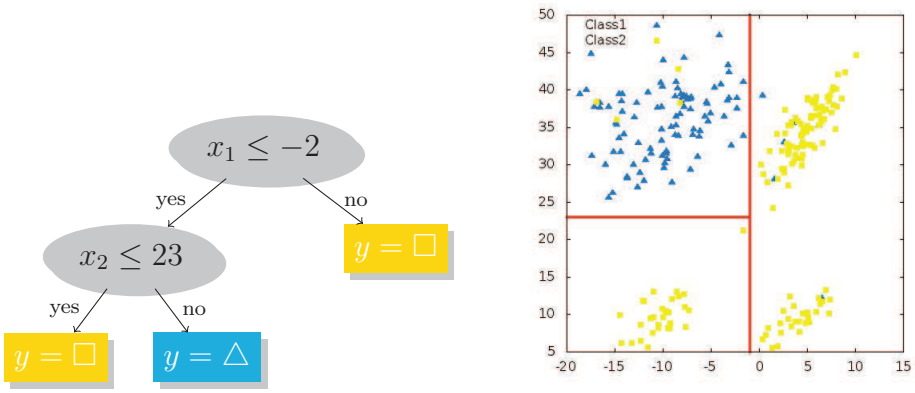
2.1.1.2 Weak Learner Principle

If a learning algorithm L on a concept class C cannot reduce the error to an arbitrary value ϵ close to zero, C cannot be said to be PAC-learnable by L. But, it can be studied from the *weak learning* angle. A concept class C is *PAC-weakly learnable* if: $\exists \gamma > 0, L$, and $\forall c \in C, \mathcal{D}, \delta > 0$, such that the learning algorithm L is able to learn a function h in a polynomial time in $\frac{1}{\delta}, |C|, n$ (input dimensions),such that:

$$P[R(h) \leq \frac{1}{2} - \gamma] \geq 1 - \delta.$$

The guaranty provided by weak learning is that the learned function will perform at least slightly better than random guessing. Clearly, this is not interesting since the goal of learning is to achieve minimum error rates. But, as we will show with more details in the next chapter, Kearns and Valiant [1994] proved that any PAC-weak learnable concept class is also PAC-learnable.

Two of the most widely used weak learners are decision stumps and decision trees. Figures 2.5 and 2.6 show examples of those learners on a dummy mixture of Gaussian data.



(a) A simple two levels decision tree

(b) Four $2D$ mixture of Gaussian for a binary classification task

Figure 2.6: An example of a decision tree

2.1.2 Supervised Learning Problems

In the literature, many different categories of supervised problems have been introduced. We will focus on such categories that are relevant for the presented work, namely *Binary Classification*, *Multi-Class Classification*, *Multi-Label Classification* and *Multi-Task Classification*. In the following, we will give the definitions of the such problems, real world examples and we will briefly list different approaches to cope with them.

2.1.2.1 Binary Classification

Binary classification is the oldest and most studied task for supervised learning. In this setting, the aim is to learn a function mapping $h : \mathcal{X} \to \mathcal{Y}$ from the input space to one of two possible outcomes. Conventionally, we denote the output space as $\mathcal{Y} = \{-1, +1\}$ (negative and positive labels). Real world examples of binary classification include: *medical diagnostic*, predicting if a patient has certain disease or not, *face recognition*, predicting if a given image contains a face or not, *spam filtering*, predicting if an email is spam or not. The theory of binary classification formed the basis on which other problems' approaches were build.

Even, some approaches propose to reduce other problems in a way or another to one or more binary classification problems.

2.1.2.2 Multi-Class Classification

In multi-class setting, the output space is a finite discrete set with cardinality superior to two, $|\mathcal{Y}| > 2$. The aim, is always to learn a function $H : \mathcal{X} \rightarrow \mathcal{Y}$ that minimizes the probability of the classification error on unseen instances $\text{argmin}_h \mathbb{E}_{(x,y)\sim D} Pr[h(x) \neq y]$. Examples of multi-class problems include: *objects recognition*, classifying images based on the objects they contain. *topic classification*, classifying textual documents by their topics, *speaker identification*, recognizing persons from their speech, etc.

We can group the existing methods for multi-class classification into three main categories: *a) problem transformation methods,* , *b) algorithm support methods* and *c) algorithm adaptation methods.*

- **Problem transformation methods:** are those which transform the multi-class classification problem into one or more binary classification problems, for which there exists plenty of learning algorithms. Examples: One-Against-One and One-Against-All binary classification.

- **Algorithm support methods:** are methods that support naturally multi-class classification. Examples: *Naive Bayes Classifier*, decision trees, *Hidden Markov Models*, etc.

- **Algorithm adaptation methods:** are methods that extend specific binary classification algorithms in order to handle multi-class data directly. Examples: *Adaboost.M1*, Multi-Class SVM where they learn a hyper plane for each class and the cost function maximizes the difference between the margin of the right class for a given instance and the maximum margin among other classes for this instance.

2.1.2.3 Multi-Label Classification

In multi-label setting, each instance x can have more than one label, thus the training set becomes: $S = \{(x, Y)|i \in \{1, \ldots, M\}, x_i \in \mathcal{X}, Y_i \subseteq \mathcal{Y}\}$ and the prediction function, $h : \mathcal{X} \rightarrow 2^{(\mathcal{Y})}$. A lot of objects (textual, visual, speech, etc.) could carry more than one label, for example a certain news article can be classified as political and economical. An image can be classified as sea and mountain.

There are different loss measures to minimize in multi-label setting, a common one is the hamming loss, $\frac{1}{k}\mathbb{E}_{(x,Y)\sim D}Pr[|h(x)\triangle Y|]$, where D is a distribution over observations (x, Y), and $\triangle$ is the hamming distance between the predicted set of labels and the target one. When the goal is to rank the predicted labels, the ranking loss is used:

$$\mathbb{E}_{(x,Y)\sim D}\Big[\frac{|\{(l_0, l_1) \in (\mathcal{Y} - Y) \times Y : f(x, l_1) \leq f(x, l_0)\}|}{|Y||\mathcal{Y} - Y|}\Big],$$

where $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$, is the function to be learned, and it give for each pair (x, l) a the confidence of giving x the label l.

We can group the existing methods for multi-label classification into two main categories: *a) problem transformation methods,* and *b) algorithm adaptation methods.*

Problem transformation methods They are the methods which transform the multi-label classification problem either into one or more single-label classification problems, where there exists plenty of learning algorithms. There are two common problem transformations in the literature, the first one considers each different set of labels that exist in the multi-label data set as a single label. Therefore, it learns a single-label multi-class classifier where each class is a set of labels. One drawback of this method is that it can result in a large number of classes and few examples per class. Yet another common problem transformation

method considers learning a binary classifier for each label. It transforms the multi-label data set into $|\mathcal{Y}|$ data sets, each label $y \in \mathcal{Y}$ will have it own data set where its examples will be labeled as positive if they contain y in the original data set and negative otherwise. The output of the final multi-label classifier is the union of positive labels predicted by all binary classifiers.

Algorithm adaptation methods They are methods that extend specific learning algorithms in order to handle multi-label data directly. A state of-the-art multi-label system is *Boostexter* Schapire and Singer [2000] which uses two algorithms *Adaboost.MH* and *Adaboost.MR*. Both algorithms are extensions of *Adaboost* Freund and Schapire [1996] for multi-label classification. They learn a function: $h : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$. In Adaboost.MH, for a pair (x, y), h output a positive (negative) number if y is predicted (not predicted) among the set of x's labels. The absolute value of the output is the confidence of the prediction. Practically, Boostexter uses an implicit reduction to binary classification, where each example (x, Y) in the training data set is transformed to $|\mathcal{Y}|$ examples of the form $((x, l), Y[l])$ for all $l \in \mathcal{Y}$, where $Y[l] = 1$ if $l \in Y$ and -1 otherwise. Many other algorithms based on SVM, KNN have been adapted to the multi-label setting.

2.1.2.4 Multi-Task Classification

Human being is able while learning a task to use knowledge induced from another task and vice versa. Which means that tasks are not learned independently. Inspired from this ability, learning multiple tasks by the same algorithm could enrich the learned model and / or save labeling efforts. Many examples can be cited here; Learning to rank documents for search engines in different geographical regions, learning to classify textual documents with two or more possible label sets, but they share a common knowledge that could be transfered from one to another. Learning shopping preferences of different users each one being a task.

Therefore, in multi-task setting, we have different learning tasks. They all share the same instance space $\mathcal{X}$. However, each task t, has its own label set $\mathcal{Y}_t$. In some problems, different tasks might share the same label set. The objective then is to solve the N classification tasks simultaneously. A multi-task classification algorithm will take as input a sample $S = \{(x_i, y_i, j_i) \mid x_i \in \mathcal{X}, y_i \in \mathcal{Y}_t, j_i \in \{1, \ldots, N\},\ 1 \leq i \leq m\}$. It should be noted that the same instance x can appear in a sample S with its label for different tasks. The goal is to find an hypothesis $h : \mathcal{X} \rightarrow Y_1 \times \ldots \times Y_N$ which minimizes error$(H) = Pr_{<x,y,j>\sim D}[H_j(x) \neq y]$, where $H_j(x)$ is the j-th component of $H(x)$ and $j \in \{1, \ldots, N\}$.

2.1.3 Problems Transformations

Supervised learning problems are not mutually exclusive, in other words, the same task can be modeled with several settings (multi-task, multi-label, ...). In the following, we present different possibilities of transforming a task from one setting to another.

2.1.3.1 Multi-Class $\leftrightarrow$ Multi-Label

A direct transformation from multi-label to multi-class is to consider each set of labels as a possible class. This method will result in a huge number of classes with few learning examples for each, which in turn, will not help in reaching a good predictive performance.

On the other hand, no additional work is needed to realize the opposite transformation (from multi-class to multi-label), because multi-label is more general than multi-class, so transforming multi-class problem to multi-label will not make the task easier. Nevertheless, if the multi-label algorithm gives a confidence degree for each label; we can by restricting its output to the label with the highest confidence, solve multi-class problems.

2.1.3.2 Multi-Class ↔ Multi-Task

Some MTL works have employed multi-class data sets to serve as benchmark beds for MTL. This has been done by transforming a multi-class dataset to several classification tasks and apply MTL approaches to learn them simultaneously. However, they don't claim that this transformation is done to improve over multi-class algorithms. Actually, the domain of multi-task learning is relatively new and there is not much off-the-shelf multi-task data sets, so the this kind of transformation is mainly done to simulate multi-task problems.

Such a transformation is simple, a multi-class problem over $\mathcal{X}$ with a label set $\mathcal{Y}$ is transformed to $|\mathcal{Y}|$ binary classification tasks, where for each label $y_t \in \mathcal{Y}$ will constitute task with the following label set $\mathcal{Y}_t = \{y_t, \neg y_t\}$. It should be noted that the transformation is not loose-less, since in multi-class setting each example has only on label $y \in \mathcal{Y}$, nevertheless, after the transformation an example can be labels by zero or more labels. For instance, if the output of all binary tasks were positive, it means that this example has all possible labels which was not possible in multi-class. But, as mentioned, this is not a serious issue as long as the purpose of the transformation is to simulate multi-task problems, in order to test the algorithms. Although this simulation can not result in real multi-task problems but it can give an idea about the performance of the algorithms.

The opposite way transformation (multi-task to multi-class) can be done by coding each combination of tasks' labels by a class, which means that the number of classes is exponential in the number of tasks. It is somehow similar to the transformation from multi-label to multi-class. We did not come across any work in the literature that uses this transformation, since it does not have an interesting application.

2.1.3.3 Multi-Label ↔ Multi-Task

Since multi-label and multi-task classification are the most similar, transforming problems from one to another would help comparing approaches from both

settings, and understanding when and why shall we use one or another.

From multi-label to multi-task the transformation is similar to the transformation from multi-class to multi-task. A multi-label problem over $\mathcal{X}$ with a label set $\mathcal{Y}$ is transformed to $|\mathcal{Y}|$ binary classification tasks, where for each label $y_t \in \mathcal{Y}$ there will be a task with the label set $\mathcal{Y}_t = \{y_t, \neg y_t\}$. Here both settings can give zero or more labels to each example and not like multi-class where each example has only one label.

The opposite way transformation (multi-task to multi-label), is feasible only when the tasks are all binary, in this case the transformation is done as follows. We first consider for each task one label as positive and the other as negative, then the set of labels for each example after the transformation (*i.e.*, in the multi-label setting) will be all the positive labels it has. For instance, if we have three tasks with label sets $\{a, b\}$, $\{c, d\}$, $\{e, f\}$ and $\{g, h\}$, we consider the first label of each task as positive. For an input example x, which has labels only for the first three tasks (a, d and e), the transformation will give an example with two labels a and c because the label d is chosen as a negative label and x does not have a label for the last task. Clearly, the transformation here is not agnostic toward the choice positive and negative labels for all tasks.

However, if the tasks are not binary there is no more the positive and negative notion which has been translated to appearance and disappearance of labels in the multi-label setting, thus, the transformation is not feasible. Which makes multi-task setting more general than multi-label.

2.1.4 Summary

In this section, we briefly introduced supervised learning. Which is the process of inducing -from a sample of input \ output pairs- a mapping function. Binary classification is the name of supervised problems with only two possible outputs, when the number of outputs is higher than two but it is still discrete, the problem is called multi-class classification. However, multi-label classification is the name

given to problems in which each input instance can have more than one possible output Multi-task learning is in general sense the name given to problems in which we want to learn multiple supervised learning tasks simultaneously in order to boost their predictive performance.

Supervised learning tasks can be formalized according to the principle of expected risk minimization: the function we are searching for is the function that leads to the lowest expected value of a task-specific loss function. Computing the expected risk is not possible, since we do not have access to the distribution underlying the learning problem. Instead, given a limited amount of training examples that are sampled from this distribution, the key idea is to approximate the expected risk with the empirical risk computed over the training set. When minimizing the empirical risk, a common phenomenon is called overfitting. It happens when the learned function is too tailored to the particularities of the training examples. In order to control overfitting, a common approach is to introduce a regularization term that gives a preference for simple models.

Some theoretical guarantees are needed to ensure that minimizing the empirical risk is consistent, which means that in the limits of infinite amount of data, minimizing the empirical risk will be the same as miminizing the true risk. In this context, we briefly came across the PAC learning framework, since it is fundamental for the statistical learning theory..

Different supervised learning tasks (binary, multi-class, multi-label and multi-task) are not mutually exclusive, therefore, we presented how one can reformalize a task to transform it from one setting to another.

In the coming section we will introduce multi-task learning from different scope, namely *Transfer Learning*. Multi-Task learning is categorized as a special case of transfer learning, and thus some approaches in the literature are inspired from transfer learning approaches.

2.2 Transfer Learning

Transfer Learning (TL) is the ability of an agent to transfer the knowledge acquired from one or more already learned tasks *(source tasks)* to a new one *(target task)*. For instance, learning to ride a bike might help in learning to ride a motorcycle. Similarly, learning to recognize apples might help to recognize tomatoes. Figure 2.7 illustrates the abstract process of transfer learning.

The topic of transfer learning in machine learning is motivated by the inherent ability of humans to use previously learned knowledge for the sake of learning new tasks more efficiently. In machine learning terms, "efficiently" can mean faster, better and / or cheaper. Faster in terms of computational complexity, better in terms of predictive performance and cheaper in terms of human annotation effort that is usually costly and not always available.

The short history of TL in machine learning has been initiated in a *NIPS-95* workshop on "Learning to Learn" [1], which focused on the need machine learning methods capable of reusing previously learned knowledge. Since then, works on transfer learning have been introduced with different names like learning to learn, life-long learning, knowledge transfer, inductive transfer, multi-task learning, knowledge consolidation, context-sensitive learning, knowledge-based inductive bias and incremental/cumulative learning Pan and Yang [2008]. Among these, a closely related learning technique to transfer learning is the multi-task learning Caruana [1997]; it tries to learn multiple tasks simultaneously, while in transfer learning the focus is on improving the performance of the target task by using the source one(s), in multi-task learning the transfer is multi-directional. Thus the asymmetry between tasks as source and target does not exist. In Figure 2.8 we show the information flow directions in both transfer and multi-task learning.

Transfer learning does not always assure an improvement of the target task,

[1] http://socrates.acadiau.ca/courses/comp/dsilver/NIPS95_LTL/transfer.workshop.1995.html.

Figure 2.7: **Transfer learning.** Is machine learning with an additional source of information apart from the standard training data, i.e., knowledge from one or more related tasks.

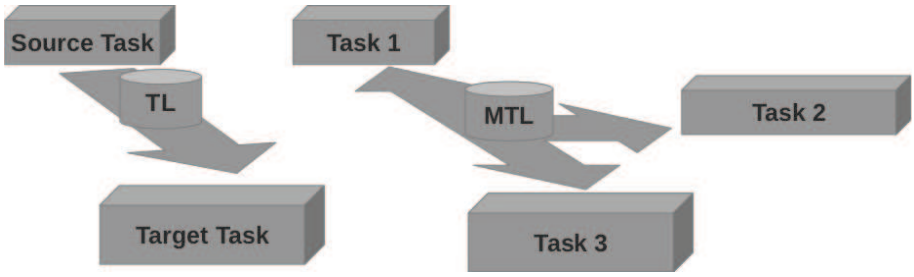


Figure 2.8: **Information flow.** In transfer learning, the information flow in one direction only. In multi-task learning, information can flow freely among all tasks.

it could be neutral or even it could hurt the performance, in such case we call it *negative transfer*. A major challenge in developing transfer methods is to produce positive transfer between related tasks while avoiding negative transfer Torrey [2009].

In this section, we give an overview of transfer learning. There has been a large amount of work on transfer learning for reinforcement learning in the machine learning literature. Nevertheless, these works go beyond the scope of this thesis. The rest of the section is organized as follows: we start by some notation and definitions. We then give a unified definition of transfer learning and categorize transfer learning into three different settings (Figure 2.9). For each setting, we review different approaches (Table 2.3). Finally, we come across the topic of negative transfer before concluding this chapter.

2.2.1 Notation and Definitions

Before giving the definition of transfer learning, we start by defining the most basic concepts in its literature, namely, the *domain* and the *task*. A domain D is the context in which a learning task is defined and it constitutes of two components: an input (feature) space $\mathcal{X}$ and a marginal probability distribution $\mathcal{D}^x$ that defines the sampling probability from $\mathcal{X}$. Two domains are considered different if they have different input spaces or different distributions. We define a task T in a given domain $D = \{\mathcal{X}, \mathcal{D}\}$ by two components: a label space $\mathcal{Y}$ and a predictive function $h : \mathcal{X} \rightarrow \mathcal{Y}$ to be learned from a training data $S = \{(x_i, y_i); 1 \leq i \leq n\}$. Most works in the literature consider the common case where we have we have one source domain D_S and one target domain D_T. A unified definition of transfer learning is given in Pan and Yang [2008] as follows:

Definition 2.1 (**Transfer Learning**). *Given a source domain D_S, a source task T_S, a target domain D_T and a target task T_T, transfer learning is a learning approach whose goal is to improve the learning of the target task using knowledge from the source task and domain, where $D_S \neq D_T$ and / or $T_S \neq T_T$*

The first condition $D_S = \{\mathcal{X}, D_S^x\} \neq D_T = \{\mathcal{X}_\mathcal{T}, D_T^x\}$ implies: 1) the input spaces are different. In text documents classification it could correspond to the case of two different languages in the source and target. 2) the distributions are different, in the same example it could correspond to the case of temporal shift between the source and the target, which induced a change in the distribution. The second condition $T_S = \{\mathcal{Y}_\mathcal{T}, h_S(.)\} \neq T_T = \{\mathcal{Y}_T, h_t(.)\}$ implies: 1) the label spaces are different; classifying text documents with different labels for the source and the target. 2) the prediction functions are different which correspond to different classification borders between classes.

2.2.2 Categories and Approaches of Transfer Learning

Traditional machine learning methods can be categorized under different settings based on the availability of the labels (teacher); supervised, semisupervised, transductive and unsupervised are the most common categories. Similarly, in the literature of transfer learning, different methods are categorized under three main categories based on availability of labeled data in source and target tasks. Those categories are: *inductive, transductive* and *unsupervised* transfer learning. Figure 2.9 shows a hierarchy of categories of TL methods.

In the inductive transfer learning setting, the target task is different from the source task, no matter whether the source and target domains are the same or not. For this reason, some labeled data are needed in the target to induce the predictive function h_T. Moreover, according to whether we have or have not labeled data in the source domain we categorize inductive transfer setting further into two subcategories: When no labeled data are available in the source domain, inductive transfer is similar to *self-taught* learning, which is proposed by Raina et al. Raina et al. [2007]. However, when we have source labeled data, inductive transfer is similar to multi-task learning, with one difference. In multi-task, tasks are symmetric according to the learning interest, whereas in transfer learning, the goal is to improve the performance of the target task.

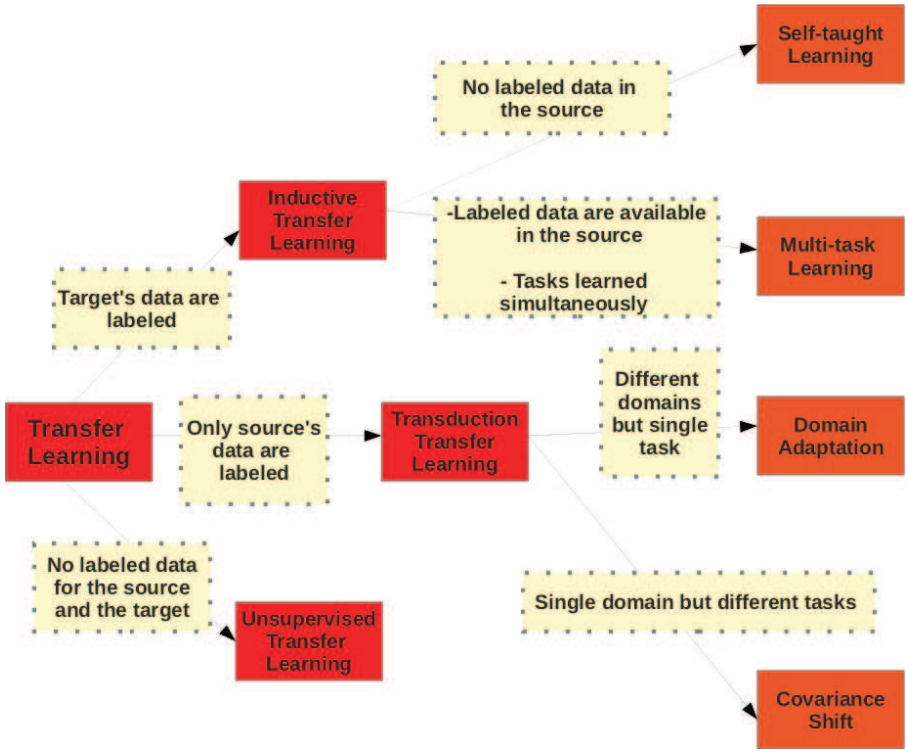


Figure 2.9: **TL Categories.** Based on the availability of labeled data in the source and target tasks, different categories are defined.

In the transductive transfer learning setting, we have no target labeled data, but a lot of source labeled data. Also, transductive transfer is divided to two subcategories, based on the situation of the source and target domains: if the domains are different, transductive transfer corresponds to the setting of *domain adaptation* Daumé and Marcu [2006]. However, if the domains are identical but the tasks are different we are in the covariance shift setting Shimodaira [2000].

Finally, unsupervised transfer learning happens when we do not have labeled data neither for the source nor for the target tasks. However, unsupervised transfer learning focus on solving unsupervised learning tasks in the target domain, such as clustering, dimensionality reduction, and density estimation Dai et al. [2008]; Wang et al. [2008]. In the next section, we present different approaches to cope with transfer learning problems.

Transfer Learning Approaches	Description
Instance Transfer	Re-weight some labeled data in the source to use in the target domain Bickel et al. [2007]; Huang et al. [2007]; Jiang and Zhai [2007].
Feature Transfer	Find the good features that reduce the difference between the source and the target Argyriou et al. [2006b, 2007]; Jebara [2004].
Parameter Transfer	Learn shared parameters between source and target, which can boost the target's performance Bonilla et al. [2008]; Gao et al. [2008]; Lawrence and Platt [2004].
Relational Knowledge Transfer	Build a relational knowledge mapping between the source and the target Davis and Domingos [2009]; Mihalkova et al. [2007].

Table 2.3: Transfer Learning Approaches.

2.2.3 Transfer Learning Approaches

By answering the question "*What to transfer ?*", we can deduce different approaches to solve transfer learning problems. Table 2.3 shows four answers to that questions, each corresponds to an approach. The first can be referred to as instance-based transfer learning (or instance transfer) approach Bickel et al. [2007]; Huang et al. [2007]; Jiang and Zhai [2007]. It assumes that certain parts of the data in the source domain can be reused for learning in the target domain by using techniques like re-weighting and importance sampling. The second approach is called feature-transfer Argyriou et al. [2006b, 2007]; Jebara [2004]. It copes with the issue of learning a good feature representation for the target domain using the source domain. A third approach is parameter-transfer Bonilla et al. [2008]; Gao et al. [2008]; Lawrence and Platt [2004], it assumes that since tasks are related, we can learn shared parameters under the form of priors or hyper-parameters. Thus, the transfer of knowledge is done across the parameters. Finally, the last case can be referred to as the relational- knowledge-transfer problem, which deals with transfer learning for relational domains. The basic assumption behind this context is that some relationships among the data in the source and target domains are similar. Thus, the knowledge to be transferred is the relationships among the data. Recently, statistical relational learning techniques dominate this context Davis and Domingos [2009]; Mihalkova et al. [2007].

2.2.4 Negative Transfer

Negative transfer happens when a transfer learning method participates in reducing the performance of the target task, rather than improving it. Studying negative transfer and developing techniques to avoid it is crucial for the domain. But yet it is modestly studied and only few works have been introduced so far.

One way of approaching negative transfer is to attempt to ignore harmful source-task knowledge during learning. The goal here is to eliminate or minimize the amount of bad information transfer from the source to the target. In Rosenstein et al. [2005], the authors present an approach for detecting negative transfer in naive Bayes classification tasks. The basic idea is to learn a hyperprior for both source and target tasks, whose variance is proportional to the dissimilarity between the tasks.

Another approach in the literature is at the task level, instead of information level. It chooses among a pool of candidate source tasks the most related ones to the target task. An example of this approach is the work of Taylor et al. [2007]. It is based on the idea that transfer is more efficient when it is done from the easier tasks toward the more difficult task. In this regard, they propose a transfer hierarchy that orders tasks by difficulty. Similar approaches propose to cluster tasks into groups. The clustering criterion is tasks relatedness Bakker and Heskes [2003]; Ben-david and Schuller [2003]. Similar tasks will be clustered in the same group and then transfer will happen between the tasks of each group.

2.2.5 Summary

In this section, we gave an introduction to transfer learning as a machine learning approach inspired from human ability to share and transfer knowledge between different learning tasks. We then reviewed the three categories of transfer learning: inductive transfer learning, transductive transfer learning, and unsupervised transfer learning. Moreover, each category to transfer learning can be

classified into four contexts based on what to transfer in learning. They include the instance-transfer approach, the feature-representation-transfer approach, the parameter-transfer approach, and the relational-knowledge-transfer approach.

The majority of works in the literature make the assumption that the source and the target tasks are related, which means that applying the transfer will help positively. However, this assumption does not hold always, and when it does not, negative transfer occurs, i.e., transfer learning participates in decreasing the performance of the target task. Despite its importance, little amount of work has been done to help developing methods able to avoid negative transfer and ensure that transfer learning will not cause a decrease of the target task's performance. We presented two approaches to deal with negative transfer, one is on the information or data level, whereas the second is on the tasks level.

2.3 Related work

Multi-task learning (MTL); an approach to improve the inductive bias of related tasks by learning them simultaneously. One of the earliest works to coin this notion is Caruana [1993]. The same author gave in his thesis (Caruana [1997]) a broad vision of the perspective and potential of MTL. He defended the idea saying that related tasks share a common knowledge, thus learning those tasks jointly might enrich the representation of their common knowledge which in turns will have a positive effect on their performance. Caruana integrated multi-task learning approach with different machine learning algorithms, like artificial neural networks, K-nearest neighbors and others.

Since then, MTL has been gaining more and more interest in the community of statistical machine learning. Researchers from different backgrounds approached MTL by different methods. In this section, we review prior art MTL methods that are relevant to our work. We first start by the methods which model the shared knowledge on the parameters level, then we discuss couple of methods that

model the shared knowledge on the features level. As our work is done within the framework of boosting / ensemble learning, we will discuss two MTL algorithms that are done within the same framework. While going through different approaches we will be pointing out their different advantages and limitations.

Some limitations, which are critical for multi-task learning, are often not addressed or listed explicitly in the prior art methods. They are usually in the form of assumptions or restrictions that help in making the extention of an approach to MTL feasible. On the other hand, they limit the scope of multi-task problems that could be covered by such approaches. The limitations we will focus on are:

- **Shared label space:** Some methods restrict the problems they address to the ones in which the tasks share the same space of labels. Moreover, a subset of those methods limit the shared label space to binary labels.

- **Same Number of Class Labels:** Almost all prior-art methods even when they address multiple tasks with different label spaces, they do assume that the tasks have the same number fo class labels. To the best of our knowledge only one method (Parameswaran and Weinberger [2010]) does not impose such a restriction.

- **Share training examples:** A very limiting restriction is the one that requires all tasks to share the same training examples, differently said, each example should be labeled by one label per task. It is worth noting, that a method requiring shared binary label space and shared training examples can cover only the problems covered by multi-label classification algorithms.

- **Global relatedness:** Task relatedness is a key assumption to do multi-task. It is a vague assumption since there is no formal definition of it. As a consequence, it is usually defined implicitly by the algorithm, then it is used to learn the common knowledge between tasks. However, MTL algorithms that assume a certain task relatedness pattern (e.g.: labels correlation), they usually assume that this pattern is global across the whole learning

space. Unfortunately, the global relatedness assumption turns often to be too strong; it might even hurt the performance, similarly to introducing noise in data. This task relatedness may show up different degrees or even different signs in different regions of the "learning space". It is therefore important that the multi-task learner determines the relatedness of tasks, learns its different degrees and accommodates the inductive bias accordingly.

2.3.1 Parameters Transfer MTL

The approach based on kernel based learning with regularization is a common approach in the literature of MTL. The paper of Evgeniou and Pontil [2004] was the first to introduce an extension of existing kernel based learning methods for single task learning, such as Support Vector Machines (SVMs) to multi-task learning. In a subsequent work (Evgeniou et al. [2005]), the same authors generalized their first work by introducing the notion of *multi-task kernel*. The basic idea behind this work is that related tasks can be mapped to space in which they behave similarly. In other words, learning is done as in single task learning in the new space. A kernel defined in the new space is called multi-task kernel. The most simple instance of this approach is to learn a weight vector w_t for each task t and a common vector for all tasks w_0. The final weight vector for t is then $w_t + w_0$. This case is shown in the equation below as a convex optimization problem for T tasks each having m_t training examples. All tasks share the same label space $\mathcal{Y}$:

$$min_{w_0,w_t,\xi_{t,i}} \left\{ \sum_{t=1}^{T} \sum_{i=1}^{m_t} \xi_{t,i} + \frac{\lambda_1}{T} \sum_{t=1}^{T} ||w_t||^2 + \lambda_2 ||w_0||^2 \right\}$$

subject to the constraints: $\forall t \in \{1 \dots t\}, i \in \{1 \dots m_t\}$

$$y_{t,i}(w_0 + w_t)x_{t,i} \geq 1 - \xi_{t,i}$$

$$\xi_{t,i} \geq 0$$

The work of Zheng et al. [2008] was also inspired from the multi-task kernel approach, but their contribution consists of learning the mapping of tasks to the space in which they behave similarly rather than predefining it.

Kernel based learning approach for MTL has been proved efficient and preferment on real world problems. But, in its current state, it suffers from two limitations, namely, binary shared label space, which narrows the scope of problems it can cover. And the global relatedness assumption. Actually, sharing the same labels prevent the tasks from being even negatively correlated ($y_1 = -1$ from the first task comes often with $y_2 = +1$ from the second).

Within the same framework of regularization, the method of tasks clustering proposed in Jacob and Bach [2008] aims at designing new regularization norms that will enforce that sharing of information between tasks. they defend the hypothesis that the different tasks are in fact clustered into different groups, and that the weight vectors of tasks within a group are similar to each other. Their method does not require the clusters' structure to be given, it can rather learn them. However, their method suffers from the same limitations as the previous methods. Shared label space and global relatedness approach. The later being relatively relaxed by clustering the tasks. But, it still cannot cope with situations where a pair of tasks have local correlation patterns which differ across the learning space.

The problem of learning clusters of tasks has been addressed by the Hierarchical Baysian learning framework. The assumption says that tasks parameters are generated from shared priors. The work proposed in Xue et al. [2007] relies on a Dirichlet process (DP) based statistical model to learn the extent of similarity between classification tasks. They consider two scenarios: first, a symmetric multi-task learning (SMTL) situation in which classifiers for multiple tasks are learned jointly. Second, they consider an asymmetric multi-task learning (AMTL) formulation in which the posterior density function from the SMTL model pa-

rameters (from previous tasks) is used as a prior for a new task: this approach has the advantage of not requiring storage and use of all previous data from prior tasks.

Staying with the very same framework of Hierarchal Bayesian learning; the authors of Liu et al. [2009] propose a formulation which encodes the information of each task inside the associated likelihood function, sparing the prior for exclusive use by the information from related tasks. In addition, the formulation lends itself to a Dirichlet process, allowing the tasks to share information in a complex manner. A key advantage of their work is the ability to learn from partly labeled data in a semi-supervised manner using label propagation by random walk technique.

Nevertheless, despite the interesting advantages granted by shared priors over tasks, some drawbacks are worth mentioning. Such a learning model relies on the tuning of hyperparameters as well as the performance of the inference / sampling algorithm. In addition, the models proposed above, assume that tasks share the same label space and a global relatedness pattern.

Yet another Bayesian work for multi-task regression and classification is proposed by Archembeau et al. [2011]. Their model is able to capture correlations between tasks, while being sparse in the features. They make use of novel group sparsity inducing priors based on matrix-variate Gaussian scale mixtures. The main advantage of their Bayesian formalism is that it enables to learn the degree of sparsity supported by the data and does not require the user to specify the type of penalization in advance. Despite the interesting empirical performance of their method, it is worth noting that it is pretty limited when it comes to the ability of covering multi-task problems. The main limitation is that the method is able to learn only binary tasks whose examples are exactly the same. However, multi-task normally brings the advantage of being able to learn multiple tasks coming from different sources (aka annotators). So, requiring the training examples to be the same across tasks implies having the same annotators for all tasks,

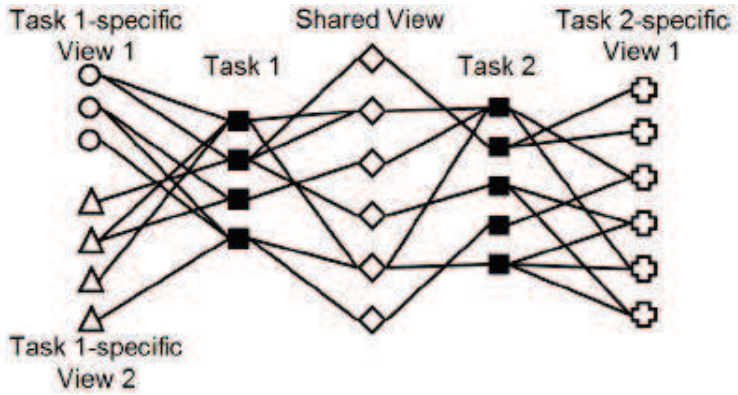


Figure 2.10: **Graph-based approach to multi-task.** Examples from Task 1 and Task 2 (black squares) have both a shared view (diamonds) and the task specific views (circles and triangles for the 2 views of Task 1, and pluses for the 1 view of task 2). The weight of an edge between an example node and a feature node is set to the feature value.

which is not the case in the majority of real problems.

Far from Baysian framework, the work of He and Lawrence [2011] models the problem of multi-task using graphs, in addition they take into account tasks which have more than one view each (e.g., the data of a task is photos with comments on them, the photo is a view and the text of the comment constitutes another view.).

Within each task, they construct a bi-partite graph for each view, modeling the relationship between the examples and the features in this view. The consistency among different views is obtained by requiring them to produce the same classification function, which is commonly used in multi-view learning. Across different tasks, they establish their relationship by imposing the similarity constraint on the common views. Figure 2.10 shows an example of their graph-based modeling for two tasks each with two views, one view being shared and the second is specific.

The method is interesting since it makes use of unlabeled examples during the training phase. But, unfortunately, the current formalism works only for binary

tasks, and it is computational complexity is quadratic in the number of tasks.

Coming back to the convex optimization / regularization framework we cite here the method of Parameswaran and Weinberger [2010]. The authors propose a a multi-task approach inspired from (Large Margin Nearest Neighbor *lmnn* Weinberger and Saul [2009]). Similar to svms, the solution of lmnn is also obtained through a convex optimization problem that maximizes a large margin between input vectors from different classes. However, instead of positioning a separating hyperplane, lmnn learns a Mahalanobis metric. One advantage that the kNN decision rule has over hyperplane classifiers is its agnosticism towards the number of class labels of a particular data set. A new test point is classified by the majority label of its k closest neighbors within a known training data set additional classes require no special treatment. Their algorithm learns one metric that is shared amongst all the tasks and one specific metric unique to each task.

This method is free of the limitations the previous cited methods suffer from. It does not require neither shared labels nor shared training examples. Also, there is no global relatedness assumption imposed on the learned model. Nevertheless, the method shares the same drawbacks of classical KNN: first, the curse of dimensionality; in fact, the notion of distances gets distorted in very large spaces. Second, the computational complexity at classification time: in order to classify a point we need to find its closet N neighbors, doing so (in the very classical version of KNN) requires the calculation of the distance between the point to classify and each point in the training data.

Another work which does not suffer from the limitations of shared examples and labels is the work of Novi Quadrianto [2010]. It relaxed these constraints by an iterative learning algorithm based on the duality between maximum entropy and MAP estimate. They model the relatedness between tasks by the mutual information among the label sets. On the other hand, their work relies on the hypothesis that the relation between tasks is the same in the whole learning space (global relatedness).

The algorithm's complexity is exponential with the number of tasks, however, the authors propose to relax the objective function so the complexity will be reduced to quadratic which could still cause a problem in the case of large number of tasks.

2.3.2 Features Transfer MTL

Tasks relatedness might be in the form of a common underlying representation. For example, in object recognition, the human visual system is organized in a way that all objects are represented using a common set of features learned.

The work proposed in Argyriou et al. [2006a, 2008] explored feature transfer approach for task relatedness, that is, they learn a low-dimensional representation which is shared across multiple related tasks. The method is based on the well known $L1$norm regularization which provides such a sparse representation for the single task case. However, they generalized this formulation to the multiple task case. Their method learns a few features common across the tasks by regularizing within the tasks while keeping them coupled to each other. Moreover, the method can be used, as a special case, to select (not learn) a few features from a prescribed set.

The first step of their algorithm consists of independently learning the parameters of the tasks regression or classification functions. The second step consists of learning, in an unsupervised way, a low-dimensional representation for these task parameters. The number of common features learned is controlled, by the regularization parameter, which means that a tuning effort is required, in addition to the requirement of a shared binary label space and the global relatedness assumption.

Another feature-level multi-task work –but in totally different context– is the work of Collobert and Weston [2008]. In their work they attempt to define a unified architecture for Natural Language Processing that learns features that are relevant to the tasks at hand given very limited prior knowledge. This is

achieved by training a deep neural network which is applied to many well known NLP tasks including part-of-speech tagging, chunking, named-entity recognition, learning a language model and the task of semantic role-labeling. All of these tasks are integrated into a single system which is trained jointly. As in single-task neural networks, their algorithm is sensitive to the design of the network which is an effortful task and it become even more demanding when it comes to multi-task setting.

2.3.3 Ensemble MTL

It is worth to note that boosting has been already used in multi-task learning for face verification (Wang et al. [2009]). Following ideas different from ours, it learns a set of boosted classifiers and is based on a probabilistic model where a multinomial variable indicates how much each boosted classifier contributes to each task. The learning algorithm involves Expectation-Maximization (EM) to learn both the multinomial random variables as well as the classifiers. The algorithm is intrinsically based on the idea that the tasks share the same labels, more specifically -1 and $+1$ labels.

Last, in Chapelle et al. [2010b], the authors provided a new boosting algorithm to capture tasks relatedness. Inspired from the common multi-task modeling assumption (Evgeniou and Pontil [2004]), the algorithm learns a specific model for each task in addition to one global model that capture the commonalities among them. However, they learn the models through gradient boosted regression, rather than SVMs quadratic programs. A worthy advantage of their algorithm is that it can learn tasks with different features as long as they share a subset of their features which will be used to the learn the global model. On the other hand, the algorithm requires the same labels to be shared across tasks.

2.3.4 Summary

In this section we reviewed a considerable part of the prior-art methods for MTL. We categorized those methods in two categories based on the transfer of knowledge approach they use (parameters or features transfer). As our contribution lies within the framework of ensemble methods, we also explored the previously done Ensemble MTL works in an explicit category.

The pros as well as the cons of the prior-art methods were discussed. We listed explicitly three limitations which we believe to be crucial for MTL methods but yet they have not been addressed as they deserve to be. Those limitations are: sharing the labels and / or examples between tasks, in addition to the assumption of global relatedness between tasks. Different methods cope with some of them but very few methods could cope with all of them. In the following chapter, we will be presenting novel classifiers which do not suffer from the above limitations. Those classifier are considered as weak classifiers. Thus, it is preferable to learn ensemble of them in order to achieve low classification error rates.

Chapter 3

Multi-Task Hypotheses

Contents

As described in the previous chapter, the majority of already existing MTL approaches are prone to one or more of the following limitations: shared examples, shared labels and global relatedness across the whole learning space. In this chapter we present two novel Multi-Task Hypotheses (classifiers) that do not suffer from those limitations. Therefore, they can cover a wider prospect of MTL problems.

3.1 Multi-Task Stump `2T-stump`

We generalize stumps for multi-task problems. Recall that stumps are one-level decision trees, i.e. a stump is defined by a test node and prediction nodes. For

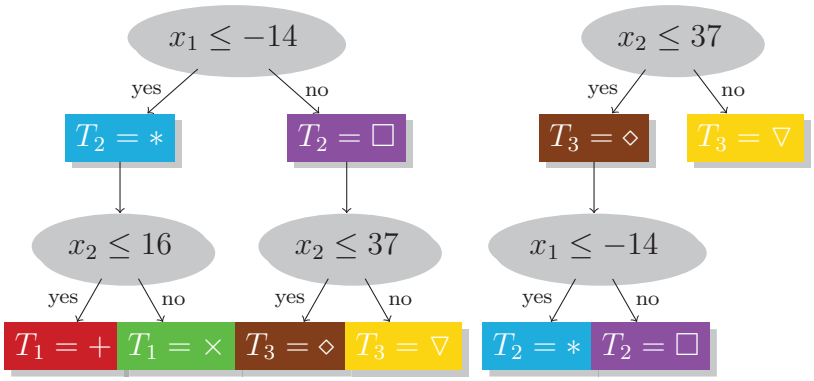
sake of clarity and since n-ary tests are a straightforward generalization of binary tests, we will consider only binary tests. Stumps can be used as weak classifiers and they allow to learn accurate strong classifiers when used in boosting algorithms Freund and Schapire [1996]. We first start by describing our Multi-Task Stumps for binary classification tasks, then we generalize them to multi-class multi-task setting.

3.1.1 `2T-stump` for Binary Tasks

For the multi-task setting, the weak classifiers we consider are 2-level decision stumps called two-task stumps (`2T-stump`). The first level of a `2T-stump` is a decision stump for one of the N tasks. At the second level, there are two decision stumps, one attached at each of the two prediction nodes. Each of these two decision stumps corresponds to one of the $N-1$ remaining tasks. An example is given in Figure 3.1.1.

For classifying a given instance x, the root test is considered and the instance x will descend to one of the two prediction nodes, which assigns a label for the first-level task. Then the second-level test is considered, the instance x will descend to one of the two prediction nodes and a label is assigned for the corresponding second-level task. It should be noted that the `2T-stump` will abstain on x for all other tasks. Examples are given in Figure 3.1.1. Formally, let us consider that the value 0 stands for abstention, a `2T-stump` defines an hypothesis $h : \mathcal{X} \rightarrow \{-1, 0, +1\}^N$, which gives for every instance x and every task j an output $h_j(x)$ in $\{-1, 0, +1\}$, where $h_j(x)$ is non zero for at most two tasks.

A `2T-stump` defines a partition of the instance space dependent on the tasks it contains. For instance, let us consider the `2T-stump` defined in Figure 3.1(a), it defines a partition of the instance space shown in Figure 3.2(a). The line with equation $x_1 = 14$ defines the first-level separator for task T_2 which allows to define two different separators for the tasks T_1 and T_3 in each of the two sub-spaces. Thus `2T-stumps` will allow to capture relations between tasks. One can consider



(a) This `2T-stump` has a stump for task T_2 at the first level, and stumps for tasks T_1 and T_3 at the second level. Consider an instance x such that $x_1 = -7$ and $x_2 = 4$ then the `2T-stump` assigns to x the class $\square$ for task T_2 and the class $\diamond$ for task T_3 while abstaining for other tasks

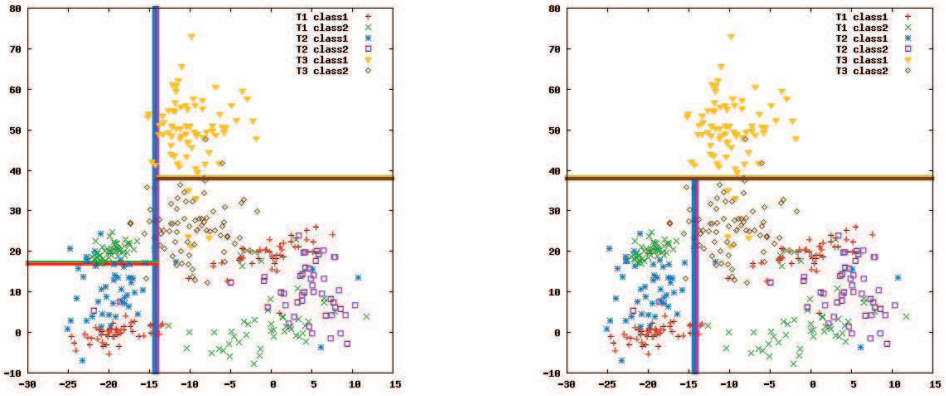
(b) This `2T-stump` has no right child. Consider an instance x such that $x_2 = 40$ then the `2T-stump` assigns to x the class ∇ for task T_3 while abstaining for other tasks

Figure 3.1: Two `2T-stumps`.

more than two levels but the number of such weak classifiers grows exponentially with the number of levels. This is why we consider `2T-stumps` with only two levels and we will show that combining `2T-stumps` in a boosting algorithm will allow to capture local dependencies between N tasks, and learn implicitly soft clusters of tasks. Actually, because each `2T-stump` has three nodes, it can predict for each example only on two tasks and abstain on the rest. Therefore, learning an a boosting ensemble of `2T-stumps`, where each task can appear in more than one `2T-stump`, is like clustering tasks in a soft way using the boosting mechanism.

3.1.2 `2T-stumps` for Multi-Class Tasks

For multi-class classification we use decision stumps similar to the ones presented in Schapire and Singer [2000]. The test of such a stump is the same as in an binary classification stump. Based only on the outcome of this test, the weak hypothesis (after learning) outputs predictions and confidences that each label is associated with the document, the label with the highest confidence for an input point x will



(a) Visualization of the partition defined by the `2T-stump` in Figure 3.1(a). It is learned on the synthetic dataset with algorithms `WL-Best-K` with $K = 5$ or `WL-Best-K` with $K = 25$. The score $W_- + \frac{1}{2}W_0$ is equal to 0.134

(b) Visualization of the partition defined by the `2T-stump` in Figure 3.1(b). It is learned on the synthetic dataset with a naive greedy algorithm. The score $W_- + \frac{1}{2}W_0$ is equal to 0.197

Figure 3.2: Visualization of `2T-stumps` learned on a synthetic dataset.

be considered as the class label for x. For example, in a text classification task. a possible term can be "The Big Bang Theory", and a learned stump might look as follows: If the term "The Big Bang Theory" appears in the document then predict that the document belongs to Physics with high confidence, to TV Series with medium confidence, and that to Sports with low confidence. If, on the other hand, the term does not appear in the document, then predict that it does not belong to any of the classes with low confidence.

As presented above, `2T-stumps` can classify tasks with different number of class labels and without any a priori assumption on the relatedness patterns of the tasks. As we will show in the next chapter; which is dedicated to the learning algorithms, learning an `2T-stump` does not require any shared examples between the tasks.

In the following, section, we will present yet another Multi-Task classifier which comes from the same spirit of decision tree classification but, it does not have the sequential behavior of `2T-stump`, it rather analyzes the data from all tasks at each node while building the tree.

3.2 Multi-Task Decision Tree MT-DT

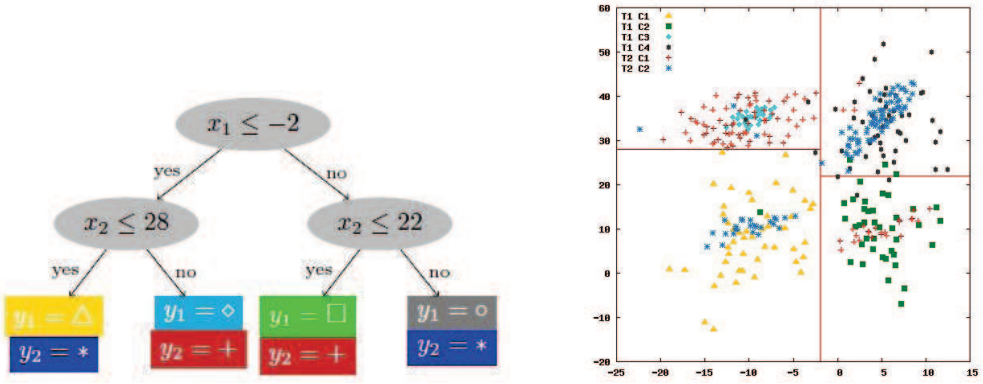
Decision tree learning is a well known technique in machine learning; it uses a decision tree as a predictive model which maps observations from the instance space to the target values. In the case of classification, tree leaves represent class labels and branches represent conjunctions of item attributes that lead to those class labels Quinlan [1993].

In this section we adapt decision trees to the multi-task setting. We propose a new multi-task classifier that we call multi-task decision tree (MT-DT). One obvious difference between one- and multi-task setting is in the tree structure. One-task decision tree uses the internal test nodes to guide the decision process while the final decision on assigning a label to a sample is made in a tree leaf.

The structure of an multi-task decision tree (MT-DT) is different in the way it guides the decision process for multiple tasks. This process is not necessarily the same for all tasks. An MT-DT can make a final decision for some tasks in an internal test node, not a tree leaf. This happens when the internal test node has enough information to classify an instance of a certain task T, in such a case a decision leaf with the appropriate classification decision for T is added to the tree and the learning proceeds with the remaining tasks.

Figure 3.3.a gives an example of an MT-DT learned for two synthetic tasks generated from 2D mixture of Gaussians (see Figure 3.3.b). T_1 has four labels ($\mathcal{Y}_1 = \{\square, \diamond, \triangle, \circ\}$) and T_2 has two labels ($\mathcal{Y}_2 = \{+, *\}$). Two labels of T_1 ($\square, \diamond$) are correlated with label $+$ of T_2, while two other labels of T_1 ($\triangle, \circ$) are correlated with label $*$ of T_1. The generated MT-DT has three internal test nodes and each decision leaf carries one rule per task.

Another example of MT-DT is showed in Figure 3.4. Task T_1 is the same as Figure 3.3, while task T_2 is generated differently from a mixture of Gaussians (see Figure 3.4.b). This results in a different correlation pattern between the tasks. The learned MT-DT has an early decision leaf for T_2 since knowing that $x_1 \leq -2$ is enough to predict the label class $*$ for T_2.



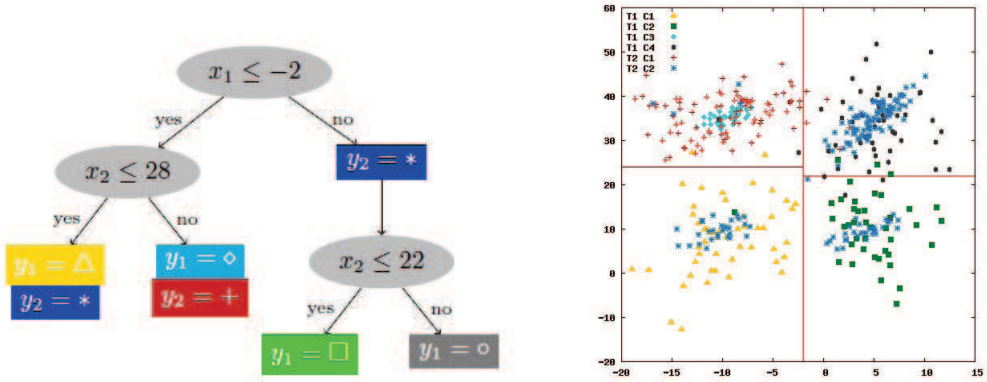
(a) `MT-DT` with early decision leaf

(b) Two $2D$ mixture of Gaussian tasks.

Figure 3.3: `MT-DT-1`

Apart from the structure difference, the main challenge when moving from one- to multi-task learning is in the optimal way of using the information gain criteria. In the next chapter we show how `MT-DT` can profit from the multi-task setting. We prove a theorem which helps increase the multi-task information gain over one-task case. This improved criterion combined with the boosting leads to an important performance increase.

As we have seen; like `2T-stumps`, `MT-DTs` can classify tasks with different number of class labels and without any a priori assumption on the relatedness patterns of the tasks. Their learning algorithms will show that they do not need shared examples between tasks. However, our proposed classifiers differ in their way of modeling shared information between tasks, a `2T-stump` uses a sequential paradigm where a stump learned for a certain task at the root divides the space in a way that makes learning other tasks easier. Differently said, learning a task at the root provides additional information to learning other tasks. In the case of `MT-DT`, the information sharing is joint in the sense that each node; during learning, exploits the data of several tasks. Nevertheless, no paradigm can be said to be absolutely better than the other, it is dependent on the tasks and how they are related. Another difference is that a `2T-stump` is concerned with



(a) `MT-DT` with early decision leaf

(b) Two $2D$ mixture of Gaussian tasks.

Figure 3.4: `MT-DT-2`

a subset of tasks, whereas, an `MT-DT` makes prediction on all tasks; a property which can be beneficial as it uses more data for building the classifier, but on the other hand it might cause problems of computation and generalization when the number of tasks becomes large.

3.3 Summary

In this chapter we presented novel multi-task hypotheses. They are both decision tree-like classifiers. The first (`2T-stump`) uses a sequential knowledge transfer paradigm, where a stump for a task is learned at the root, then at each split created by the root stump, other stumps for other tasks are learned. The second classifier is a multi-task decision tree. Each node of the tree could be shared among one or more tasks.

In the following chapter we introduce Ensemble methods, then we present our Ensemble Multi-Task algorithm along with the learning algorithm we propose to learn `2T-stumps` and `MT-DTs`.

Chapter 4

Ensemble Multi-Task Learning

Contents

This chapter covers the learning algorithms contributed by this thesis. Before presenting them we start by introducing the framework of *Ensemble Learning*. We briefly review its history as well as its advantages in comparison to other machine learning methods. At the core of ensemble learning lies the weak learners as the key component of the ensemble to be learned. We will present two of the most widely used weak learners for classification. Afterwards, we elaborate on Boosting methods by presenting a generic boosting algorithm, then we go through the popular Adaboost along with its variations theoretical properties and guarantees. The Bagging method is also addressed because of its interesting computational efficiency and its ability to avoid overfitting especially in noisy tasks.

In the sequel, we present our contribution in extending ensemble learning framework to Multi-Task setting. We start by giving our notation and problem definition, then we present MT-Adaboost algorithm; a multi-task adaptation of Adaboost. Variations of the algorithm for multi-class multi-task classification are addressed, namely *MT-Adaboost.M1* and *MT-Adaboost.MH*. We show the error convergence bounds on MT-Adaboost which proves that it is a proper boosting

algorithm with guarantees on the reduction of the training error. Then, we propose different learning algorithms to the multi-task weak classifiers presented in the previous chapter.

4.1 Ensemble of Classifiers

We start this part by a simple motivating example of a human task. This example is cited from Freund and Schapire [1999a]: *A horse-racing gambler, hoping to maximize his winnings, decides to create a computer program that will accurately predict the winner of a horse race based on the usual information (number of races recently won by each horse, betting odds for each horse, etc.). To create such a program, he asks a highly successful expert gambler to explain his betting strategy. Not surprisingly, the expert is unable to articulate a grand set of rules for selecting a horse. On the other hand, when presented with the data for a specific set of races, the expert has no trouble coming up with a rule of thumb for that set of races (such as, Bet on the horse that has recently won the most races or Bet on the horse with the most favored odds). Although such a rule of thumb, by itself, is obviously very rough and inaccurate, it is not unreasonable to expect it to provide predictions that are at least a little bit better than random guessing. Furthermore, by repeatedly asking the expert's opinion on different collections of races, the gambler is able to extract many rules of thumb.*

In order to use these rules of thumb to maximum advantage, there are two problems:

- How should he choose the collections of races presented to the expert (training samples) so as to extract the most useful rules of thumb?.
- Once he has collected many rules of thumb, how can they be combined into a single, highly accurate prediction rule (final classifier)?.

Different answers are given to the above questions, each define an ensemble

learning approach. A widely used and one of the most studied approach; on which the presented work is based, is called *boosting*. In this approach the two questions are addressed as follows. Regarding the choice of the collections of races, boosting puts the most weight (importance) on the examples most often misclassified by the preceding weak rules of thumb. As a consequence the next expert will be forced to come up with a rule which is good at handling those misclassified examples. As for the second question; how to combine the rules ? Boosting does combine them through a weighted majority vote, where the weights are correlated with accuracy of each rule, higher weights are granted to the more accurate rules.

In ensemble learning terms, the expert who is asked to come up with rules of thumb is called a *weak learner* or *base learner*. Nevertheless, there is also the question of what algorithm to use in order to learn the rules of thumb ? (what weak learner to use?). This question is answered in the section 4.1.3. However, in the next section we present advantages of using ensemble learning over single hypothesis learning.

4.1.1 Motivating Advantages of Ensemble Learning

Learning algorithms that output only a single hypothesis might suffer from three problems that can be partly resolved by ensemble learning: the statistical problem, the computational problem and the representation problem.

The statistical problem is encountered when the learning algorithm is searching a space of hypotheses that is too large for the amount of available training data. In such cases, there may be plenty of different hypotheses that all perform equally on the training data. In single hypothesis approaches, the algorithm will choose one of them to output. The risk here is to choose a hypothesis which is not good at predicting future data points. However, with a simple vote of those equally-good hypotheses we can reduce the risk of overfitting.

The computational problem arises when the learning algorithm cannot guar-

antee to find the best hypothesis within the hypothesis space. In some algorithms like decision trees, finding the best hypothesis that fits the training data is intractable. So, heuristics are used with a risk of getting stuck in a local minima and hence fail to find the best hypothesis. Again, a simple vote of those local minima solutions might overcome the problem.

Finally, the representation problem arises when the hypothesis space does not contain any hypotheses that are good approximations to the target function. In such cases, a weighted vote of hypotheses expands the space of functions. Thus, the algorithm might be able to find a more accurate approximation to the target function.

An algorithm that suffers from the statistical problem is said to have high variance. An algorithm that suffers from the computational problem is described as having computational variance. And an algorithm that suffers from the representation problem is said to have high bias. So, ensemble methods can reduce both the bias and the variance of learning algorithms.

4.1.2 A Brief History of Ensemble Learning

In the literature of ensemble learning there are two main approaches. They are based on the way the ensemble of classifiers or hypotheses is constructed, more specifically, in the first approach the hypotheses are learned in a coordinated way; boosting has emerged from this approach. Whereas, in the second approach the hypotheses are learned independently.

4.1.2.1 Interdependently Constructed Ensemble Methods

The idea of Boosting has its roots in PAC learning (cf. Valiant [1984]). Kearns and Valiant [1994] proved a counter-intuitive fact: learners which can perform only slightly better than random guessing, can be combined to form an arbitrarily good hypothesis (under the condition of data availability). The first polynomial time boosting algorithm was proposed by Schapire [1990]. However, the first

application to boosting to real-world OCR task was presented by Drucker et al. [1993], in their algorithm they used neural networks as base learners.

Although Boosting principle seems intuitive in terms of algorithmic design, a step forward in the theoretical interpretation was taken by explaining Boosting in terms of a stage-wise gradient descent procedure in an exponential cost function (refer to Breiman [1997]; Frean and Downs [1998]; Friedman et al. [1998]). From practical point of view, a considerable part of the early literature of boosting has pointed out (based on empirical studies) that boosting does not exhibit overfitting even when running for a large number of iterations. However, simulations by Grove and Schuurmans [1998] on data sets with higher noise content could clearly show overfitting effects. In this context and in order to avoid overfitting, it is important to elucidate the relations between Optimization Theory and Boosting (e.g. Breiman [1997]; Freund and Schapire [1999b]). Studying this relationship opened the field to new types of Boosting algorithms: Boosting for regression tasks was proposed by Duffy and Helmbold [2000]; Rtsch et al. [2000]. Unsupervised learning tasks were approached by boosting algorithms as well Campbell and Bennett [2001]; Rtsch et al. [2002]. Studying boosting from optimization theory point of view helped also in establishing convergence proofs for Boosting algorithms.

4.1.2.2 Independently Constructed Ensemble Methods

Another yet simpler approach, to construct an ensemble is to run the based learning algorithm several times provided with a different training data each time. Breiman [1996] introduced the *Bagging (Bootstrap Aggregating)* method which functions as follows. Given a set of m training examples, Bagging chooses in each iteration a set of examples of size $\leq m$ by sampling uniformly with replacement from the original data set. This yields in a training set with some points appearing eventually multiple times whereas others dot appear. If the weak learning algorithm is *unstable* -which means that small changes in the training set

might lead to great changes in the resulting hypothesis-then Bagging will learn a diverse ensemble of hypotheses, which is a key feature in ensemble learning; learning an ensemble of almost similar hypotheses would not result in a ensemble that is significantly better than its componenents.

A second way to induce diversity in the weak learner and to build independently an ensemble of diverse hypotheses is to change the features at each iteration instead of changing the training examples. This method is called *Random Sub-Space Sampling*.

A third way to induce diversity is to introduce randomness at the algorithmic level. For example, the backprpagation algorithm can be run many times, starting each time from a different random setting of weights. Decision trees algorithms can be randomized by adding randomness to the process of choosing which feature and threshold to split on. Dietterich [2000] reported significant improvements of randomized ensemble of trees over simple trees and other simple classifiers. Ho [1998] introduced random sub-space method for decision trees. His method chooses a random subset of the features at each node of the tree, and constraints the learning algorithm to choose the splitting rule at this node of the sampled subset of features. This algorithm is called *Random Forests* algorithm, it has shown very nice empirical features, from noise resistance, slow overfitting to computational efficiency.

4.1.3 Common Weak Learners

We present below two of the most common weak learners; they are relevant for our work as they represent the building blocks for our learners.

Decision Stumps A decision stump is very simple classifier model, it consists of one test on a feature value and two leafs corresponding to the output of the stump Ai and Langley [1992]. Depending on the type of the input feature, several variations are possible. For categorical (discrete) features, a stump can

either contain a leaf for each possible value of a certain feature or it can be with only two leaves, one of which corresponds to some chosen category, and the second one to all the other categories. For continuous features, a threshold feature value is selected, and the stump contains two leaves one values below and the other for the values above that threshold. Stumps can be learned by an exhaustive search over all possible stumps in order to chose the one with best predictive performance. This scheme is costly when the number of features and their possible values is very large. In this case, one can sample a subset of features and chose the best stump for this subset. Despite that sampling will not result in the best possible stump, but knowing that it is better than random guessing is enough to boost its performance by constructing an ensemble of stumps through an ensemble learning algorithm.

Decision Trees Decision tree learning is the well known technique in statistics and machine learning; it uses a decision tree as a predictive model which maps observations from the instance space to the target values. In the case of classification, tree leaves represent class labels and branches represent conjunctions of item attributes that lead to those class labels. In the C4.5 and C5.0 tree generation algorithms Quinlan [1993], the decision tree learning uses the concept of the *information gain* (IG) from the information theory. At the root of the tree, the algorithm chooses an attribute that yields the highest IG on the training set. Such an attribute splits the training set into two subsets whose sum of labels entropy is the lowest. The algorithm then recursively applies the information gain rule on the subsets. The recursion is stopped when all items of a subset have the same label, a decision leaf corresponding to this label [1]. The information gain about a random variable Y obtained from an observation that a random variable X takes the value $X = x$ is the Kullback-Leibler divergence $D_{KL}(p(Y|X)||p(Y|I))$ of the prior distribution $p(Y|I)$ from the posterior distribution $p(Y|X)$ for Y given X.

[1] Some pruning is often used to generalize the rules learned to unobserved items.

Also, the expected value of the information gain is The IG rule defines a preferred sequence of attributes to investigate to most rapidly narrow down the state of Y. An attribute with high information gain should be preferred to other attributes.

4.2 Generic Boosting Algorithm

Boosting algorithms share a generic scheme which is summarized as follows:

- A given training set $S = \{(x_i, y_i); i \in \{1 \in \dots m\}\}$ drawn *i.i.d.* from an unknown distribution $\mathcal{D}$.
- A weak learning algorithm L defined on a hypotheses space $h \in \mathcal{H}$
- The goal is to construct an ensemble hypothesis H with an arbitrarily close to zero error ϵ by running L several times (T rounds):

  For $t = 1, \dots T$:
  Construct D_t , where D_t is a distribution over the indices $1, \dots, m$
  Run L on D_t to produce h_t, *s.t.* :
  $err_{D_t}(h_t) = \gamma_t$, where $\gamma_t \geq \gamma > 0; \forall t$ (weak learning assumption)
  End
  Output H (a combination of the weak hypotheses $h_1, \dots, h_T$)

Any concrete boosting algorithm must define how to build the distribution at each round and how to combine the weak hypotheses to produce H. We present next *Adaboost* algorithm as it is the most successful boosting algorithm.

4.3 Adaboost

Adaboost constructs the distributions D_t in a recurrent way. The intuitive idea is to increase the weights on "hard" examples (examples which have been misclas-

sified on earlier round), and decrease the weights on "easy" examples (examples which have been classified correctly).

$$D_1(i) = 1/m$$

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } y_i = h_t(x_i) \\ \exp(\alpha_t) & \text{if } y_i \neq h_t(x_i) \end{cases}$$

where $\alpha_t > 0$ are the weights given to each hypothesis, and Z_t is a normalizing constant to ensure that $D_t + 1$ is a probability distribution. The weak hypotheses are combined to form the function $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$, note that in the case of binary classification where $\mathcal{Y} = \{-1, +1\}$, the final hypotheses is $H(x) = sign(f(x))$.

Also, note that D_t is a probability distribution over the indices $\{1, \ldots, m\}$ so that $D_t(i)$ represents the probability of the pair (x_i, y_i). For the time being, we did not mention how Adaboost chooses α_t(s). Follows, we give an analysis of the training error which yields in a upper bound on it, then we derive the values of α_t that minimize this bound.

4.3.1 Training Error

One of the reasons that granted AdaBoost its wide popularity and made it more practical for applications, is that it does not depend on the weak learning parameter γ. In this sense, the algorithm can adapt to the weak learning algorithm (hence, the name AdaBoost). Moreover, as this section will show, the training error of a hypothesis H generated by AdaBoost decreases exponentially fast in the number of rounds T.

Theorem 4.1. *Let H be the output hypothesis of AdaBoost which has a training error ($\hat{\epsilon}$). Then:*

$$\leq \prod_{t=1}^{T} (2\sqrt{\epsilon_t(1-\epsilon_t)})$$

$$= \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2}$$

$$\leq \exp(-2 \sum_{t=1}^{T} \gamma_t^2)$$

where $\epsilon_t = 0.5 - \gamma_t$ *is the training error of* h_t*. The last line is obtained from the inequality* $1 + x \leq \exp(x)$*. Thus if* $\forall t; \gamma_t \geq \gamma > 0$ *then:*

$$\hat{\epsilon} \leq \exp(-2T\gamma^2),$$

Proof. The proof is given in Freund and Schapire [1997], we sketch its three steps:

1. Unravel the recurrence of distribution update rule in order to write the distribution at $T + 1$ as follows:

$$D_{T+1}(i) = \frac{\exp\left(-y_i f(i)\right)}{m \prod_{t=1}^{T} Z_t}$$

2. Bound the training error of H by the product of Z_ts:

$$\hat{\epsilon} \leq \prod_{t=1}^{T} Z_t$$

3. Now that the training error has been bounded in step 2 by the product of the normalizing weights Zt , the last step is to express Zt in terms of ϵ_t :

$$Z_t = 2\sqrt{\epsilon_t(1 - \epsilon_t)}$$

In order to do so, we calculate the value of a_t which minimizes $Z_t = \sum_{i=1}^{m} D_t(i) \exp(-a_t y_i h_t(x_i))$. This results in $a_t = 0.5 \ln(\frac{1-\epsilon_t}{\epsilon_t})$

□

4.3.2 Multi-Class Adaboost Variations

Several algorithms have been proposed to extend AdaBoost to the multi-class setting. The most straightforward generalization is called *AdaBoost.M1* Freund and Schapire [1997], it is suitable when the base learner is strong enough to achieve reasonably high accuracy (e.g., decision trees). However, this method fails if the weak learner cannot achieve at least 50 percent accuracy, which is not always easy in the case of multiple class labels. For instance, a random guessing between 5 class labels would achieve 20 percent of accuracy, so requiring at least 50 might not be feasible by many weak learners.

To cope with the latter limitation, several more sophisticated methods have been developed. These generally work by reducing the multi-class problem to several binary problems. Freund and Schapire [1997] introduced *AdaBoost.MH* algorithm. It works by creating a set of binary problems, one for each class label. Then it learn them all in a single boosting scheme. Another algorithm introduced by the same authors is called *AdaBoost.M2* (which is a special case of Schapire and Singer [1999] *AdaBoost.MR* algorithm). Those also reduce the problem to binary, but learn instead to discriminate between pairs of (correct, incorrect) labels.

However, these methods require additional effort in the design of the weak learner. A different approach Schapire [1997], which makes use of Dietterich and Bakiri [1995] method of error-correcting output codes, achieves similar provable bounds to those of AdaBoost.MH and AdaBoost.M2, but it can be used with any weak learner that can handle binary tasks.

4.4 Bagging

Another well-known method to combine weak hypotheses in order to construct a stong one is called *bootstrap aggregating* or *bagging*, it has been invented by Leo Breiman (see his paper Breiman [1996].

Briefly, the method works by training the weak learner at each iteration on a bootstrap sample of the training set, i.e., a sample with replacement of the training set with the same size. The multiple hypotheses that are computed are then combined using a simple majority voting ($a_t = 1$). The method can be quite efficient, especially, for noisy data, because; thanks to its random behavior, it does not concentrate its effort on noisy examples till they are well learned, thus, it could avoid overfitting.

4.5 Summary

This part of the chapter presented ensemble learning framework by focusing the most on Boosting methods as they lie in the center of our interest because of their nice theoretical properties. We presented also Bagging method as a computationally efficient ensemble method because of its sampling mechanism on the features and on the examples as well. In the following, we will present our contribution in extending ensemble learning framework to Multi-Task setting. We start by giving our notation and problem definition, then we present MT-Adaboost algorithm a multi-task adaptation of Adaboost. We show its error convergence bounds. Then, we propose different learning algorithms to the multi-task weak classifiers presented in the previous chapter.

4.6 Multi-Task Ensemble Learning

In the following sections we will be presenting our contribution to Multi-Task learning. We present the learning algorithms for our Multi-Task classifiers which were presented in the previous chapter. We then plug them into Boosting algorithms adapted to the Multi-Task setting and discuss their convergence properties. We first start by defining our notation.

Let $\mathcal{X}$ be the instance space, a supervised classification task T is defined as

follows. Let D be a distribution over $\mathcal{X}$, let $f : \mathcal{X} \to Y$ be a target function, given a sample $S = \{(x_i, f(x_i)) \mid x_i \in \mathcal{X},\ 1 \leq i \leq m\}$, find an hypothesis function h which minimizes error$(h) = Pr_{x \sim D}[h(x) \neq f(x)]$.

Now, let us consider a sample S in the multi-task setting, it can also be written as $S = \cup_{j=1}^{j=N} S_j$ where, for every j, $S_j = \{e_i =< x_i, y_i, j > |\ y_i = f_j(x_i)\}$, i.e. we decompose S in samples corresponding to the different tasks. Let us consider a distribution over S.

We consider N classification tasks $T_1, \ldots, T_N$ over the instance space $\mathcal{X}$ and label sets $Y_1, \ldots, Y_N$. For sake of clarity, we consider binary classification tasks and we assume without loss of generality that the binary labels for all tasks are encoded as -1 and $+1$, then we describe how to generalize the work to multi-class classification tasks with different labels (and eventually different number of labels) per task.

The objective is to solve the N classification tasks simultaneously. We suppose a distribution D over $\mathcal{X} \times \{1, \ldots, N\}$. We will assume that, for every j in $\{1, \ldots, N\}$, the projection on the distribution's j-th component will correspond to the original distribution for task T_j. A multi-task classification algorithm will take as input a sample $S = \{< x_i, y_i, j_i > |\ x_i \in \mathcal{X}, y_i = f_{j_i}(x_i) \in \{-1, +1\}, j_i \in \{1, \ldots, N\},\ 1 \leq i \leq m\}$. It should be noted that a same instance x can appear in a sample S with its label for different tasks. The goal is to find an hypothesis $h : \mathcal{X} \to Y_1 \times \ldots \times Y_N$ which minimizes error$(h) = Pr_{<x,j> \sim D}[h_j(x) \neq f_j(x)]$, where $h_j(x)$ is the j-th component of $h(x)$ and $j \in \{1, \ldots, N\}$.

In the following sections, we first present our Multi-Task Boosting algorithm with `2T-stump` as weak learner. Then we present the algorithm with `MT-DT` as weak learner

4.7 Abstaining Multi-task Adaboost with 2T-stumps

In this section, Then we show that our formalization of multi-task problems will allow to adapt Adaboost and to benefit from theoretical results of boosting. Finally, we propose several weak learners in order to provide weak multi-task hypotheses to Adaboost.

4.7.1 MTAA: Adaboost for Multi-task Learning with Abstention

We consider the multi-task setting as defined above, with `2T-stumps` used as weak classifiers. Given a `2T-stump` $h : \mathcal{X} \to \{-1, 0, +1\}^N$, we define W_0, W_{-1} and W_{+1} by

$$W_b = \sum_{i=1}^{i=N} D(\{e_i =< x_i, y_i, j > |\ e_i \in S,\ y_i \times h_j(x_i) = b\})$$

for $b \in \{-1, 0, +1\}$. We abbreviate W_{-1} and W_{+1} by W_- and W_+ respectively. W_- is the sum of weights of misclassified instances, W_+ is the sum of of weights of well-classified instances, and W_0 is the sum of weights of instances on which h abstains. Since D is a distribution, $W_+ + W_- + W_0 = 1$.

4.7.1.1 MTAA.

Adapted from Schapire and Singer [1999], the generic Adaboost algorithm with abstention for multi-task learning (`MTAA`) is presented in Algorithm 1 where T is the number of boosting iterations; `init` is a procedure to initialize the distribution D_1 over S; and `WL` is a weak learner that returns a `2T-stump` given as input a sample S and a distribution D over S. The final output is a classifier H from $\mathcal{X}$ into $\{-1, +1\}^N$. We should note that we suppose that the choice of the number

of boosting iterations, the choice of the updating coefficients α and the weak learner `WL` for `MTAA`, ensure that H does not abstain for every instance and for every task.

Require: $S = \cup_{j=1}^{j=N}\{e_i =< x_i, y_i, j >| \; x_i \in \mathcal{X};\; y_i \in \{-1, +1\}\}$
1: $D_1 = \texttt{init}(S)$ *initialize distribution*
2: **for** $t = 1$ to T **do**
3: $h^t = \texttt{WL}(S, D_t)$ {*train the weak learner and get an hypothesis* `2T-stump` }
4: Choose α_t
5: $D_{t+1}(e_i) = \frac{D_t(e_i)\exp(-\alpha_t y_i h_j^t(x_i))}{Z_t}$ {*update distribution*}
6: **end for**
7: **return** Classifier H defined by $H_j(x) = \texttt{sign}(\sum_{i=1}^{i=T}\alpha_t h_j^t(x))$, $1 \leq j \leq N$

Algorithm 1: A generic version of `MTAA`

The training error of the final classifier H is defined by

$$\text{error}(H) = Pr_{<x_i,y_i,j>\sim D_1}[H_j(x_i) \neq y_i].$$

We now prove that the training error decreases to zero exponentially fast.

Theorem 4.2. *Let us consider* `MTAA` *with the update rule in line 5 of Algorithm 1 and let us suppose that there exists* $\gamma > 0$ *such that, at each boosting iteration,* $W_+ - W_- \geq \gamma$, *then*

(i) $error(H) \leq \prod_{t=1}^{T} Z_t$,

(ii) Z_t *is minimized by choosing*

$$\alpha_t = \frac{1}{2}\ln\left(\frac{W_+}{W_-}\right), \tag{4.1}$$

(iii) $\prod_{t=1}^{T} Z_t \leq e^{-T\frac{\gamma^2}{2}}$.

Proof. Following Schapire and Singer [1999], it can be shown that $\text{error}(H) \leq \prod_{t=1}^{T} Z_t$. Also, for every t, $Z_t = W_0 + e^{\alpha}W_- + e^{-\alpha}W_+$ where W_0, W_- and W_+ are

computed as above with $D = D_t$. And, it can be verified that Z_t is minimized when $\alpha_t = \frac{1}{2}\ln(\frac{W_+}{W_-})$. With this setting of α_t, we have $Z_t = W_0 + 2\sqrt{W_-W_+} = 1 - (\sqrt{W_+} - \sqrt{W_-})^2$.

It remains to show property (iii), i.e. to show that the training error decreases exponentially with the number of boosting iterations. Let $W_+ - W_- = \gamma_t \geq \gamma > 0$, then $\sqrt{W_+} - \sqrt{W_-} = \frac{\gamma_t}{\sqrt{W_+}+\sqrt{W_-}} \geq \frac{\gamma_t}{\sqrt{2}}$ because $\sqrt{W_+} + \sqrt{W_-} = \sqrt{W_+} + \sqrt{1 - W_0 - W_+} \leq \sqrt{W_+} + \sqrt{1 - W_+} \leq \sqrt{2}$. Thus we get $Z_t \leq 1 - \frac{\gamma_t^2}{2}$, and then we have $\prod_{t=1}^{T} Z_t \leq \prod_{t=1}^{T}(1 - \frac{\gamma_t^2}{2}) = e^{\sum_{t=1}^{T}\ln(1-\gamma_t^2)}$. Last using $\gamma_t \geq \gamma > 0$ and $\ln(1-x) \geq -x$, we obtain: $\text{error}(H) \leq \prod_{t=1}^{T} Z_t \leq e^{-T\frac{\gamma^2}{2}}$. □ □

Thus, the output of the weak learner is a weak classifier that must satisfy $W_+ - W_- \geq \gamma > 0$. Or, equivalently, it must satisfy $W_- + \frac{1}{2}W_0 \leq \frac{1}{2} - \frac{\gamma}{2} < \frac{1}{2}$ because $W_+ + W_- + W_0 = 1$. Then, the goal of a weak learner will be to maximize $W_+ - W_-$ which is equivalent to minimize $W_- + \frac{1}{2}W_0$. It should be noted that these equivalent criteria avoid hypothesis such that $W_- \leq W_+$ with small values for W_- and W_+ (and W_0 is high). This is consistent with the intuition that weak classifiers are not allowed to abstain on a sample of high weight.

The optimal rule given in Equation 4.1 for updating coefficients α_t may lead to very large or infinite values, then it can be necessary to smooth the update rule for coefficients, then leading to the update rule:

$$\alpha_t = \frac{1}{2}\ln\left(\frac{W_+ + \epsilon}{W_- + \epsilon}\right) \tag{4.2}$$

where ϵ is a small real number.

4.7.1.2 MTAA with conservative weighting strategy.

We now consider a more conservative version of Adaboost for hypotheses that abstain proposed in Schapire and Singer [1999] with a modified update weighting rule as defined in Theorem 4.3. We have the following result:

Theorem 4.3. *Let us consider* `MTAA` *with the new update rule*

$$D_{t+1}(e_i) = \frac{D_t(e_i)\exp(-\alpha_t y_i h_j^t(x_i))}{Z_t} \qquad \text{if } h_j^t(x_i) \neq 0 \tag{4.3}$$

$$= \frac{D_t(e_i)(\frac{exp(\alpha)+exp(-\alpha)}{2})}{Z_t} \qquad \text{otherwise,} \tag{4.4}$$

and let us suppose that there exists $\gamma > 0$ *such that, at each boosting iteration,* $W_- + \frac{1}{2}W_0 \leq \frac{1}{2} - \gamma$*, then*

(i) $error(H) \leq \prod_{t=1}^{T} Z_t$*,*

(ii) Z_t *is minimized by choosing*

$$\alpha_t = \frac{1}{2}\ln\left(\frac{W_+ + \frac{1}{2}W_0}{W_- + \frac{1}{2}W_0}\right), \tag{4.5}$$

(iii) $\prod_{t=1}^{T} Z_t \leq e^{-2T\gamma^2}$*.*

The proof is not given because it is very similar to the proof of Theorem 4.2.

4.7.1.3 Discussion on the weighting strategies.

Let us compare the two weighting strategies implied by Theorems 4.2 and 4.3 when the number of tasks N increases. Every `2T-stump` must abstain for every instance on $N-2$ tasks, which leads to larger values of W_0. Accordingly, the quantities Z_t and $\prod_{t=1}^{T} Z_t$ become close to 1. Moreover, with the weighting strategy given by Theorem 4.3, large values of W_0 may lead to values of α_t close to 0 and the normalization factor Z_t converges more quickly to 1. Finally, let us denote by Z_1 and Z_2 the normalization factors according to Theorems 4.2 and 4.3, we can note that Z_1 is always smaller than Z_2 because $Z_2^2 - Z_1^2 = 2W_0(\sqrt{W_+} - \sqrt{W_-})^2$.

We show in Figure 4.1 random trajectories of $\prod_{t=1}^{T} Z_t$ with fixed values of W_0 and γ. For small values of W_0, the two weighting strategies are similar. But with a greater value of W_0, we obtain very different results for the second strategy:

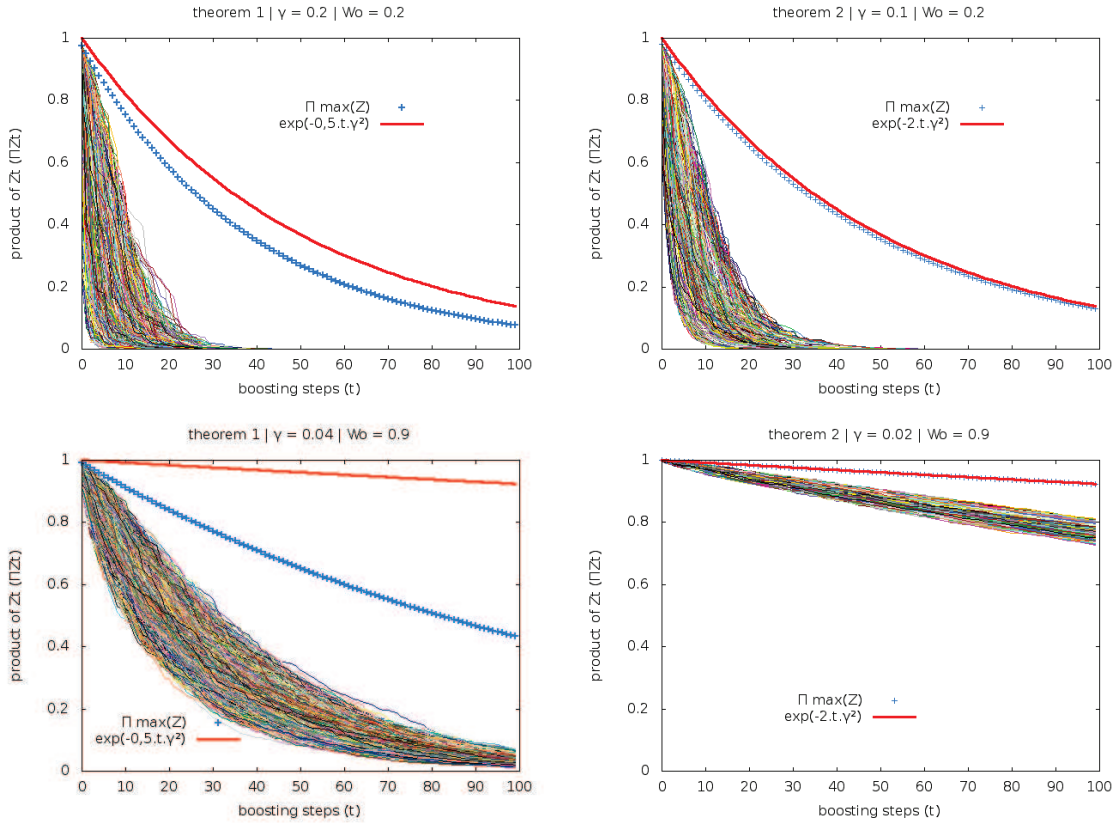


Figure 4.1: Evolution of $\prod_{t=1}^{T} Z_t$ following the two possible strategies for different values of W_0: Theorem 4.2 (left) and Theorem 4.3 (right); $W_0 = 0.2$ (up) and $W_0 = 0.9$ (down).

the trajectories are close to $y = 1$. Let us also note that the theoretical bound for the first strategy on the training error is not precise for small values of the edge γ induced by great values of W_0 and we have to directly minimize $\prod_{t=1}^{T} Z_t$ in order to control the training error.

We conclude the section with a note on the number of boosting iterations. When the number N of tasks increases, the edge γ is small, thus we have to increase the number of boosting iterations: if T is the number of boosting iterations for a single task boosting algorithm, we suggest to use `MTAA` with a number of boosting iterations equal to $\frac{N.T}{2}$. And, we recall that the goal of a weak

learner is to output hypotheses maximizing $W_+ - W_-$ (or equivalently minimizing $W_- + \frac{1}{2}W_0$).

In the sequel of the paper, we will only consider the weighting strategy implied by Theorem 4.2.

4.7.2 The Weak Learners for `2T-stump`

A weak learner `WL` takes as input a sample S and a probability distribution D over S and returns a `2T-stump` h. The objective of `WL` is that the score $W_- + \frac{1}{2}W_0$ of the output h w.r.t. S and D is as small as possible. We suppose that a set of tests has been previously computed according to attributes and values observed in the sample S. For sake of clarity, we will suppose that the tests are binary. This defines a set of stumps where every stump is defined by a test, a task and two decision nodes.

A first naive weak learner compute the score of all `2T-stumps` and then select the best `2T-stump`. But the complexity is cubic in the number of tests. A second naive algorithm is to select at the first level the stump with the best score and then greedily select, at each of the two decision nodes, the stump with the best score. It can be easily shown that such an algorithm does not output the best, at least a good, `2T-stump` because the algorithm does not take into account the correlation between tasks. For instance, a greedy algorithm will learn the `2T-stump` shown in Figure 3.1(b) on a synthetic dataset visualized in Figure 3.2(b) because the best first-level decision stump is obtained for task T_3, while a less naive algorithm, as defined below, will find the `2T-stump` shown in Figure 3.1(a) and visualized in Figure 3.2(a). Therefore we now introduce weak learners with the aim of finding a good `2T-stump` while avoiding the cubic complexity.

Let S be a (multi-task) sample and W be (a vector of) weights for examples in S. We define a procedure `Score` which assigns to every stump st a score w.r.t. S and W. The score will be chosen equal to $W_- + \frac{1}{2}W_0$. We define functions `Best` and `Best-per-Task` as follows. Given a sample S and weights W, `Best` takes as

input a positive integer K and outputs the set of K stumps with highest score; `Best-per-Task` takes as input an integer K and output, for every task *task*, the set of K stumps for *task* with highest score. Also, we can define a distribution over stumps where the probability of a stump is inversely proportional to its score. Then, the function `Sto-Best` takes as input an integer K and output a list of K stumps drawn randomly according to this distribution.

First, we define the weak learner `WL-Best-K` in Algorithm 2. Given a score function, the K stumps with the best scores are chosen as candidates for the root. Let us suppose that the chosen stump *st* corresponds to task j. Then, we consider the set $S \setminus S_j$ and split the training examples according to the test defined by *st*. This defines the sets S_1 and S_2. We define the weights of examples to be equal to the probabilities given by D. For the second level, we choose the best stump for each branch. Note that if we consider the score defined by $W_- + \frac{1}{2}W_0$, we can show that this implies to choose the best `2T-stump` with root stump *st*. Then `WL-Best-K` output the `2T-stump` with the best score among the K candidate `2T-stumps`.

Require: $S = \cup_{j=1}^{j=N}\{e_i =< x_i, y_i, j >\}$ and a distribution D on S; parameter K

1: Compute $BeST =$ `Best`(K) w.r.t. S, D {*choose K root stumps*}
2: **for** every stump *st* in $BeST$ **do**
3: Compute S_1 and S_2 {*descend examples according to the root test*}
4: Let $W_1 = D|_{S_1}$; $st_1 =$ `Best`(1) w.r.t. S_1, W_1 {*the best stump for the left child*}
5: Let $W_2 = D|_{S_2}$; $st_2 =$ `Best`(1) w.r.t. S_2, W_2 {*the best stump for the right child*}
6: Let h_{st} be the `2T-stump` with root *st*, left child st_1 and right child st_2
7: **end for**
8: **return** `2T-stump` h with the best score among all h_{st} for *st* in $BeST$

Algorithm 2: Weak learner `WL-Best-K`

The weak learner `WL-Best-K` chooses the K best stumps as candidates for the root of the output `2T-stump`. It may be the case that these K best stumps

concern only some of the tasks because slight variations of a test can lead to slight variations of the score. Therefore, we also consider the weak learner `WL-Best-per-Task-K` obtained by replacing the instruction $BeST = \texttt{Best}(K)$ in line 1 of `WL-Best-K` by $BeST = \cup_{j=1}^{N}\texttt{Best-per-Task}(k)$. Let K be an integer and let us consider $K = k \times N$, where N is the number of tasks. The weak learner `WL-Best-per-Task-K` with parameter value K will ensure that among the K stumps chosen at the root, for every task, K stumps are chosen. Thus, all tasks are represented when computing a hypothesis `2T-stump` with `WL-Best-per-Task-K` which was not the case for `WL-Best-K`.

It can be shown that neither of these algorithms is ensured to output an optimal hypothesis because the choice of root stumps is made independently of the choice of the second-level stumps. And it can be the case that the best `2T-stump` has a root stump which is not in the K-best stumps. Thus, we also consider the idea to introduce diversity in the choice of the candidate root stumps. For this, we define the weak learner `WL-Sto-Best-K` by replacing the instruction $BeST = \texttt{Best}(K)$ in line 1 of `WL-Best-K` by $BEsT = \texttt{Sto-Best}(K)$.

The three weak learners are not optimal. But, the complexity of `WL-Best-K` and `WL-Best-per-Task-K` is in $O((N \times Tt \times k)$ where N is the number of tasks, Tt the number of tests and K is the parameter value. The complexity of `WL-Sto-Best-K` must include also a logarithmic factor. We compare empirically the weak learners in the next chapter.

4.8 Multiple Multi-Class Tasks with MT-Adaboost

So far we presented MT-Adaboost and `2T-stumps` for binary classification tasks. In this section we propose to extend them to cope with multi-class tasks. We adapt *BoosTexter* Schapire and Singer [2000] learning system to multi-task learning (MTL). BoosTexter is the extention to the original binary classification Ad-

aboost to multi-class / multi-label problems. It has different variations in terms of booster (*Adaboost.MH*, *Adaboost.MR*, etc). We consider Adaboost.MH as a booster and multi-class /multi-label stumps, then we plug them into our multi-task boosting framework. Adaboot.MH transforms a multi-label problem to several binary problems each of which corresponds to a label. Each training example x has a set of labels Y instead of one label. For each training example (x, Y) and each label $l \in \mathcal{Y}$, we define:

$$Y[l] = \begin{cases} +1 & \text{if } l \in Y \\ -1 & \text{if } l \notin Y \end{cases}$$

Each example (x, Y) in the training data set is transformed to $|\mathcal{Y}|$ examples of the form $((x, l), Y[l])$ for all $l \in \mathcal{Y}$. The distribution is over the pairs (example, label), i.e., over $\mathcal{X} \times \mathcal{Y}$ The algorithm can learn multi-class as well as multi-label problems, multi-class is a special case of multi-label, where $|Y| = 1$ for all examples. The output classifier is the following function:

$$H : \mathcal{X} \times \mathcal{Y} \to \mathcal{R},$$

where the sign of $H(x, l)$ indicates if l is predicted as one of x's labels (positive) or not (negative), whereas, the magnitude of $H(x, l)$ indicates the classification confidence.

Adaboost.MH minimizes the Hamming loss between the predicted labels and the given labels:

$$Hamming(S) = \frac{1}{m} \sum_{i \in 1,\ldots,m} |Y_i \Delta H(x_i)|,$$

where $H(x_i) = \{H(x_i, l); l \in \mathcal{Y}\}$

We adapt Adaboost.MH to MTL. The distribution we define is over examples

and labels from all tasks, i.e.,

$$D \in \mathcal{X} \times \{\mathcal{Y}_1 \cup \ldots \cup \mathcal{Y}_N\}.$$

And the output classifier:

$$H : \mathcal{X} \times \{\mathcal{Y}_1 \cup \ldots \cup \mathcal{Y}_N\} \rightarrow \mathcal{R}.$$

Algorithm 3 present the extention to Adaboost.MH to MTL.

Require: $S = \cup_{j=1}^{j=N} \{e_i =< x_i, y_i, j > |\ x_i \in \mathcal{X};\ y_i \in \mathcal{Y}_j\}$
1: $D_1 = \texttt{init}(S)$
2: **for** $t = 1$ to T **do**
3: $\quad h^t = \texttt{WL}(S, D_t)$
4: $\quad$ Choose α_t
5: $\quad D_{t+1}(e_i, l) = \frac{D_t(e_i,l)\exp(-\alpha_t y_i[l] h^t(x_i,l))}{Z_t}$ $\{update\ distribution\}$
6: **end for**
7: **return** Classifier H defined by $H(x, l) = (\sum_{i=1}^{i=T} \alpha_t h^t(x, l))$, $1 \leq j \leq N$ The predicted label for task j is: $\grave{y} = argmax_{l \in \mathcal{Y}_j} H(x, l)$

Algorithm 3: MT-Adaboost.MH algorithm

4.8.1 Error Analysis of MT-Adaboost.MH

By following the same analysis that is done for MT-Adabosot we can show that the empirical error of MT-Adaboost.MH decreases exponentially with the number of boosting iterations:

The training error $\tilde{\epsilon}$ is calculated by using the hamming Δ error as follows:

$$\tilde{\epsilon} = \frac{1}{N} \sum_{<x_i,y_i,j> \in S} \frac{1}{m_j} |sign(H(x_i)) \Delta Y_i|,$$

where $H(x_i) = \{H(x_i, l); l \in \mathcal{Y}_j\}$

$$\tilde{\epsilon} = \sum_{<x_i,y_i,j>\in S} \sum_{l\in Y_j} \frac{1}{m_j \times K_j \times N} |sign(H(x_i,l)) \neq Y_i[l]|,$$

where N is the number of tasks, $K_j = |Y_j|$ is the number of class labels of task j, and m_j is the size of j's training set.

$$\tilde{\epsilon} \leq \sum_{<x_i,y_i,j>} \sum_{l\in Y_j} \frac{1}{m_j \times K_j \times N} \exp(-Y_i[l]H(x_i,l)) \tag{4.6}$$

Let $D(i,l)_{final}$ be the final distribution over the training data after learning, if we unveil the recursion of the weights update rule we get:

$$D(i,l)_{final} = \frac{\exp(-Y_i[l]H(x_i,l))}{m_j \times k_j \times N \prod Z_t},$$

Using D_{final} in equation 4.6 gives

$$\tilde{\epsilon} \leq \sum_{<x_i,y_i,j>} \sum_{l\in Y_j} D_{final}(i,l) \prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} Z_t$$

4.8.2 The Weak Learner: Multi-Task Multi-Class Stump

For multi-class multi-task weak learners (`MC-2T-stump`) we propose the same structure as `2T-stump`, a.k.a, two levels decision tree with three nodes. The difference is that, each node is a **multi-class** decision stump for one of the N tasks. Those stumps have real valued predictions for multi-class / multi-label problems. They were introduced in the Boostexter system Schapire and Singer [2000].

Therefore, a node in `MC-2T-stump` is a multi-class stump for a certain task.It takes an example x as input and it give a confidence value for each possible class

label in its task. $h : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{R}$

$$h(x,l) = \begin{cases} c_{0l} & \text{if } TEST(x) \\ c_{1l} & \text{if } !TEST(x) \end{cases} .$$

For multi-class setting the predicted label is: $argmax_{l \in \mathcal{Y}} h(x,l)$. Let $X_0 = \{x \in X; TEST(x)\}$, $X_1 = \{x \in X; !TEST(x)\}$.

For an example x, `MC-2T-stump` predicts labels for two tasks and abstains for the rest. Let $S_0^j \subset S$ denotes the examples of task j for which an `MC-2T-stump` abstains. $\forall e =< x, Y, j > \in S_0^j, l \in \mathcal{Y}_j : H(x,l) = 0$ For a given distribution D, example (x, Y, j) and label $l \in \mathcal{Y}_j$ we calculate the values:

$$W_b^{kl} = \sum_{i \notin S_j^0} D(i,l)[[x_i \in X_k \wedge Y_i[l] = b]],$$

where $k \in \{0,1\}$ and $b \in \{-1,+1\}$

$$W_0^{kl} = \sum_{i \in S_j^0} D(i,l)$$

As for single task case Schapire and Singer [1999], it can be shown that choosing the output of the stump as: $c_{kl} = 0.5 \ln(\frac{W_+^{kl}}{W_-^{kl}})$, minimizes Z. And by setting a to 1 the resulted Z will be:

$$Z = \sum_j^N \sum_{k \in \{0,1\}} \sum_{l \in \mathcal{Y}_j} W_0^{kl} + 2\sqrt{W_+^{kl} W_-^{kl}}$$

The same learning algorithms proposed for `2T-stump` are used to learn `MC-2T-stump`. namely `N-Best`, `N-Best-Per-Task`.

4.9 Summary

Adaboost with stumps has been extended to fit the settings of multi-task. MT-Adaboost with `2T-stumps` is a boosting algorithm able to learn multiple tasks without neither restrictions nor a priori assumptions. MT-Adaboost.MH; an adaptation to multi-class multi-task setting has been presented along with the corresponding multi-task weak learners.

In the sequel we continue with boosting for multi-task but with different weak learner, it is a multi-task decision tree that we call `MT-DT`. We address the learning algorithm of `MT-DT` by proving a criterion which guides the construction of the multi-task tree. The proposed criterion makes use of the data from several tasks at each step of the tree learning. We also propose another adaptation of Adaboost to multi-class, which is this time inspired from Adaboost.M1; a straightforward adaptation of Adaboost to multi-class and we justify our choice of this algorithm which is better adapted to be combined with `MT-DTs`.

4.10 Multi-task Adaboost with `MT-DTs`

Decision tree learning is based on the entropy-based criteria , in particular, on the quantity of the mutual dependency between two random variables, the label variable $Y \in \mathcal{Y}$ and the observation attribute a which is one of the attributes of an input vector $x \in \mathcal{X}$. The information gain denoted $IG(Y;a)$ can be expressed as follows

$$IG(Y;a) = H(Y) - H(Y|a), \tag{4.7}$$

where $H(Y) = -\sum_{y\in\mathcal{Y}} p(y)\log p(y)$ is the marginal entropy of label set $\mathcal{Y}$ and $H(Y|a) = \sum_v p(v)H(Y|a=v)$ is the conditional entropy of Y knowing a.

Assume now we cope with N tasks with the corresponding label sets $\mathcal{Y}_1, \ldots,$ $\mathcal{Y}_N$, respectively. For learning the `MT-DT`, the baseline approach is to treat all

the tasks together by concatenating the label sets, denoted as $\oplus_{j=1}^{N}\mathcal{Y}_j$. The concatenated task takes as input a sample $S = \{< x_i, y_i > |\ x_i \in \mathcal{X},\ y_i = f(x_i) \in \oplus_{j=1}^{N}\mathcal{Y}_j,\ 1 \leq i \leq m\}$. It can use the *joint information gain* for learning decision rules, defined as $IG_J = IG(\oplus_{j=1}^{N}Y_j; a)$. As an alternative to IG_J, we could use the unweighted sum of individual task information gains, $IG_U = \sum_{j=1}^{T} IG(Y_j; a)$. Evaluations however show that IG_U fails to improve over IG_J. Instead, we prove below that IG_J is equivalent to the weighted sum of individual task information gains and infer an IG criterion superior to IG_J. The novel IG criterion, denoted IG_M, takes the maximum value among the individual IGs, $IG_M = max\{IG(Y_j; a), j = 1, \ldots, N\}$.

We first recall the generalized grouping feature of the entropy Gray [2010] in the following lemma. It establishes a relationship between the entropy of an entire set of values and the entropies of its disjoint subsets.

Lemma 4.4. *For $q_{kj} \geq 0$, such that $\sum_{k=1}^{n}\sum_{j=1}^{m} q_{kj} = 1, p_k = \sum_{j=1}^{m} q_{kj}, \forall k = 1, \ldots, n$, the following holds*

$$H(q_{11}, \ldots, q_{1m}, q_{21}, \ldots, q_{2m}, \ldots, q_{n1}, \ldots, q_{nm}) = \tag{4.8}$$

$$H(p_1, \ldots, p_n) + \sum p_k H\left(\frac{q_{k1}}{p_k}, \ldots, \frac{q_{km}}{p_k}\right), p_k > 0, \forall k. \tag{4.9}$$

Using Lemma 1, we can prove the following theorem on the relationship between the joint information gain $IG(\oplus_{j=1}^{N}Y_j; a)$ of the full task set and of the individual tasks $IG(Y_j; a), j = 1, \ldots, N$.

Theorem 4.5. *For N tasks with the class sets $\mathcal{Y}_1, \ldots, \mathcal{Y}_N$, let p_j denote the fraction of task j in the full dataset, $p_j = \frac{|S_j|}{\sum_{j=1}^{N}|S_j|}, j = 1, \ldots, N$, $\sum_{j=1}^{N} p_j = 1$. Then we have*

$$IG(\oplus_{j=1}^{N}Y_j; a) = \sum_{j=1}^{N} p_j IG(Y_j; a) \leq max(IG(Y_1; a), \ldots, IG(Y_N; a)). \tag{4.10}$$

Proof. First, we use Lemma 1 to develop the entropy term $H(\oplus_{j=1}^{N}Y_j)$ of the

information gain (4.7). We have

$$H(\oplus_{j=1}^{N} Y_j) = H(p_1, \ldots, p_N) + \sum_{j=1}^{N} p_j H(Y_j), \tag{4.11}$$

where $\sum_{j=1}^{N} p_j = 1$.

Second, we develop the conditional entropy term in (4.7), as follows

$$\begin{aligned}
H(\oplus_{j=1}^{N} Y_j|X) &= \sum_{x} p(x) H(\oplus_{j=1}^{N} Y_j|a = v) && (4.12)\\
&= \sum_{v} p(v) \left(H(p_1, \ldots, p_N) + \sum_{j=1}^{N} p_j H(Y_j|a = v) \right) && (4.13)\\
&= H(p_1, \ldots, p_N) + \sum_{j=1}^{N} p_j \sum_{v} p(v) H(Y_j|a = v) && (4.14)\\
&= H(p_1, \ldots, p_N) + \sum_{j=1}^{N} p_j H(Y_j|a). && (4.15)
\end{aligned}$$

Now we combine the entropy (4.11) and the conditional entropy (4.15) terms to evaluate the joint information gain $IG(\oplus_{j=1}^{N} Y_j; a)$. We obtain

$$\begin{aligned}
IG(\oplus_{j=1}^{N} Y_j; a) &= H(\oplus_{j=1}^{N} Y_j) - H(\oplus_{j=1}^{N} Y_j|a) && (4.16)\\
&= \sum_{j=1}^{N} p_j IG(Y_j; a) && (4.17)\\
&\leq \sum_{j=1}^{N} p_j max(IG(Y_1; a), \ldots, IG(Y_N; a)) && (4.18)\\
&= max(IG(Y_1; a), \ldots, IG(Y_N; a)). && (4.19)
\end{aligned}$$

This completes the proof of the theorem.

Theorem 1 says that criterion IG_M for the decision tree learning in the multi-task case is superior to the joint one IG_J. It suggests that using the maximum

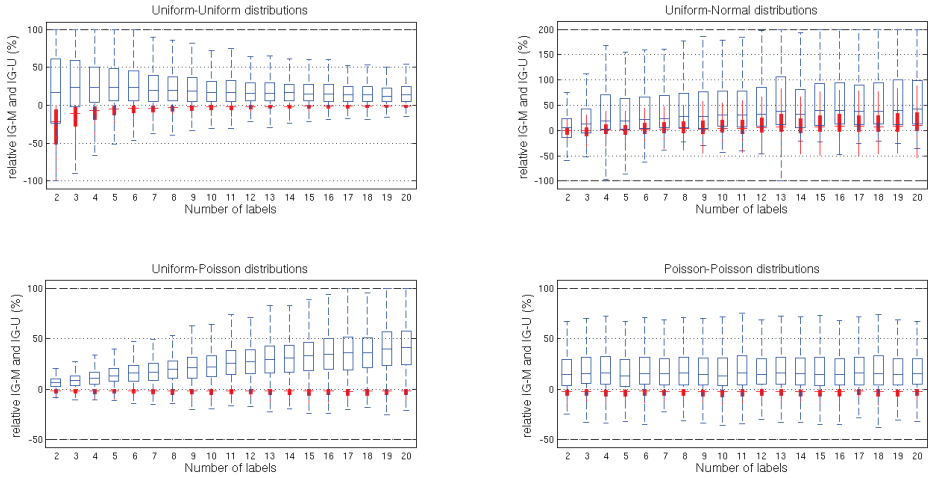


Figure 4.2: Information gain for synthetic two-task datasets. The relative values of IG_M (in blue) and IG_U (in red).

information gain among individual tasks can learn a better decision rule than one using the full data set.

Figure 4.10 compares three criteria IG_U, IG_J and IG_M for some randomly generated two-task datasets. Two label sets are generated by sampling from the Uniform, Normal (with $\mu = 0$, $\sigma = 1$) and Poisson ($\lambda = 1$) distributions; the number of labels in the two sets vary from 2 to 20. Attributes values are sampled from uniform distributions in all cases. We measure the relative values of IG_M and IG_U with respect to IG_J. In all cases, we report the median, the upper and lower percentiles, and the whiskers over 100 runs. As the figure shows, IG_M yields on average up to 42% more of information gain than IG_J, with the minimal gain in the case of two Uniform distributions.

The learning algorithm for `MT-DT` applies one of proposed information gain criteria to the available training set S:

$$MTIG(S) \equiv (\mathbf{a}^*, v^*) = max_{a,v} IG_*(S),$$

where a is an attribute in feature space $\mathcal{X}$ of instances x_i in set S, value v is

a value of a and pair (a^*,v^*) yields the optimal split of set S by the test rule $a^* \leq v^*$, and IG_* refers to IG_J, IG_U or IG_M.

The pseudo code of the MT-DT algorithm is presented in Algorithm 4. The algorithm makes a call to a function $MTIG$ which returns the node with rule $a \leq v$ that maximizes a given information gain on a multi-task training set S, Then it gets subsets S_1, S_2 resulting from splitting S on the chosen node. At each node the algorithm adds decision leaves for the tasks having no items in the subset or having items with the same label. Then, it calls recursively the procedure on each of subsets. In the evaluation section, we test three versions of the IG criterion introduced before, IG_J, IG_U and IG_M. It is worth noting that we can limit the depth of the trees by modifying the stopping criterion, instead of stopping the growth of a certain branch when we have homogenous labels for all tasks in the subspace corresponding to that branch, we can stop when we exceed a threshold. For instance, when 80% of the examples are from the same labels. This should not be an issue as long as we are using an ensemble of trees learned by a boosting algorithm.

4.10.1 Multi-Task Multi-Class Adaboost

In the previous section we developed a novel technique for learning `MT-DT`'s with an improved information gain criterion. To avoid all disadvantages of the decision trees such as overfitting, in this section we proceed by plugging the `MT-DT`'s in the boosting framework.

We adapt Adaboost.M1 which was introduced in Schapire and Singer [1999]. We preferred M1 to MH or other multi-class boosting algorithm because it is the most straightforward extention to binary Adaboost and it is fast. It has however a drawback; it puts strong requirement on the weak learner; actually, it requires the classification error of the weak classifier to be less than 0.5 w.r.t. to the current weight distribution, regardless the number of class labels. Some weak learners, such as stumps, are unable to satisfy such a strong boosting condition.

Require: $S = \cup_{j=1}^{N}\{e_i =< x_i, y_i, j > |\ x_i \in \mathcal{X};\ y_i \in \mathcal{Y}_j\}$
Require: $MTIG$: multi-task information gain criterion
1: `res = []` {Will contain the chosen node and early decision leaves, if any}
2: **for** $j = 1$ to N **do**
3: **if** task j's examples (S_j) has all the same label **or** $S_j = \emptyset$ **then**
4: Add to `res` a leaf for task j and label y. {y is either the unique label of S_j in case it is homogeneous or it is the majority label of its parent subset in case $S_j = \emptyset$}
5: $S = S \setminus S_j$
6: **end if**
7: **end for**
8: Get the `bestnode` rule $(a, v) = MTIG(S)$ which maximizes the information gain
9: Call `split`$(S, a, v))$
10: Get back $[S_1, S_2]$, two subsets resulted from splitting S based on `bestnode`
11: Add `bestnode` to `res`
12: Call recursively the algorithm on S_1 and S_2 to get the children of `res`
13: **return** `res`

Algorithm 4: `MT-DT` algorithm.

We choose multi-task decision tree as a weak learner. Normally, decision trees perform better than stumps and they can achieve a classification error lower than 0.5 on multi-class problems which makes them suitable as weak learners for Adaboost.M1.

The proposed Multi-Task Adaboost algorithm (`MTAA`) is presented in Algorithm 5. T is the number of boosting iterations; `init` is a procedure to initialize the distribution D_1 over S; and `WL` is a weak learner that returns an `MT-DT` given as input a sample S and a distribution D over S. The final output is a multi-task classifier H from $\mathcal{X}$ into $\mathcal{Y}_1 \times \ldots \times \mathcal{Y}_N$. As in single task boosting algorithms, `MTAA` calls `WL` repeatedly in a series of rounds. On each round t, the algorithm provides `WL` with the current distribution D_t and the training sample S, in return `WL` learns a classifier $h_t : \mathcal{X} \to \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_N$ which minimizes the training error on S with respect to D_t. The distribution D_{t+1} is then calculated from D_t and h_t as follows. Correctly classified examples by h_t will have their weights multiplied

by $0 \leq \beta_t \leq 1$ (i.e., decreased), and the weights of misclassified examples will be left unchanged. Finally, the weights are renormalized by using the normalization constant Z_t.

The final classifier H for a given task j is a weighted vote of the weak classifiers' predictions for this task. The weight given to hypothesis h_t is defined to be $ln(1/\beta_t)$ so that greater weight is given to hypotheses with lower error. `MTAA` has the same theoretical properties of Adaboost.M1, that is, if the weak hypotheses have error only slightly better than 1/2, then the (training) error of the final hypothesis H drops to zero exponentially fast in function to the number of boosting iterations T.

4.11 Random Forests

Random Forests Breiman [2001] is an ensemble learning method that consists of aggregating a forest of decision trees. The term came from random decision forests that was proposed by Ho [1998]. The method combines Breiman's "bagging" idea and the random selection of features, introduced independently by Ho [1998] and Amit and Y [1997].To classify a new data point, get the prediction of each tree on the data point. The forest chooses the prediction having the most votes (over all the trees in the forest). Each tree is learned as follows:

- If the number of training examples is m, sample m' examples at random - but with replacement, from the original data. This sample will be the training set for growing the tree.

- If there are F features, a pre-specified number $f << F$ is used s.t. at each node, f features are randomly selected, then the best split on these f is used to split the node. The value of f is held constant during the forest growing. Each tree is grown to the largest extent possible. There is no pruning.

In Breiman [2001], it was shown that the forest error rate depends on two things:

- The correlation between any two trees in the forest. Increasing the correlation increases the forest error rate.
- The strength of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.

Reducing f reduces both the correlation and the strength. Increasing it increases both. Somewhere in between is an "optimal" range of f. This is the only adjustable parameter to which random forests is somewhat sensitive. Random Forests features include:

- It is fast and thus it runs efficiently on large data sets.
- It can handle large number of features.
- It gives estimates of what variables are important in the classification.

We make use of RFs as a method to learn and aggregate our `MT-DTs` instead of Adaboost with C4.5 algorithm. In Chapter 5 we report some results comparing `MT-DTs` learned by random forests algorithms and those learned by Adaboost.

4.12 Summary

We proposed an adaptation of decision tree learning to the multi-task setting, with the following important contributions. First, we developed multi-task decision trees to deal with multi-class tasks with no label correspondence. The criterion to learn the decision rules makes use of the data from several tasks at each step of the decision tree learning, thus enabling to capture any degree of relatedness between the tasks. We then feature an important property of information gain rule when

working with multiple tasks. This enabled us derive the new information gain criterion for learning decision trees in the multi-task setting. We also modified MT-Adaboost to cope with multi-class problems. Next chapter is dedicated to validate the proposed methods by series of experiments on synthetic and real large scale data sets.

Require: $S = \cup_{j=1}^{N}\{e_i =< x_i, y_i, j > |\ x_i \in \mathcal{X};\ y_i \in \mathcal{Y}_j\}$

$D_1 = \texttt{init}(S)$ *initialize distribution*
for $t = 1$ to T **do**
  $h^t = \texttt{WL}(S, D_t)$ {*train the weak learner and get an hypothesis* `MT-DT` }
  Calculate the error of h^t: $\epsilon_t = \sum_{j=1}^{N} \sum_{i:h_j^t(x_i)\neq y_i} D_j(x_i)$.
  if $\epsilon_t > 1/2$ **then**
    Set $T = t - 1$ and abort loop.
  end if
  $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$
  {*Update distribution:*}
  if $h_j^t(x_i) == y_i$ **then**
    $D_{t+1}(e_i) = \frac{D_t(e_i)\times\beta_t}{Z_t}$
  else
    $D_{t+1}(e_i) = \frac{D_t(e_i)}{Z_t}$
  end if
end for
{*Where* Z_t *is a normalization constant chosen so that* D_{t+1} *is a distribution*}

return Classifier H defined by:

$$H_j(x) = \arg\max_{y\in\mathcal{Y}_j}(\sum_{i=1}^{i=T}(\ln 1/\beta_t)),\ 1 \le j \le N$$

Algorithm 5: MT-Adaboost.

Chapter 5

Experiments

Contents

In this chapter, we report the results of the experiments we have done on our algorithms. We tested the algorithms on synthetic tasks generated from random Bayesian networks which define the relatedness patterns. We also conducted experiments on real world / large scale email data sets (Enron and ECML'06 spam filtering challenge). Other tests were done on MNIST character recognition data set to compare our weak learners and to compare our algorithm with the work of Novi Quadrianto [2010]. It should be noted that multi-task problems with different number of classes are not quite well addressed in the literature which makes it difficult to find other methods to compare with for certain scenarios.

5.1 Data Sets

5.1.1 Synthetic

We generate synthetically tasks with local relatedness patterns, by following the data generation technique described in Freno et al. [2010]. Each pattern is generated a random Bayesian network (BN) from which one can derive different but related probabilistic distributions. The BN is created by generating (a) a random (directed acyclic) graph, (b) a set of functions (with random parameters) characterizing the dependence of every node on each one of its parents in the graph, and (c) a set of functions (with randomly assigned parameters) defining the probability density of each node.

Figure 5.1.1 shows some examples of the local tasks relatedness generated using such method. In the plotted examples, the distributions feature cubic, exponential and linear correlation functions, with Beta, Gaussian and Laplacian densities. Using the random relatedness generator we generate three multi-task learning datasets. DS_1 consists of two tasks T_1 and T_2, having three and two labels, respectively. They are plotted in Figure reffig:mtsynthetic.a. We can see that the red class of T_1 is locally correlated with the light blue class of T_2;

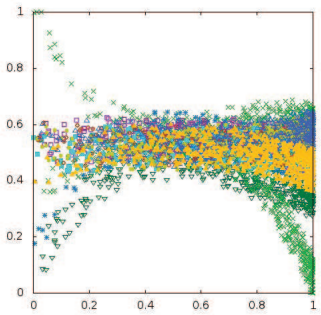
similarly, the green class is locally correlated with the violet. However the dark blue class of T_1 which is locally correlated with the violet in the upper part of its density and with the light blue in the lower part. The second dataset DS_2 is shown in Figure reffig:mtsynthetic.b with tasks being also locally correlated. For both DS_1 and DS_2 we generate 100 examples per class.

We increase both the number of tasks and labels in dataset DS_3 (fee Fig. 5.1.1); it consists of five tasks each having six labels, local relatedness patterns are generated randomly using the same method [1] , 25 examples are generated per each class in each task. Finally, random noise is added to the labels of all tasks as follows. For a certain example with label y we place a discrete probability distribution over the label set with 90% of mass concentrated over y and the rest distributed equally over the other labels. Then we sample the noisy label from this distribution. It should be noted that we generate tasks with different number of class labels on purpose, in order to test the proposed methods on configurations not addressed by prior-art methods.

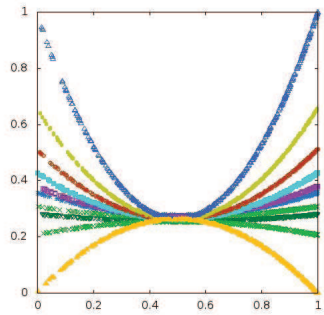
5.1.2 MNIST.

We use this dataset adapted to the multi-task setting because it was used in Novi Quadrianto [2010] and we follow their protocol. For the experiments, we consider multi-task learning problems with a number of tasks equal to 5, 7 or 10. We consider digits $\{6, 7, 8, 9, 0\}$, $\{4, 5, 6, 7, 8, 9, 0\}$ and $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 0\}$ for the 5-task, 7-task and 10-task problems, respectively. Every task is a binary classification task. For instance, in the 5-task problem, the first task has binary labels $\{+1, -1\}$, where label $+1$ means digit 6 and label -1 means digit 7, 8, 9 or 0; for the second task, label $+1$ means digit 9 and label -1 means other digits; and so on for other tasks. Similar one-against-all setting is also used for 7-task and 10-task problems. We use a small subset of the whole sample as training set

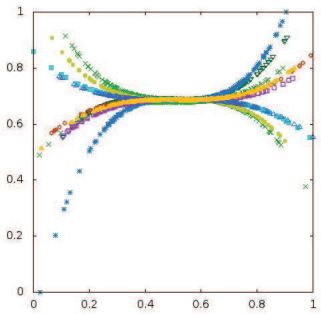
[1] We do not show DS_3's tasks superposed in a single plot because of low readability of a plot with 30 classes.



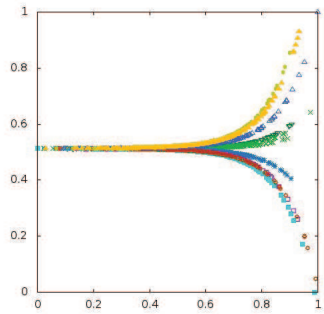
(a) A correlation pattern from beta-cubic distributions



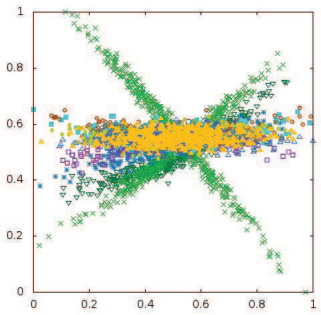
(b) A correlation pattern from beta-quadratic distributions



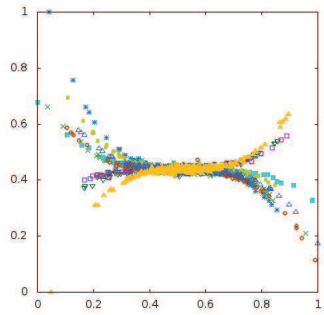
(c) A correlation pattern from gaussian-cubic distributions



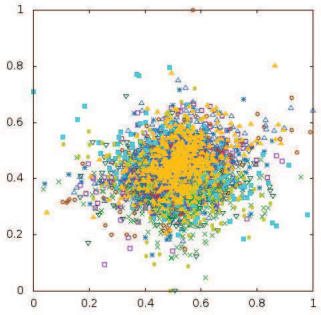
(d) A correlation pattern from gaussian-exponential distributions



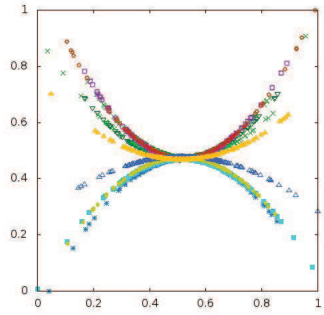
(e) A correlation pattern from gaussian-quadratic distributions



(f) A correlation pattern from laplace-cubic distributions

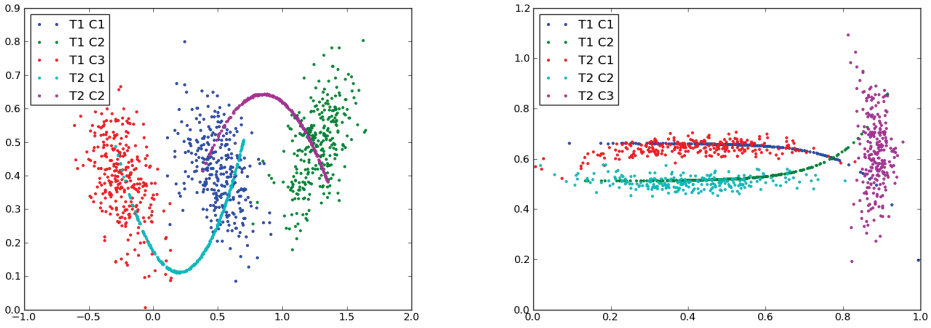


(g) A correlation pattern from laplace-linear distributions



(h) A correlation pattern from laplace-quadratic distributions

Figure 5.1: Tasks Relatedness Patterns for synthetic 2D data



(a) Two related multi-class tasks

(b) Two related multi-class tasks

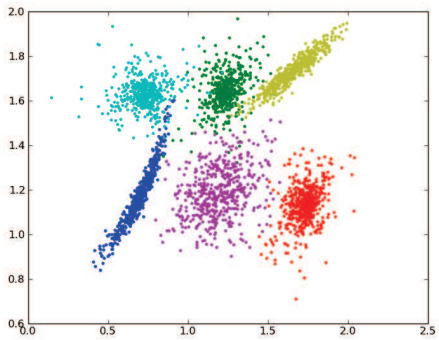
Figure 5.2: Two classification problems Dataset1 and Dataset2, each with two multi-class tasks.

to simulate the situation when we only have limited number of labeled examples. In the experiments, we draw 5 learning sets whose size is recalled in the tables according to the choices done in Novi Quadrianto [2010]. We present the average accuracy results over the 5-random runs where the accuracy is estimated on the fixed test set defined by the dataset creators.

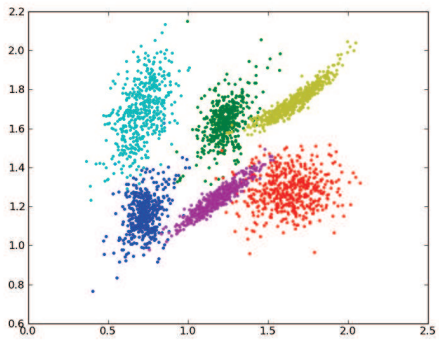
5.1.3 Enron.

Lawsuits involving companies and/or individuals have huge collections of documents varying from hard copy official documents to emails. A group of lawyers are engaged to mine those collections of millions of documents in order to decide which ones are *responsive* for a certain lawsuit. Case mining is costly, time consuming and critical since a single document might have an impact on the lawsuit. This kind of legal document collections is not easily available and even if they were, they would require considerable annotation efforts due to their huge size. To the best of our knowledge the Enron dataset[1] is the most known dataset of this kind and it is widely used by the machine learning community McCallum

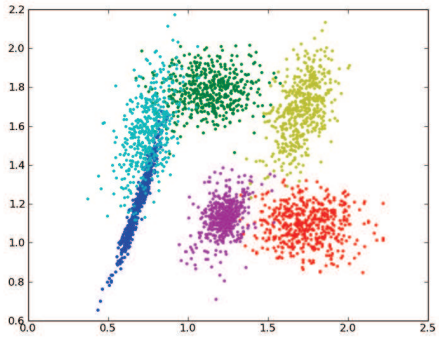
[1] http://www.cs.cmu.edu/~enron/



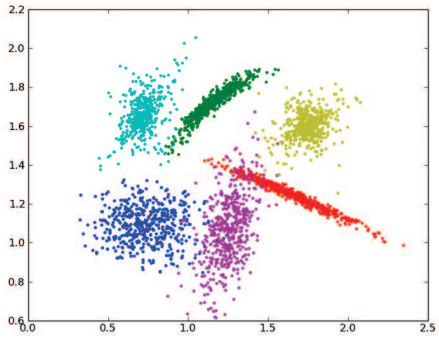
(a) Task1



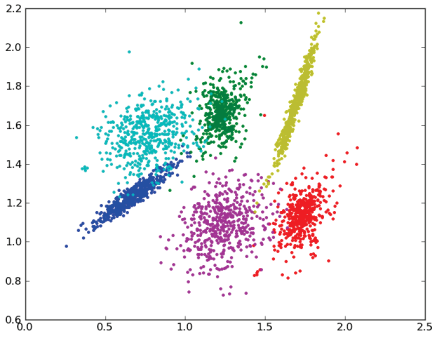
(b) Task2



(c) Task3



(d) Task4



(e) Task5

Figure 5.3: DS_3 consists of 5 related classification tasks each of which with 6 classes

et al. [2007]; Shetty [2005] and Bekkerman et al. [2004]. It contains all e-mails sent and received by some 150 accounts of the top management of Enron and spanning a period of several years

Annotations of the Enron dataset come from two different sources. The first is from the Department Of Justice of the United States DOJ[1], which has published a list of responsive emails used in the trials against the two CEO's of Enron. This set along with a manually annotated non-responsive emails constitute a binary classification task, *Responsive* Vs. *Non-responsive*, with total of 372 emails. The second annotated set comes from students of Berkeley University. Emails in this set are annotated by topic, for an average of 250 emails per topic. Five topics are used in our experiments: *Business*, *Legal*, *Influence*, *Arrangement* and *Personal*. Since the two sets are small, and they share a common knowledge (ex. a personal email is not likely to be a responsive email), so learning them simultaneously would be advantageous. It should be noted, that those two sets are disjoint, i.e., there are no examples provided with both annotations.

We used the textual features of Enron dataset along with the social features generated from the underlying social network (for more details, see Hovelynck and Chidlovskii [2010]. The main task is to discover responsive documents. We will try to improve performance on this task by considering the multi-task setting by also considering tasks from the topic annotated set: *Legal* Vs. *Personal* and *Business* Vs. *Arrangement*.

5.1.4 Spam Filtering

This dataset was used for the ECML/PKDD 2006 discovery challenge. It contains email inboxes of 15 users. Each inbox has 400 spam/ham emails. The necessity of multi-task and thus the gain of using its methods increase when we do not have enough data for each task,to simulate this case, we use 40 emails for training per each inbox, the rest are used for testing.

[1] http://www.usdoj.gov/enron/

Emails are encoded by standard bag-of-word vector representation. We consider each user as a task, the tasks are related because they all aim to filter out spam emails but they are not identical because each user has different preferences (a user might consider a certain email as spam whereas another user might not).

5.2 Empirical Studies of MTAA with 2T-Stumps

First, we describe in detail the datasets used, the preprocessing applied on them. Second, we compare the three weak learners `WL-Best-K`, `WL-Best-per-Task-K` and `WL-Sto-Best-K`. Third, we study the influence of the number of boosting rounds on our algorithms `MTAA`. Last, we report our experimental comparisons between `MTAA` with the closest prior art multi-task algorithm `MTL` Novi Quadrianto [2010]. Last, we give the performance of `MTAA` on the ENRON dataset.

5.2.1 Weak Learners Comparison

First, we have done experiments to compare the weak learners independently of the boosting algorithms. For this, we have considered the datasets MNIST-5, MNIST-7 and MNIST-10 and a fixed distribution over instances. Here, we only report our conclusions:

`WL-Best-K` is the simplest weak learner. When the parameter value K is not large enough, `WL-Best-K` may not find an optimal `2T-stump`. When the number of tasks increases, one may need a large value of K. For instance, $K = 30$ is enough for MNIST-5 while $K = 100$ is required for MNIST-10. This is because a greater value of K is necessary for every task to appear in a candidate root stump. Nevertheless, the output `2T-stump` is always not far from optimal;

`WL-Best-per-Task-K` find the optimal `2T-stump` in a large number of cases even with small values of the parameter K (the number of optimal root stumps

Tasks	Train (Test)	MTAA-30Best	MTAA-3Best-per-task	MTAA-30StoBest
1/-1	100 (10000)	94.15 ± 0.98	94.73 ± 0.88	93.95 ± 1.64
2/-2	100 (10000)	86.45 ± 1.55	86.01 ± 1.85	86.52 ± 1.08
3/-3	100 (10000)	84.60 ± 1.47	86.71 ± 1.62	86.94 ± 0.97
4/-4	100 (10000)	88.02 ± 1.25	86.19 ± 1.11	85.79 ± 1.32
5/-5	100 (10000)	83.36 ± 1.05	81.84 ± 1.47	84.01 ± 1.89
6/-6	100 (10000)	92.86 ± 0.86	92.92 ± 1.22	93.21 ± 1.96
7/-7	100 (10000)	91.98 ± 0.91	91.62 ± 1.69	90.11 ± 0.98
8/-8	100 (10000)	82.66 ± 1.74	80.73 ± 1.78	83.98 ± 1.67
9/-9	100 (10000)	84.27 ± 1.49	84.80 ± 1.58	84.56 ± 0.73
0/-0	300 (10000)	96.44 ± 0.46	96.55 ± 0.39	95.78 ± 0.37
Avg		88.48	88.21	88.49

Table 5.1: Comparison on the dataset MNIST-10 of MTAA with the weak learners WL-Best-K with $K = 30$, WL-Best-per-Task-K with $K = 3$, and WL-Sto-Best-K with $K = 30$.

per task). For instance $K = 5$ gives optimal results for all MNIST datasets.

WL-Sto-Best-K allows to introduce diversity in the choice of the root stumps, but with small values of K (the number of root stumps drawn randomly), it fails to output an optimal MTAA. Moreover, the output 2T-stump has often a lower score than the MTAA output by WL-Best-K for the same parameter value K.

WL-Best-per-Task-K is the more robust w.r.t. the choice of the parameter value. But WL-Best-K is the simplest and it often output good hypotheses. Thus, we also compare the weak learners when used in the MTAA algorithm. We give experimental results on the MNIST-10 dataset in Table 5.1. They show no significant differences between the weak learners when used in MTAA. For instance WL-Best-K with $K = 30$ gives good results when used in MTAA although non optimal as an independent weak learner.

Tasks	Train (Test)	$T = 100$	$T = 200$	$T = 500$	$T = 1000$
1/-1	100 (10000)	93.75 ± 1.31	94.15 ± 1.42	96.43 ± 0.78	96.24 ± 0.88
2/-2	100 (10000)	86.21 ± 1.65	86.45 ± 1.48	85.33 ± 0.54	84.52 ± 1.01
3/-3	100 (10000)	81.50 ± 2.31	84.60 ± 1.03	85.09 ± 0.89	85.39 ± 0.95
4/-4	100 (10000)	87.45 ± 2.61	88.02 ± 1.15	88.24 ± 1.25	88.75 ± 1.21
5/-5	100 (10000)	81.33 ± 1.11	83.36 ± 1.15	81.04 ± 2.46	82.2 ± 1.76
6/-6	100 (10000)	92.29 ± 2.86	92.86 ± 1.16	94.06 ± 1.25	94.14 ± 1.32
7/-7	100 (10000)	88.67 ± 1.32	91.98 ± 0.93	90.27 ± 0.64	90.29 ± 1.03
8/-8	100 (10000)	81.88 ± 2.2	82.66 ± 1.81	85.1 ± 0.97	85.13 ± 1.61
9/-9	100 (10000)	83.1 ± 1.56	84.27 ± 1.29	86.49 ± 1.82	86.47 ± 0.68
0/-0	300 (10000)	94.98 ± 1.01	96.44 ± 0.66	95.58 ± 0.41	95.57 ± 0.38
Avg		87.12	88.48	88.76	88.87

Table 5.2: Experimental results on the dataset MNIST-10 with `MTAA` when varying the number T of boosting iterations. The weak learner used is `WL-Best-K` with $K = 30$

5.2.2 Varying the Number of Boosting Iterations

We consider our algorithm `MTAA` with `WL-Best-K` chosen as weak learner with a parameter value chosen to be 30. We consider the MNIST-10 dataset and we let vary the number of boosting iterations. The experimental results are given in Table 5.2. They show that the accuracy increases with the number of boosting iterations as announced in Section 4.7.1.

5.2.3 Comparison between MTAA and MTL

We compare `MTAA` with `MTL` defined in Novi Quadrianto [2010] on the MNIST datasets. We also compare `MTAA` with (single-task) Adaboost. For `MTL`, we take the results from the paper. For Adaboost, we fix the number of boosting iterations to be $T = 100$. For `MTAA`, we fix the number of boosting iterations to be $T = 500$ and, as before we choose the weak learner `WL-Best-K` with K set to 30. For the initial distribution of Adaboost, we balance the probability mass between classes, which means that the probability of sampling an instance from a certain class is the same across classes. For `MTAA`, since we learn many tasks simultaneously, the probability mass is balanced between each pair (task, class).

The experimental results are presented in Table 5.3. Statistical significant improvements (according to `t-test` with $\alpha = 0.05$) are shown in bold face and

they show that MTAA outperforms both Adaboost and MTL. It is also worth noting that the standard deviation of accuracy results is lower for MTAA than for MTL, which shows the stability of boosting methods across the different runs.

5.2.4 MTAA on the ENRON Dataset

We consider in this paper the Enron dataset because it is a real world large scale dataset and it is associated with difficult learning tasks because the number of annotated examples is low. The different learning tasks have been defined independently by different communities. We consider three tasks: the case mining task, i.e. responsive Vs. non-responsive; a topic task legal Vs. personal; and another topic task business Vs. arrangement). The number of tasks is small but the tasks are difficult enough to study the performance of MTAA on this 3-task learning problem.

Since there are no available multi-task results on the Enron dataset, we compare MTAA with (single-task) Adaboost. No test set is available thus accuracy is estimated over 3-runs of 5-fold cross validation. For MTAA, the number of boosting iterations is set to 300 while the number of boosting iterations for Adaboost is set to 100. The weak learner used in MTAA is again WL-Best-K with K set to 9. The experimental results are shown in Table 5.8 in which statistical significant improvements are shown in bold face. The results on Enron emphasizes the claimed advantage behind learning multiple related tasks together.

5.3 Experiments with MT-DTs

In this section we present the results of the experiments conducted on MT-DTs. We first report the results of simple MT-DTs (without ensemble methods), then we report experimental results on boosted trees using MT-Adaboost.

Tasks	Train (Test)	Adaboost	MTL	MTAA
MNIST-3				
6/-6	25 (4949)	89.84 ± 0.37	83.86 ± 9.51	91.56 ± 3.21
7/-7	25 (4949)	85.25 ± 2.35	72.84 ± 15.77	83.35 ± 1.22
8/-8	25 (4949)	81.73 ± 3.21	66.77 ± 9.43	84.11 ± 2.02
9/-9	25 (4949)	73.215 ± 6.51	67.26 ± 12.65	76.85 ± 2.11
0/-0	150 (4949)	96.43 ± 0.28	96.60 ± 1.64	97.29 ± 0.62
Avg	-	85.29	77.74	**86.63**
MNIST-5				
4/-4	70 (6823)	86.11 ± 1.071	73.49 ± 6.77	87.52 ± 1.46
5/-5	70 (6823)	83.99 ± 2.92	70.10 ± 4.61	86.26 ± 1.03
6/-6	70 (6823)	92.23 ± 1.01	87.21 ± 2.77	93.02 ± 1.41
7/-7	70 (6823)	$87.97 \pm 0.0.51$	84.02 ± 3.69	90.05 ± 2.01
8/-8	70 (6823)	88.32 ± 0.13	76.97 ± 5.12	87.63 ± 1.38
9/-9	70 (6823)	78.09 ± 1.33	65.74 ± 10.15	80.31 ± 1.38
0/-0	210 (6823)	96.00 ± 0.81	96.56 ± 1.67	96.12 ± 0.61
Avg	-	87.53	79.16	**88.70**
MNIST-10				
1/-1	100 (10000)	94.62 ± 1.23	96.80 ± 1.91	96.43 ± 0.78
2/-2	100 (10000)	85.72 ± 0.58	69.95 ± 2.68	85.33 ± 0.54
3/-3	100 (10000)	85.71 ± 0.99	74.18 ± 5.54	85.09 ± 0.89
4/-4	100 (10000)	88.31 ± 0.64	71.76 ± 5.47	88.24 ± 1.25
5/-5	100 (10000)	82.34 ± 2.11	57.26 ± 2.72	81.04 ± 2.46
6/-6	100 (10000)	91.28 ± 0.4	80.54 ± 4.53	94.06 ± 1.25
7/-7	100 (10000)	90.20 ± 0.50	77.18 ± 9.43	90.27 ± 0.64
8/-8	100 (10000)	81.66 ± 2.13	65.85 ± 2.50	85.1 ± 0.97
9/-9	100 (10000)	81.42 ± 0.38	65.38 ± 6.09	86.49 ± 1.82
0/-0	300 (10000)	96.85 ± 0.35	97.81 ± 1.01	95.58 ± 0.41
Avg	-	87.77	75.67	**88.76**

Table 5.3: Comparison on the MNIST datasets of (single-task) Adaboost, MTL and MTAA.

Tasks	Train (Test)	Adaboost	MTAA
Responsive Vs. NonResponsive	299 (74)	90.49 ± 0.90	90.99 ± 2.74
Legal Vs. Personal	265 (66)	83.90 ± 0.75	84.44 ± 4.90
Business Vs. Arrangement	615 (153)	71.69 ± 1.5	74.32 ± 3.54
Avg		82.03	**83.25**

Table 5.4: Comparison on the Enron dataset of (single-task) Adaboost and MTAA

5.3.1 Results on Trees

In this section we show experimental results of MT-DTs learned either by C4.5 or by random forests while using IG_J, IG_U or IG_M criteria. The results on MT-DTs are compared to single task learning algorithms: C4.5, Adaboost.MH with stumps, and to the MTL algorithms MTAA with 2T-stumps.

In all experiments we report average results of 5 random shuffles of 5-fold cross validation, where each run consists of training on four folds and testing on the remaining one. In order to avoid large C4.5 trees, the algorithm stops at a certain branch when 80% of the examples are from the same class label.

Tables 5.5, 5.6 and 5.7 report the evaluation results the three synthetic datasets: DS_1,DS_2 and DS_3. We note that MT-DT with IG_M brings a significant improvement over C4.5. While IG_J and IG_U behave comparably to C4.5.

For single task C4.5 and MT-DT we did not use boosting, because the synthetic data sets are small dimensional (2D), thus, it was not necessary to learn multiple trees to fit the data. However, for weaker classifiers: stumps, random forests, 2T-stumps, and multi-task random forests we used boosting or random forests accordingly. The size of the random forests was to 20 by CV. However, the number of boosting iterations for stump based methods was set to 10 by CV. The parameter N-Best was set to $2 \times N$, where N is the number of tasks and N-Best-Per-Task was set to 2.

The results of experiments on the Enron data set are reported in Table 5.8.

Single Task Algorithms					
	AMH		M1C45	RF	
T1	71.86 ± 4.45		90.75 ± 0.08	87.88 ± 0.45	
T2	67.27 ± 5.96		83.74 ± 0.55	87.64 ± 0.23	
Avg	69.57		87.24	87.76	
Multi Task Learning with `2T-stumps` and `MT-DTs`					
	MTMH NB	MTMHN NBPT	MTM1 IG_J	MTM1 IG_U	MT IG_M
T1	90.17 ± 0.17	90.51 ± 0.07	87.97 ± 0.80	89.88 ± 0.06	90.77 ± 0.07
T2	88.70 ± 0.77	88.57 ± 0.64	88.45 ± 1.56	88.58 ± 1.50	88.371 ± 0.26
Avg	89.44	89.54	88.21	89.23	89.57
`MT-DTs` with Random Forest					
	MTRF IG_J		MTRF IG_U	MTRF IG_M	
T1	88.33 ± 0.46		87.59 ± 0.61	87.75 ± 0.43	
T2	88.14 ± 0.53		88.61 ± 0.40	88.20 ± 0.37	
Avg	88.24		88.10	87.97	

Table 5.5: Comparison between all single task and multi-task algorithms on the first DS_1 synthetic dataset in Fig. 5.1.1-a. MH: AdaboostMH, M1C45: Adaboost.M1 /w C45 trees, RF: random forest, MTMH NB: MT-Adaboost.MH /w N-best `2T-stump`, MTMH NBPT: MT-Adaboost.MH /w N-best per task, MTM1 IG_x: MT-Adaboost with `MT-DT` and IG_x as criterion.

Single Task Algorithms					
	AMH		M1C45	RF	
T1	67.00 ± 4.59		86.22 ± 0.44	85.16 ± 0.46	
T2	71.00 ± 4.32		89.60 ± 0.16	89.00 ± 0.54	
Avg	68.98		87.91	87.07	
Multi Task Learning with `2T-stumps` and `MT-DTs`					
	MTMH NB	MTMHN NBPT	MTM1 IG_J	MTM1 IG_U	MT IG_M
T1	87.39 ± 0.11	87.08 ± 0.57	86.12 ± 0.05	86.078 ± 0.04	87.14 ± 0.07
T2	88.82 ± 0.10	88.94 ± 0.18	89.07 ± 0.33	89.26 ± 0.38	89.36 ± 0.30
Avg	88.10	88.01	87.59	87.67	88.25
`MT-DTs` with Random Forest					
	MTRF IG_J		MTRF IG_U	MTRF IG_M	
T1	85.23 ± 0.43		85.27 ± 0.47	85.93 ± 0.31	
T2	87.22 ± 0.21		86.96 ± 0.42	86.93 ± 0.27	
Avg	86.23		86.10	86.43	

Table 5.6: Comparison between all single task and multi-task algorithms on the second DS_2 synthetic dataset in Fig. 5.1.1-a. MH: AdaboostMH, M1C45: Adaboost.M1 /w C45 trees, RF: random forest, MTMH NB: MT-Adaboost.MH /w N-best `2T-stump`, MTMH NBPT: MT-Adaboost.MH /w N-best per task, MTM1 IG_x: MT-Adaboost with `MT-DT` and IG_x as criterion.

Single Task Algorithms					
	AMH		M1C45	RF	
T1	62.72 ± 3.05		78.63 ± 1.10	78.90 ± 0.93	
T2	67.14 ± 4.15		78.02 ± 0.51	75.58 ± 1.14	
T3	62.42 ± 4.50		76.91 ± 0.36	76.34 ± 1.58	
T4	48.83 ± 3.44		76.29 ± 0.94	75.83 ± 1.84	
T5	68.35 ± 1.67		76.10 ± 0.91	77.42 ± 2.65	
Avg	61.89		77.19	76.82	
Multi Task Learning with `2T-stumps` and `MT-DTs`					
	MTMH NB	MTMHN NBPT	MTM1 IG_J	MTM1 IG_U	MT IG_M
T1	84.37 ± 0.28	74.15 ± 4.26	82.28 ± 0.87	81.23 ± 0.47	83.33 ± 1.15
T2	83.49 ± 0.71	68.43 ± 4.53	79.00 ± 0.39	77.41 ± 0.26	79.96 ± 0.85
T3	82.77 ± 0.59	76.39 ± 5.09	80.02 ± 0.66	77.71 ± 0.91	80.72 ± 0.80
T4	83.36 ± 0.23	76.43 ± 4.58	78.78 ± 0.64	77.23 ± 0.89	79.20 ± 1.41
T5	17.20 ± 1.07	59.87 ± 1.52	77.24 ± 1.02	76.09 ± 0.16	78.92 ± 0.53
Avg	70.240	71.06	79.46	77.93	80.43
`MT-DTs` with Random Forest					
	MTRF IG_J		MTRF IG_U	MTRF IG_M	
T1	80.51 ± 1.22		80.81 ± 0.74	80.84 ± 1.35	
T2	80.51 ± 1.15		77.53 ± 1.32	78.53 ± 1.71	
T3	78.16 ± 0.49		77.20 ± 1.35	79.30 ± 1.52	
T4	79.89 ± 1.54		79.72 ± 1.41	79.92 ± 1.53	
T5	77.61 ± 0.88		77.52 ± 1.24	77.74 ± 1.15	
Avg	79.34		78.55	79.27	

Table 5.7: Comparison between all single task and multi-task algorithms on the third DS_3 synthetic dataset in Fig. 5.1.1-a. MH: AdaboostMH, M1C45: Adaboost.M1 /w C45 trees, RF: random forest, MTMH NB: MT-Adaboost.MH /w N-best `2T-stump`, MTMH NBPT: MT-Adaboost.MH /w N-best per task, MTM1 IG_x: MT-Adaboost with `MT-DT` and IG_x as criterion.

Tasks	Train (Test)	C4.5	IG_J	IG_U	IG_M
Res Vs. NonRes	299 (74)	80.32 ± 1.87	80.59 ± 2.23	80.01 ± 3.11	**81.81±1.16**
5 Topics	265 (66)	43.12 ± 1.03	43.65 ± 1.77	44.12 ± 0.42	**48.11±0.023**
Avg		61.72	62.12	62.066	**64.96**

Table 5.8: Average classification accuracy on Enron tasks.

Tasks	Train (Test)	C4.5	IG_J	IG_U	IG_M
User-1	320 (80)	86.45 ± 1.23	86.19 ± 1.14	86.00 ± 1.88	**87.65±3.42**
User-2	320 (80)	85.13 ± 2.16	85.53 ± 2.22	85.07 ± 3.16	**88.93±3.44**
User-3	320 (80)	88.03 ± 2.11	88.22 ± 2.56	**88.52±1.33**	88.19 ± 2.51
Avg		86.54	86.65	86.53	**88.26**

Table 5.9: Average classification accuracy on three ECML'06 user inboxes.

It shows a superiority of IG_M over other MT-DT criteria in accuracy values. However, learning tasks simultaneously does not bring the same improvement to all tasks, some tasks tend to benefit more from multi-task learning than others. Similarly, the results on ECML'06 data (see Table 5.9) show that more difficult tasks (tasks with a lower accuracy) have a larger margin of improvement. In other words, the transfer of knowledge between tasks is not symmetric, easier tasks provide more knowledge to the more difficult ones.

5.3.2 Results on Boosted Trees

In the previous section we experimentally validated the advantage of learning related tasks simultaneously, by using multi-task information gain criteria, in particular IG_M. In this section we compare boosted `MT-DT`'s to the boosted C4.5 trees. We use Adaboost.M1 Schapire and Singer [1999] and MT-Adaboost (see algorithm 5) as boosters for C4.5 and for `MT-DT` respectively. Both algorithms have only one parameter, the number of boosting iterations which we set equally to 20. Table 5.10 reports the average values of classification accuracy over three random runs for Enron dataset. With boosted trees we observe an accuracy

Tasks	Train (Test)	Adaboost C4.5	MT-Adaboost IG_J	MT-Adaboost IG_U	MT-Adaboost IG_M
Responsive Vs. NonResponsive	299 (74)	85.10 ± 1.21	84.66 ± 2.15	84.52 ± 1.2	**86.01±1.53**
5 Topics	265 (66)	51.34 ± 0.43	52.89 ± 0.87	52.17 ± 0.74	**57.11±0.02**
Avg		68.22	68.78	68.35	**71.65**

Table 5.10: Average classification accuracy of boosted trees on Enron tasks.

improvement similar to simple trees. Namely, MT-Adaboost+MT-DT is significantly better than Adaboost+C4.5; also the most difficult tasks enjoy a larger margin of improvement.

5.4 Summary

In this chapter, we presented the data sets we used to conduct our experimental studies. We then reported the results of the experiments we have done on our algorithms. We compared our weak learners for `2T-stump`, we also compared different IG criteria for `MT-DT`. Comparisons with single task boosting algorithms were done on synthetic data sets as well as well as large scale email data sets, they showed that our approach outperforms single task learning. On MNIST character recognition data set we compared our algorithm with the work of Novi Quadrianto [2010]. MTL works in the literature do not address tasks with different number of class labels. We hope in the future we will be able to compare our approach with more algorithms from the literature.

Chapter 6

Perspective and Conclusion

This thesis addressed the subject of multi-task learning in order to provide an approach for the configuration in which the tasks do not share neither their labels nor their examples. We wanted to cope with the problem without making prior assumptions about task relatedness patterns. Our contribution lies in the ensemble learning framework. Two learning algorithms were proposed in this framework, both consists of devising novel multi-task weak classifier along with its learning algorithms and adapt ensemble learning algorithms to fit multi-task setting.

First, we introduced `2T-stumps` as weak classifiers that abstain and we defined weak learners. We adapted Adaboost and defined `MTAA` as a multi-task learning algorithm. We gave empirical evidence that `MTAA` achieves good results and allows to capture relations between tasks without explicit priors. We think that more empirical validation and more theoretical work is needed in the case of very large number of tasks. Also, we should relate our work on Adaboost for multi-task learning with the recent work of Mukherjee and Schapire Mukherjee and Schapire [2010] on multi class boosting.

We then, proposed an adaptation of decision tree learning to the multi-task setting, with the following important contributions. First, we developed multi-task decision trees to deal with multi-class tasks with no label correspondence. The criterion to learn the decision rules makes use of the data from several tasks

at each step of the decision tree learning, thus enabling to capture any degree of relatedness between the tasks. We then featured an important property of information gain rule when working with multiple tasks. This enabled us derive the new information gain criterion for learning decision trees in the multi-task setting. We also modified MT-Adaboost to cope with multi-task multi-class problems.

We finally validated the proposed methods by series of experiments on related tasks synthetically generated from Bayesian networks, in addition to real world large scale data sets.

We are currently conducting experiments on web pages categorization. We have extracted two sets of annotated web pages, one set is annotated by Yahoo! web directory categories and the other by DMOZ web directory categories. Each set constitutes a task related but not identical to the other one. Our extracted data sets are available on http://mldata.org/repository/tags/data/web-pages/.

In a future work and on the algorithmic level, we aim to address two crucial questions for MTL. They are both about online learning. The first is how to integrate task-level online learning in our algorithms. In other words, the ability of the algorithm to incorporate new tasks into its learning process without the need to reconstruct the model from scratch. The second question concerns example-level online learning: In the case of very large or temporal data, it would be more suitable if the algorithm has the ability to incorporate new examples as soon as they are obtained instead of limiting the learning on a batch of training examples.

References

Wayne Iba Ai and Pat Langley. Induction of one-level decision trees. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 233–240. Morgan Kaufmann, 1992. 61

Yali Amit and Donald Geman Y. Shape quantization and recognition with randomized trees. *Neural Computation*, 9:1545–1588, 1997. 87

Cedric Archembeau, Shengbo Guo, and Onno Zoeter. Sparse Bayesian Multi-Task Learning. In *NIPS*, 2011. 1, 41

Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Multi-task feature learning. In *Advances in Neural Information Processing Systems 19*, 2006a. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.103.7734. 1, 44

Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Multi-task feature learning. In *NIPS*, pages 41–48, 2006b. 35

Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Multi-task feature learning. In *Advances in Neural Information Processing Systems 19*. MIT Press, 2007. 35

Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex multi-task feature learning. *Mach. Learn.*, 73(3):243–272, December 2008. ISSN 0885-6125. doi: 10.1007/s10994-007-5040-8. URL http://dx.doi.org/10.1007/s10994-007-5040-8. 44

Bart Bakker and Tom Heskes. Task clustering and gating for bayesian multitask learning. *Journal of Machine Learning Research*, 4:2003, 2003. 36

Ron Bekkerman, Andrew Mccallum, and Gary Huang. Automatic categorization of email into folders: Benchmark experiments on enron and sri corpora. In *Technical Report, Computer Science department, IR-418*, pages 4–6, 2004. 97

Shai Ben-david and Reba Schuller. Exploiting task relatedness for multiple task learning. In *Lecture Notes in Computer Science (2003)*, 2003. 36

Steffen Bickel, Michael Brckner, and Tobias Scheffer. Discriminative learning for differing training and test distributions. In *In ICML*, pages 81–88. ACM Press, 2007. 35

Edwin V. Bonilla, Kian Ming, A. Chai, and Christopher K. I. Williams. Multi-task gaussian process prediction, 2008. 35

Leo Breiman. Bagging predictors. In *Machine Learning*, pages 123–140, 1996. 60, 66

Leo Breiman. Prediction games and arcing algorithms, 1997. 60

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. 87, 88

Colin Campbell and Kristin P. Bennett. A linear programming approach to novelty detection, 2001. 60

Rich Caruana. Multitask learning: A knowledge-based source of inductive bias. In *the 10th International Conference on Machine Learning (ICML)*, pages 41–48, 1993. 37

Rich Caruana. *Multitask Learning*. PhD thesis, Carnegie Mellon University, 1997. 1, 30, 37

Olivier Chapelle, Pannagadatta Shivaswamy, Srinivas Vadrevu, Kilian Weinberger, Ya Zhang, and Belle Tseng. Multi-task learning for boosting with application to web search ranking. In *Proceedings of the 16th ACM SIGKDD international Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1189–1198, 2010a. 1

Olivier Chapelle, Srinivas Vadrevu, Kilian Weinberger, Pannagadatta Shivaswamy, Ya Zhang, and Belle Tseng. Multi-Task Learning for Boosting with Application to Web Search Ranking. In *SIGKDD*. ACM, 2010b. 45

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 160–167, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390177. URL http://dx.doi.org/10.1145/1390156.1390177. 44

Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine Learning*, pages 273–297, 1995. 14

Wenyuan Dai, Qiang Yang, Gui R. Xue, and Yong Yu. Boosting for transfer learning. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 193–200. ACM, 2007. ISBN 978-1-59593-793-3. doi: 10.1145/1273496.1273521. 1

Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. Self-taught clustering. In *ICML*, pages 200–207, 2008. 34

Hal Daumé, III and Daniel Marcu. Domain adaptation for statistical classifiers. *J. Artif. Int. Res.*, 26:101–126, May 2006. ISSN 1076-9757. 34

Jesse Davis and Pedro Domingos. Deep transfer via second-order markov logic. In *In Proceedings of the 26th International Conference on Machine Learning (ICML-09*, 2009. 35

Thomas G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, 2000. 61

Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *CoRR*, cs.AI/9501101, 1995. 66

Harris Drucker, Robert E. Schapire, and Patrice Simard. Boosting performance in neural networks. *IJPRAI*, 7(4):705–719, 1993. 60

Nigel Duffy and David Helmbold. Leveraging for regression. In *IN COLT'00*, pages 208–219. Morgan Kaufmann, 2000. 60

Theodoros Evgeniou and Massimiliano Pontil. Regularized multi–task learning. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference*

on Knowledge discovery and data mining, pages 109–117, New York, NY, USA, 2004. ACM. ISBN 1-58113-888-1. doi: 10.1145/1014052.1014067. URL http://dx.doi.org/10.1145/1014052.1014067. 1, 39, 45

Theodoros Evgeniou, Charles A. Micchelli, and Massimiliano Pontil. Learning Multiple Tasks with Kernel Methods. *Journal of Machine Learning Research*, 6:615–637, 2005. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.86.9708. 39

Jean Baptiste Faddoul, Boris Chidlovskii, Fabien Torre, and Rémi Gilleron. Boosting multi-task weak learners with applications to textual and social data. In *Proceedings of the Ninth International Conference on Machine Learning and Applications (ICMLA)*, pages 367–372, 2010. 2

Marcus Frean and Tom Downs. A simple cost function for boosting, 1998. 60

Antonino Freno, Edmondo Trentin, and Marco Gori. Kernel-based hybrid random fields for nonparametric density estimation. In *ECAI*, pages 427–432, 2010. 92

Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning (ICML)*, pages 148–156, 1996. 2, 25, 48

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of online learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997. 65, 66

Yoav Freund and Robert E. Schapire. A brief introduction to boosting. In *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1401–1406. Morgan Kaufmann, 1999a. 57

Yoav Freund and Robert E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, 1999b. 60

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:2000, 1998. 60

Jing Gao, Wei Fan, Jing Jiang, and Jiawei Han. Knowledge transfer via multiple model local structure mapping. In *In International Conference on Knowledge Discovery and Data Mining, Las Vegas, NV*, 2008. 35

Robert M. Gray. *Entropy and Information Theory*. Springer, 2010. 82

Adam J. Grove and Dale Schuurmans. Boosting in the Limit: Maximizing the Margin of Learned Ensembles. In *AAAI/IAAI*, pages 692–699, 1998. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.58.2398. 60

Silviu Guiasu and Abe Shenitzer. The principle of maximum entropy. *The Mathematical Intelligencer*, 7:42–48, 1985. ISSN 0343-6993. URL http://dx.doi.org/10.1007/BF03023004. 10.1007/BF03023004. 15

Jingrui He and Rick Lawrence. A Graph-based Framework for Multi-Task Multi-View Learning. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 25–32, New York, NY, USA, June 2011. ACM. 42

Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:832–844, 1998. ISSN 0162-8828. doi: http://doi.ieeecomputersociety.org/10.1109/34.709601. 61, 87

Matthijs Hovelynck and Boris Chidlovskii. Multi-modality in one-class classification. In *Proceedings of the 19th international conference on World wide web (WWW)*, pages 441–450, 2010. 97

Jiayuan Huang, Arthur Gretton, Bernhard Schlkopf, Alexander J. Smola, and Karsten M. Borgwardt. Correcting sample selection bias by unlabeled data. In *In NIPS*. MIT Press, 2007. 35

Laurent Jacob and Francis Bach. Clustered Multi-Task Learning: a Convex Formulation, 2008. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.143.9278. 40

Tony Jebara. Multi-task feature and kernel selection for svms. In *Proc. of ICML 2004*, 2004. 35

Jing Jiang and Chengxiang Zhai. Instance weighting for domain adaptation in nlp. In *In ACL 2007*, pages 264–271, 2007. 35

Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *J. ACM*, 41(1):67–95, January 1994. ISSN 0004-5411. doi: 10.1145/174644.174647. URL http://doi.acm.org/10.1145/174644.174647. 21, 59

Neil D. Lawrence and John C. Platt. Learning to learn with the informative vector machine. In *In Proceedings of the International Conference in Machine Learning*. Morgan Kaufmann, 2004. 35

Qiuhua Liu, Xuejun Liao, Hui L. Carin, Jason R. Stack, and Lawrence Carin. Semisupervised Multitask Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(6):1074–1086, 2009. ISSN 0162-8828. doi: 10.1109/TPAMI.2008.296. URL http://dx.doi.org/10.1109/TPAMI.2008.296. 41

Tie-Yan Liu, Yiming Yang, Hao Wan, Hua-Jun Zeng, Zheng Chen, and Wei-Ying Ma. Support vector machines classification with a very large-scale taxonomy. *SIGKDD Explor. Newsl.*, 7(1):36–43, June 2005. ISSN 1931-0145. doi: 10.1145/1089815.1089821. 1

Francis Maes. *Learning in Markov Decision Processes for Structured Prediction*. PhD thesis, Pierre and Marie Curie University, Computer Science Laboratory of Paris 6 (LIP6), October 2009. 14

Amin Mantrach and Jean-Michel Renders. Search engine for mailboxes based on cross-modal, topics and communities query expansion. In *Proceedings of European Conference on Information Retrieval (ECIR'2012)*, 2012. 2

Andrew McCallum, Xuerui Wang, and Andrés C. Emmanuel. Topic and role discovery in social networks with experiments on enron and academic email. *J. Artif. Int. Res.*, 30(1):249–272, 2007. 95

Lilyana Mihalkova, Tuyen Huynh, and Raymond J. Mooney. Mapping and revising markov logic networks for transfer learning. In *In Proceedings of the 22 nd National Conference on Artificial Intelligence (AAAI*, pages 608–614, 2007. 35

Tom M. Mitchell. The need for biases in learning generalizations. Technical report, CMU, 1980. 9

Indraneel Mukherjee and Robert Schapire. A theory of multiclass boosting. In *Proceedings of the Twenty-Fourth Annual Conference on Neural Information Processing Systems (NIPS)*, 2010. 109

Alex J. Novi Quadrianto. Multitask Learning without Label Correspondences. In *NIPS*, 2010. 1, 43, 92, 93, 95, 98, 100, 107

Sinno J. Pan and Qiang Yang. A survey on transfer learning. Technical report, Hong Kong University of Science and Technology, 2008. 1, 30, 32

Shibin Parameswaran and Kilian Q. Weinberger. Large Margin Multi-Task Metric Learning. In *NIPS*, December 2010. 38, 43

J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-238-0. 51, 62

Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y. Ng. Self-taught learning: Transfer learning from unlabeled data. In *Proceedings of the Twenty-fourth International Conference on Machine Learning*, 2007. 33

Michael T. Rosenstein, Zvika Marx, Leslie Pack Kaelbling, and Thomas G. Dietterich. To transfer or not to transfer. In *In NIPS05 Workshop, Inductive Transfer: 10 Years Later*, 2005. 36

Gunnar Rtsch, Ayhan Demiriz, and Kristin Bennett. Sparse regression ensembles in infinite and finite hypothesis spaces, 2000. 60

Gunnar Rtsch, Sebastian Mika, Bernhard Schlkopf, and Klaus-Robert Mller. Constructing boosting algorithms from svms: an application to one-class classification, 2002. 60

Robert E. Schapire. The strength of weak learnability. *Mach. Learn.*, 5(2): 197–227, July 1990. ISSN 0885-6125. doi: 10.1023/A:1022648800760. URL http://dx.doi.org/10.1023/A:1022648800760. 59

Robert E. Schapire. Using output codes to boost multiclass learning problems. In *ICML*, pages 313–321, 1997. 66

Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37, 1999. 2, 3, 15, 66, 69, 70, 71, 80, 85, 106

Robert E. Schapire and Yoram Singer. BoosTexter: A Boosting-based System for Text Categorization. In *Machine Learning*, volume 39, pages 135–168, 2000. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.1666. 25, 49, 76, 79

Jitesh Shetty. Discovering important nodes through graph entropy: The case of enron email database. In *Proceedings of the 3rd international workshop on Link discovery*, pages 74–81, 2005. 97

H. Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2): 227–244, October 2000. ISSN 03783758. doi: 10.1016/S0378-3758(00)00115-4. URL http://dx.doi.org/10.1016/S0378-3758(00)00115-4. 34

Matthew E. Taylor, Gregory Kuhlmann, and Peter Stone. Accelerating search with transferred heuristics, 2007. 36

Lisa Torrey. Transfer learning. In *Handbook of Research on Machine Learning Applications*, 2009. 32

L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27: 1134–1142, 1984. 11, 18, 59

Vladimir Vapnik. An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, 10(5):988–999, 1999. 11, 14

Xiaogang Wang, Cha Zhang, and Zhengyou Zhang. Boosted multi-task learning for face verification with applications to web image and video search. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 0: 142–149, 2009. doi: 10.1109/CVPRW.2009.5206736. URL http://dx.doi.org/10.1109/CVPRW.2009.5206736. 45

Zheng Wang, Yangqiu Song, and Changshui Zhang. Transferred dimensionality reduction. In *ECML/PKDD (2)*, pages 550–565, 2008. 34

Kilian Q. Weinberger and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. *J. Mach. Learn. Res.*, 10:207–244, June 2009. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1577069.1577078. 43

Ya Xue, Xuejun Liao, Lawrence Carin, and Balaji Krishnapuram. Multi-task learning for classification with dirichlet process priors. *Journal of Machine Learning Research*, 8, 2007. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.73.6069. 1, 40

Vincent W. Zheng, Sinno J. Pan, Qiang Yang, and Jeffrey J. Pan. Transferring Multi-device Localization Models using Latent Multi-task Learning, 2008. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.140.9253. 40