Analyse et prédiction d'erreurs critiques dans une application industrielle

Myriam Lopez

université de BORDEAUX

LABORATOIRE BORDELAIS DE RECHERCHE EN INFORMATIQUE LaBRI

THESE PRÉSENTÉE
POUR OBTENIR LE GRADE DE

DOCTEUR DE L'UNIVERSITÉ DE BORDEAUX

École Doctorale Mathématiques et Informatique
Spécialité Informatique

Par Myriam Lopez

ANALYSE ET PRÉDICTION D'ERREURS CRITIQUES DANS UNE APPLICATION INDUSTRIELLE

CRITICAL ERROR PREDICTION IN AN INDUSTRIAL APPLICATION

Sous la direction de : Marie Beurton-Aimar
Co-directeur : Sofian Maabout

Soutenue le 26 janvier 2023

Membres du jury :
Marie Beurton-Aimar MCF-HDR, LaBRI, Bordeaux ..... Directrice
Sofian Maabout MCF-HDR, LaBRI, Bordeaux ..... Co-Directeur
Ladjel Bellatreche Professeur, ENSMA, Poitiers ..... Rapporteur
Bernard Kamsu-Foguem Professeur, ENIT, Toulouse ..... Rapporteur
Karell Bertet MCF-HDR, L3i, La Rochelle ..... Examinatrice
Omar Boussaid Professeur, ERIC, Lyon ..... Examinateur
Richard Chbeir Professeur, LIUPPA, Pau ..... Président du jury

Membre invité :
Gayo Diallo Professeur, ISPED, Bordeaux ..... Invité, Co-Encadrant

Résumé

Titre : Analyse et prédiction d'erreurs critiques dans une application industrielle

La **maintenance prédictive** (MP) permet d'anticiper la survenue des pannes sur les équipements industriels au moyen de techniques d'**analyse de données** et d'**apprentissage automatique**. Dans ce contexte, les machines industrielles sont conçues pour fournir quotidiennement un ensemble de données aux formats variés. Dans un objectif de MP, ces travaux de recherche consistent à analyser les **données de fichiers *logs*** émises par les machines. Aussi, cette thèse présente un modèle de **classification supervisée**, dédié à la prédiction d'erreurs critiques. Dans cette optique, un historique d'erreurs critiques et non critiques a été collecté durant un an sur une flotte de machines de l'industrie du textile. Pour faciliter l'étiquetage des jeux de données, nous avons élaboré une approche de structuration inspirée du ***Multiple Instance Learning*** dans laquelle les données sont organisées sous la forme de ***bags***. Cette thèse explore plusieurs méthodes de classification telles que le Bayésien Naïf, les arbres de décisions, le SVM, et enfin un **réseau de neurones artificiel**, ce dernier nous ayant permis d'obtenir les prédictions les plus fiables. Ce modèle, qui tient compte du **déséquilibre des classes** dont les données sont entachées, permet de prédire la survenue d'une erreur de haute sévérité, avec une très bonne fiabilité (0.8 en termes de F1-score).

Enfin, nous avons exploité les dépendances temporelles auxquelles les erreurs sont associées à travers l'application de méthodes d'apprentissage basées sur un **réseau de neurones récurrents**.

Mots-clés : Maintenance Prédictive, classification, déséquilibre de classes, apprentissage profond, données *log*

Laboratoire Bordelais de Recherche en Informatique (LaBRI)
Unité Mixte de Recherche CNRS (UMR 5800)
351 cours de la Libération, 33400 Talence, France

Abstract

Title: Critical error prediction in an industrial application

Predictive Maintenance aims to anticipate breakdowns on industrial equipment using data analysis and **machine learning** methods. Therefore, industrial machines are designed to provide a daily set of data in various formats. The goal of this PhD thesis research is to exploit **log events** data to predict the occurrence of several **critical errors** on machine tools.

In this thesis, we propose a supervised **classification** based approach dedicated to critical errors. In this perspective, a history of critical and non-critical errors has been collected during one year on a fleet of textile industry machines. We explored different classification methods such as naive bayes, Decision Trees, SVM, and finally an artificial neural network, the latter allowed us to obtain the most reliable predictions. The model tackles the problem of imbalance classes in the data and produced high quality results for a specific critical error with a **F1-score of 0.8**.

To the purpose of labeling ambiguous data, we have developed a data structuration approach inspired by the **Multiple Instance Learning** paradigm. The data are organized in a structure referred to as **bags**.

Finally, to capture and analyze the temporal error dependencies, we explored learning methods based on recurrent neural networks.

Keywords: Predictive Maintenance, classification, class imbalance, deep learning, log data

---

Laboratoire Bordelais de Recherche en Informatique (LaBRI)
Unité Mixte de Recherche CNRS (UMR 5800)
351 cours de la Libération, 33400 Talence, France

Remerciements

Tout d'abord, je tiens à remercier d'une part, Ladjel Bellatreche et Bernard Kamsu-Foguem, pour avoir accepté de rapporter mon manuscrit, et d'autre part Karell Bertet, Omar Boussaid ainsi que Richard Chbeir pour avoir accepté de participer à mon jury.

Si ce projet de recherche a pu aboutir, c'est grâce à la bienveillance et l'expérience inestimables de Marie Beurton-Aimar, Sofian Maabout et Gayo Diallo qui ont su diriger cette thèse avec toute la patience et la rigueur scientifique qui était nécessaire. Je vous remercie vivement pour m'avoir encouragée et soutenue durant ces quatre années.

Un grand merci également à tous les membres du LaBRI avec lesquels j'ai eu l'opportunité d'interagir sur des sujets scientifiques et moins scientifiques, au détour d'un GT ... ou d'un thé. Pascal Desbarats, Akka Zemmari, Xavier Blanc pour les nombreux vrais cafés proposés à ton bureau, Bruno Pinaud, Patricia Thébault, David Auber, Romain Bourqui et Romain Giot pour avoir suivi le périple de cette thèse et m'avoir apporté de l'aide, des conseils avisés, du soutien et des encouragements tout au long de celle-ci. Hervé Hocquard, pour m'avoir donné l'opportunité de goûter à l'enseignement. J'ai ainsi pu enrichir mon parcours académique en passant de l'autre côté de la barrière. Adrian Tanassa et Vincent Penelle pour nos bavardages dans la cuisine du 3ème étage, sous couvert d'une bouilloire qui chauffe. Puis enfin tous les membres de l'équipe BKB pour votre écoute et vos conseils lors des différents exposés.

J'adresse maintenant mes remerciements à l'ensemble du bureau 328 : Cécil, Nadia, Kévin, Tù, Luc et puis surtout Gala, Claire et Elsa. En quatre ans, il y en a eu du *turn-over*, et pourtant j'ai pu tisser avec chacun de vous une amitié profonde. Gala, avec qui j'ai vécu le confinement à distance, tu as même eu l'audace de vivre quelques expériences de montagne en ma compagnie ! Merci énormément les filles pour tout le soutien que vous avez pu m'apporter, dans ce bureau et en dehors, malgré toute l'espièglerie dont j'ai fait preuve.

Je remercie les futurs docteurs rencontrés récemment au labo, Maxime (ou Xima pour les intimes) et Chabname pour les soirées Tarot et surtout Yannis pour nos explorations des *hot spot* buffet.

Après les plus jeunes, un mot de remerciement pour les anciens, la plupart étant déjà docteur à l'heure où j'écris ces lignes, je les ai vus essuyer les plâtres avant moi. Gaëlle, merci infiniment pour ta relecture minutieuse de ce manuscrit et tes nombreux conseils, merci également pour ta curiosité sans fin qui pousse mon esprit à se questionner davantage. J'espère que les quelques semi-marathons/triathlon/GR ensembles n'étaient que le début d'une longue série. Gaëtan,

mon meilleur binôme d'escalade de tous les temps, a pris la fuite trop tôt en *Parisie*, mais il en reste heureusement une belle amitié. Aaron, rencontré à l'ISPED mais retrouvé avec joie sur les débuts de ma thèse au LaBRI pour les café-philo. Jason, avec qui j'ai régulièrement discuté au détour d'un couloir ou d'une soirée jeu de société improvisée. Tous mes copains de l'ISPED, Hadrien qui m'a fait découvrir les joies de l'escalade en nature. Bruno toujours prêt à papoter à toute heure, quel que soit le sujet de discussion, Anna pour ton écoute et tes efforts à organiser régulièrement des retrouvailles entre copains.

Enfin mes copains de soirée, Rohan, Julien, Rémi, Christelle, Tina, Imane, et surtout Simon, qui est toujours prêt à me proposer des activités (très souvent des bars) pour sortir et me changer les idées. Grâce à toi, mon rapport aux manèges à sensation aura changé à tout jamais !

Mes quatre années de thèse, c'était aussi pour moi beaucoup de sport, alors merci à tous ceux qui m'ont permis de réaliser mes défis sportifs et de me dépenser régulièrement. Je pense particulièrement à François-Luc pour toutes mes ascensions hivernales en raquettes, mais également Bruno, Hervé, Flore-Anne pour ton esprit de coach, Alain toi qui t'es révélé être un binôme d'escalade assidu et patient, et grâce à qui j'ai pu me vider l'esprit à *Climb'up*.

Au cours de cette expérience, j'ai pu approfondir en parallèle ma passion pour le cyclisme. Pour cela, merci à mes copains cyclistes qui m'ont accompagnée sur des sorties de parfois plus de 100 km, (mais la plupart du temps des "*tranquilles*" entre midi et deux). Merci Xavier Blanc et Bruno Pinaud pour les départs depuis le LaBRI, François-Luc, Hervé, Jean-Luc, Cédric, Thomas, Gurvan et Marie-Claire pour les départs depuis Cestas. Puis Étienne pour nos excursions périgourdines ou girondines toutes en finesse !

Enfin, merci aux membres du club MRS pour vos conseils lors des entraînements de rollers qui m'ont permis de progresser dans cette discipline et d'améliorer ma technique.

Si j'ai pu tenir le rythme d'un bout à l'autre de cette aventure, c'est aussi grâce à des amitiés très précieuses que j'ai eu la chance de conserver depuis plus de 10 ans. En premier lieu, Manon, "ma femme" de toujours, un soutien inestimable sur lequel j'ai toujours pu compter, à tout moment. Tu as toujours su être à l'écoute et surtout trouver les mots pour m'éclairer. Merci également d'avoir relu mon manuscrit ! Ce n'était pourtant pas une sinécure. Marie-Claire, nos retrouvailles sont à chaque fois un plaisir accompagné d'un bol d'air frais ! Kévin, une amitié de licence biologie, toujours partant pour des activités en ma compagnie.

Enfin, j'aimerais adresser ces dernières lignes de remerciement à l'ensemble de ma famille, dont les liens resteront quelles que soient les épreuves. Maman et Papa, vous avez toujours été là pour m'encourager dans tous mes choix et tous mes projets, le plus important pour vous étant de vous assurer que chacun de vos quatre enfants soit heureux dans sa vie. Prisca, merci de t'être rendue disponible de nombreuses fois pour passer du temps avec moi au cours d'un thé et d'un *gloubi-boulga* de mon cru. Christel pour avoir prêté une oreille attentive à mes péripéties et mes doutes, Bobby Julien pour ton enthousiasme à expérimenter de nouvelles activités avec moi et enfin Abel, pour le partage de ton énergie et de ton insouciance, issues de ta jeune âme d'enfant.

Table des matières

Introduction

Le concept d'industrie se développe au XVIIIe siècle avec la première révolution industrielle et la nécessité de produire en masse, puis le besoin évolue, avec l'arrivée de l'automobile, des produits plus complexes avec des phases d'assemblage plus nombreuses sont conçus. Il ne faut pas seulement produire vite, il faut également produire proprement. Très vite, il a fallu proposer et diversifier des moyens automatiques plus évolués pour fournir, à faible échéance et à moindre coût, les matières premières et les produits manufacturés.
Entraînée dans le mouvement de la mondialisation, l'automatisation industrielle s'est développée de plus en plus, touchant tous les domaines de manufacture. Les produits s'exportent plus facilement à travers le monde, favorisant la collaboration industrielle : plusieurs acteurs situés sur différents continents contribuent à la chaîne de fabrication des produits.

L'année 2010, voit naître la quatrième révolution industrielle de l'histoire, qui s'appuie sur l'avènement de l'Internet des Objets ou *Internet of Things* (IoT). Cette révolution industrielle, qui prend berceau en Allemagne, est appelée Industrie 4.0.
Dans le cadre de l'**Industrie 4.0**, les objectifs de production évoluent pour proposer plus de diversité aux clients et il devient alors nécessaire de réduire les coûts et minimiser les délais de fabrication.
Les usines se mettent alors à gérer les stocks en flux tendu, ce qui leur permet de restreindre les stocks de matières premières uniquement aux besoins de la production immédiate.
La production et l'acheminement à la fois des matières premières et des produits finis se coordonnent dans un équilibre parfait où la moindre panne d'un équipement peut entraîner l'interruption de la ligne de production, et par conséquent une diminution la productivité de l'usine. Cependant, le vieillissement des équipements conduit souvent à des pannes, et donc leur arrêt.

La gestion de ces pannes (ou défaillances), passe par trois types de politique de maintenance. Premièrement, lorsqu'elles sont observées, les défaillances sont traitées par une opération de **maintenance corrective** qui consiste à remplacer ou réparer l'élément dégradé identifié comme responsable de la défaillance. Cette opération est effectuée *a posteriori*, lorsque la panne est constatée. Cette politique nécessite une étape de diagnostic du système et d'identification de la démarche corrective à appliquer. Cela a pour conséquence de décupler le temps d'inertie de la machine et d'altérer la production se traduisant par une perte de rendement. Afin d'anticiper ces pertes, un deuxième type de politique de maintenance a été introduit et adopté dans l'industrie : la **maintenance préventive**. Plutôt que d'attendre l'observation d'une panne, le principe est d'effectuer par prévention un entretien de la machine à intervalles de temps réguliers (maintenance systématique) ou par dépassement de seuil d'usage (maintenance conditionnelle).
La maintenance systématique se montre efficace en termes de prévention, spécifiquement dans les cas où les pannes sont dues à des pièces dégradées par le temps. Pourtant, du fait de la fréquence des entretiens, cette pratique peut se montrer onéreuse. Si la plupart des maintenances planifiées sont en réalité inutiles, le coût sera amplifié par le nombre de pièces remplacées à tort, et la main d'œuvre employée pour ce faire. Parallèlement, le temps d'arrêt de la production est également impacté par les interventions régulières nécessitant une interruption du système.
La maintenance conditionnelle qui est une stratégie plus fine permet de réduire le nombre de maintenances inutiles grâce à la surveillance de l'usure des pièces, mais dans le cas de systèmes aux conditions variables de fonctionnement, le seuil d'usure seul devient vite inadapté. Face à ces limitations, et afin de traiter le problème par anticipation, la politique de **maintenance prédictive** (MP) a été introduite dans l'industrie. Cette politique consiste à analyser les données d'usage du système dans le but de prédire la survenue de l'événement de défaillance. Ainsi, les pannes qui surviennent malgré la politique de maintenance préventive mise en place, peuvent être évitées et les opérations de maintenance corrective peuvent être réduites.

L'objectif de cette thèse est de proposer une solution de maintenance prédictive, pour une flotte de machines de découpe utilisées dans l'industrie du textile. Les travaux de cette thèse sont à l'interface entre les approches de type statistiques et les approches purement informatiques telles que l'apprentissage profond. Les machines, réparties dans le monde entier, sont utilisées pour des besoins de découpe variés, adressés à différents marchés, tels que l'ameublement, l'automobile, l'avionique et surtout l'habillement.
Avant les travaux de cette thèse, seules les politiques de maintenance corrective et de

maintenance préventives étaient mises en place pour gérer les pannes de ces machines de découpe.
L'ensemble de ces machines produisent au quotidien des fichiers *logs*, qui retracent les données d'usage, de maintenance et d'erreurs, qui ont été émises sur les dernières 24 heures. Ces données de *log* contiennent donc des informations sur l'état et le fonctionnement des machines qui sont essentielles pour la prédiction des pannes

Afin de répondre à l'objectif, nous nous sommes intéressés à prédire la survenue des erreurs de haute sévérité. Ainsi, nous avons développé une méthode de structuration et d'analyse de ces **fichiers *log*** basée sur le paradigme du ***Multiple Instance Learning*** (MIL). Cette structuration des données, sous la forme de ***bags*** a permis l'application d'un modèle d'apprentissage supervisé pour la prédiction des erreurs critiques.

Dans le chapitre 1, nous proposons une revue de la littérature concernant différentes approches de maintenance prédictive appliquées dans l'industrie.
Le second chapitre, présente et détaille notre méthodologie de structuration des données appliquées aux fichiers *logs* pour prédire la survenue d'erreurs critiques. Le chapitre 3 est consacré à l'application de notre approche de prédiction d'erreurs. Plusieurs méthodes d'apprentissage supervisé ont été comparées, telles que le Bayésien Naïf, le SVM, l'arbre de décision, la forêt aléatoire, puis finalement un réseau de neurones artificiels. Le chapitre 4 présente une nouvelle approche de prédiction par apprentissage profond, que nous avons explorée, consistant en un **réseau de neurones récurrents**. Enfin, ce manuscrit se termine sur quelques perspectives de recherche que nous proposons pour traiter davantage les différentes problématiques rencontrées dans ces travaux de thèse.

Chapitre 1

État de l'art

Introduction

Le maintien des équipements industriels dans un état opérationnel passe par leur entretien régulier au moyen d'opérations de maintenance. De nos jours, il existe trois types de politiques de maintenance qui peuvent être mises en place, et combinées afin de s'assurer que les machines restent opérationnelles. Ces politiques sont la **maintenance corrective**, la **maintenance préventive** et la **maintenance prédictive** (MP). Cette dernière a pour objectif d'anticiper les éventuelles défaillances au moyen d'un modèle de prédiction, formalisé à l'aide d'un algorithme qui analyse l'ensemble des données de fonctionnement issues des machines.

Certains efforts ont déjà été réalisés pour la prédiction de pannes dans un contexte de maintenance prédictive. Ainsi, dans ce chapitre, nous présentons une revue de la littérature d'une partie des approches existantes qui pourraient être utilisées dans ce contexte. Il est à noter qu'une grande partie des méthodes décrites dans ce chapitre sont des méthodes statistiques d'analyse de données. Même si elles sont couramment utilisées dans des contextes et des problématiques différents, nous nous sommes concentrés sur leur application dans le cadre de la MP.

Ce premier chapitre est découpé en six parties. La première partie nous permet d'introduire les problèmes et les besoins liés à la MP et contextualiser son usage dans le cadre de l'industrie. Les quatre sections qui suivent décrivent des familles d'approches pouvant être utilisées en MP. Enfin, dans la dernière section, nous reviendrons sur plusieurs méthodes de préparation des données permettant d'améliorer la qualité de prédiction.

1.1 Positionnement du problème

L'utilisation des machines en milieu industriel nécessite le maintien de leur fiabilité [Lap92, ALRL04].

1.1.1 Fiabilité des équipements industriels

La fiabilité est un concept étroitement lié à trois types d'événements : les défaillances, les défauts et les erreurs.

La **défaillance** est définie comme un comportement de l'équipement à un instant t, qui ne respecte pas les spécifications fonctionnelles et nécessite une réparation.

Une **erreur** est un événement immédiat qui révèle une divergence de l'un des comportements internes du système.

Enfin, un **défaut** est une irrégularité qui peut causer une défaillance interne de l'équipement. Parfois un défaut entraîne l'émission d'une erreur.

Par exemple, prenons un cas d'application consistant en un système de convoyeurs mécaniques, utilisés pour transporter et trier des charges. Les convoyeurs sont dotés de roulements mécaniques. Supposons que l'un des roulements contienne une bille abîmée : il s'agit d'un défaut. Le roulement impacté perd de son efficacité et surchauffe. À ce stade, le défaut s'amplifie puis avec le temps le roulement finit par rompre. Si le système est programmé pour surveiller l'état des roulements, une erreur est émise pour signaler la rupture du roulement, mais le convoyeur fonctionne toujours. Le manque de roulement à cet endroit entraîne une augmentation de la charge sur les autres roulements qui doivent alors compenser et finissent par rompre eux aussi. D'autres erreurs sont émises, pour signaler la rupture des autres roulements, puis avec le manque de roulements viables nécessaires à son bon fonctionnement, le convoyeur surchauffe et cesse de fonctionner : il s'agit alors d'une défaillance du convoyeur mécanique. Finalement, une défaillance peut être expliquée par une erreur ou une combinaison d'erreurs qui elles-mêmes révèlent la présence d'un défaut.

La gestion des défaillances sur les machines est étroitement liée à la notion de **fiabilité**. La fiabilité d'un équipement est définie comme la confiance que l'on peut accorder au service qu'il fournit. Elle dépendra essentiellement de la disponibilité, la sécurité, la robustesse ainsi que de la maintenabilité de l'équipement. La **robustesse** d'un système correspond à sa capacité à rester le plus longtemps possible dans un état de fonctionnement qui respecte les spécifications. La **maintenabilité** est la capacité du système à revenir le plus rapidement possible à cet état de fonctionnement normal afin d'optimiser sa durée de vie. Enfin, c'est la combinaison de la maintenabilité et de la robustesse qui contribue à garder l'équipement opérationnel plus longtemps. Pour répondre à cet objectif de fiabilité, de nombreuses méthodes de diagnostic ont été développées dans l'industrie. La section suivante présente une partie d'entre elles.

1.1.2 Diagnostic des défaillances

Dans le cadre du maintien de la fiabilité des équipements, le **diagnostic** permet d'ajuster la tâche de maintenance corrective nécessaire et de réadapter la politique de maintenance préventive. En effet, avec la connaissance des défauts qui mènent à la défaillance et en combinaison d'une approche de détection de ces défauts, la régularité des entretiens des pièces pourra être réajustée et les opérations de maintenance corrective réduites. Le diagnostic a trait majoritairement à la détection de défaut(s) (appelée *Fault Diagnosis* dans la littérature scientifique associée à ce sujet) mais peut également

s'appliquer pour comprendre le processus à l'origine des défaillances. Cette approche permet d'acquérir une compréhension suffisamment solide de l'événement de défaillance pour déterminer l'opération préventive à appliquer dans le futur. Le diagnostic nécessite, d'une part, un travail de **détection** qui consiste à vérifier si le comportement réel du système est cohérent par rapport aux spécifications. Plus précisément, la détection consiste à décider qu'un indicateur de défaut témoigne d'un réel symptôme. D'autre part, le diagnostic nécessite un travail d'**isolation** et/ou d'**identification** [Ise11] des défaillances, qui consiste à établir les relations causales entre les symptômes observés et la défaillance. Ainsi, les travaux d'identification et de localisation de défaillance font l'objet de nombreuses contributions scientifiques dans le champ du diagnostic [ZL13, LS95, TYTS01, SOvdAtH13, VJA+19] durant ces dernières décennies.
La plupart de ces travaux met en place un modèle représentatif du système physique réel, afin de simuler les différentes étapes du processus menant à la défaillance. Une telle méthode de simulation peut par exemple être mise en place au moyen d'un réseau de Petri (`RdP`) [CGPS11, MWS13, Bas14, CGHS15].

La simulation au moyen d'un `RdP` repose sur les probabilités de transition d'un état à l'autre. Dans un système à événements discrets [Laf19], le nombre d'états du système est dénombrable (par exemple les états "éteint" et "allumé" d'un ensemble de lampes dans le système). Le passage d'un état à un autre est déclenché par un événement binaire (par exemple l'actionnement de la commande d'allumage). Dans un système continu, ce passage d'un état à l'autre se fera de manière continue, c'est-à-dire que la transition se déroule par une valeur qui s'éloigne d'un état pour se rapprocher vers un autre. En MP, il est courant de traiter des problématiques associées à des systèmes continus. Par exemple lorsque les systèmes industriels nécessitent une montée en température et en pression pour la fabrication d'aluminium. Aussi, plusieurs travaux sur les `RdP` ont été appliqués dans le cadre de systèmes continus [PZHK05, DA10]. En parallèle, les systèmes ayant des comportements physiques continus peuvent être assimilés à des systèmes à événements discrets en appliquant des seuils de valeur. Par exemple, Sun *et al.* [SQS04] ont modélisé un réseau électrique par un `RdP` discret. Leur approche consiste à représenter les différents éléments physiques du réseau électrique tels que les lignes, les relais électriques ou les transistors dans un modèle discret. La propagation normale du signal électrique au sein du système ainsi représenté est modélisée par le passage des transitions dans le `RdP`, valable en fonction de l'état de l'élément physique au moment de la transition.

Par ailleurs, les systèmes continus peuvent également être modélisés à l'aide d'approches basées sur l'étude des résidus [IB97]. Dans un contexte de diagnostic, les résidus correspondent à des indicateurs de défaut, déterminés mathématiquement à partir

de l'écart entre une mesure observée et une mesure attendue. Par exemple, Roth *et al.* [RLL09] ont mis en place la mesure des résidus au travers d'un automate qui modélise la séquence d'un système de convoyeurs mécaniques. De telles méthodes ont également été adoptées dans des objectifs similaires à la détection de défauts, par exemple, la protection contre les intrusions [AMZ09], ou la détection d'anomalie [GTDVMFV09] dans les réseaux informatiques.

Les approches de diagnostic permettent d'identifier et de localiser le, ou les défauts à l'origine de la défaillance et de comprendre l'enchaînement des événements qui mènent à cette défaillance. Ces méthodes peuvent être appliquées afin d'avoir une connaissance experte du système étudié, à condition que ce dernier ne fasse pas l'objet d'une trop grande instabilité comportementale, sans quoi le diagnostic doit être régulièrement mis à jour.

Le diagnostic d'un équipement industriel reste une stratégie adressée *a posteriori*, vis-à-vis de la défaillance étudiée et ne permet pas d'anticiper directement les défaillances à venir. Par opposition, la maintenance prédictive (MP), introduite dans la prochaine partie, s'attache à prédire les défaillances futures afin d'organiser la maintenance en avance.

1.1.3 Maintenance prédictive

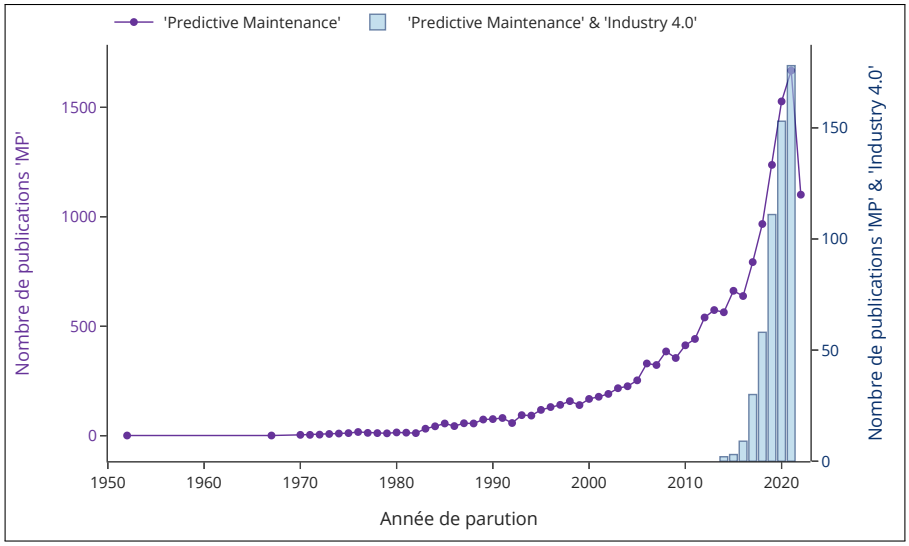

FIGURE 1.1 – Distribution des publications associées aux mots-clé "Industry 4.0" et "Predictive Maintenance" sur les 50 dernières années.

Avec l'avènement de l'industrie 4.0, et l'utilisation de capteurs de plus en plus précis sur les équipements industriels, la recherche scientifique en MP a connu un large essor ces dernières années. Pour illustrer ces propos, la figure 1.1 représente les statistiques de publications en lien avec l'industrie 4.0 et la MP depuis les années 1950 (Source : Scopus). La figure 1.1 représente l'évolution du nombre de publications associée à l'expression "*Predictive Maintenance*" (courbe violette) ainsi que le nombre de publications associées à la combinaison des mots-clé "*Predictive Maintenance*" et "*Industry 4.0*" (histogramme bleu). Ce graphique montre un essor du nombre de publications jusqu'en 2021. La MP a suscité dès les années 50 un vif intérêt dans la recherche, qui a brusquement augmenté dès 2018, année au cours de laquelle les contributions liées à l'industrie 4.0 se sont multipliées. Les approches de MP sont largement étudiées, et ce quels que soient les domaines comme en atteste la figure 1.2.

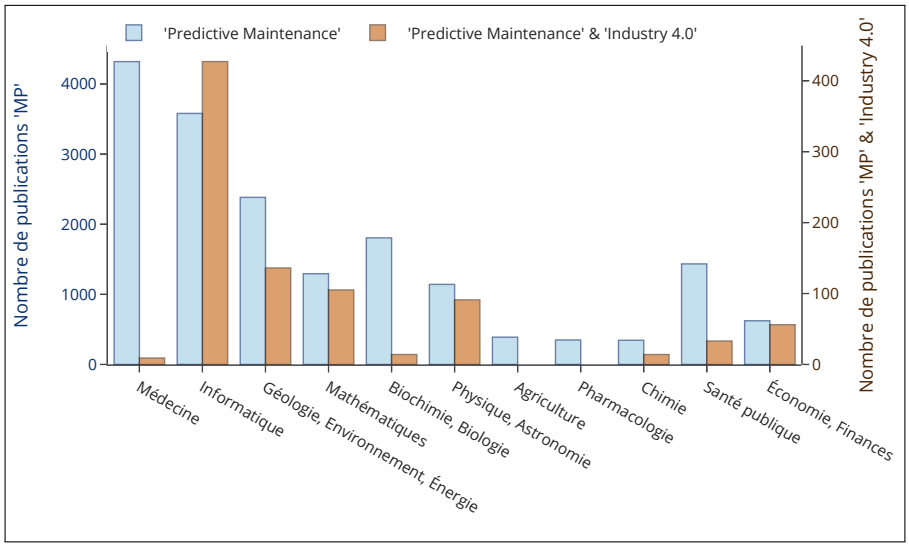

FIGURE 1.2 – Répartition des publications associées aux mots-clé "*Predictive Maintenance*" par domaine.

Nous allons maintenant nous intéresser aux approches permettant de prédire la survenue d'une défaillance par une stratégie d'analyse des données du système étudié. Pour commencer, la section suivante présente une famille de méthodes permettant d'extraire des combinaisons de données, appelés motifs. Ces motifs, corrélées à la survenue de la défaillance, peuvent constituer un signal fort que l'on peut chercher à détecter pour prédire la défaillance.

1.2 Extraction de motifs

Les observations recueillies à partir des systèmes (mécaniques et/ou logiciels) peuvent former des combinaisons d'événements, apparaissant plus ou moins fréquemment et pouvant expliquer la survenue d'une panne. Dans le cadre de la MP, certaines approches s'appuient sur l'extraction de ces motifs, dont la relation avec l'événement de défaillance peut être déterminée par la suite à l'aide de règles d'association [KFRM13]. Par exemple, on pourrait déceler un motif composé des événements "connexion", "démarrage de tâche", "déconnexion", et "arrêt système". Lorsque ce motif apparaît, une défaillance survient l'instant d'après car l'événement "fin de tâche" n'est pas présent dans cette combinaison d'événements. Dans cet exemple, l'ordre d'apparition des événements n'a pas d'importance, l'ensemble de ces événements suffit à indiquer l'arrivée imminente de la défaillance. Les motifs peuvent également être extraits à partir d'une séquence, c'est-à-dire un ensemble d'événements où l'ordre a de l'importance. Dans la littérature du domaine, on parlera plus particulièrement d'extraction de motifs séquentiels ou *sequence pattern mining*. L'un des travaux fondateurs du *sequence pattern mining* est celui proposé par Agrawal et Srikant [AS95], inspiré de l'algorithme `Apriori` [AS94] dédié à la recherche de sous-ensembles fréquents. Pour évaluer la fréquence des motifs, ces approches s'appuient sur la notion de support, définie par le nombre de séquences qui contiennent le motif considéré. Si le support dépasse une valeur seuil fixée par l'utilisateur, le motif est conservé.
D'autres déclinaisons ont été proposées pour la recherche de sous-séquences fréquentes. Par exemple, l'algorithme `SPADE` [Zak01] a permis de réduire le parcours d'une base de données, minimisant ainsi le coût généré par les tâches de lecture/écriture. Les algorithmes `SPAM` [AFGY02], `CM-SPADE` [FVGCT14], ou encore `Prefix-Span` [JQS+15] en sont des améliorations.

Ces algorithmes présentent cependant un inconvénient : ils peuvent dans certains cas extraire des motifs qui partagent un préfixe commun, dont la taille peut donc être réduite pour décharger la capacité de mémoire nécessaire. Par exemple, on pourrait extraire les motifs "ABCDE" et "ABCFGH" alors que les motifs "DE" et "FGH" suffisent. Pour pallier cette spécificité, la recherche de motifs clos a été introduite par Yan *et al.* [YHA03] avec l'algorithme `Clospan`. Un motif α de taille k est dit clos, s'il n'existe pas de motif β de taille $k+1$, contenant α tel que le support de α est le même que le support de β. Par exemple, $\alpha = BCDE$ est observé trois fois et $\beta = ABCDE$ est observé trois fois également. Dans ce cas, α n'est pas un motif clos.

La notion de motifs clos a été reprise ensuite pour la conception des algorithmes `BIDE` [WH04], `ClaSP` [GCMG13] ou encore `CloFAST` [FLCM16]. Le principe général de

ces algorithmes est de générer un ensemble de motifs séquentiels fréquents candidats puis de ne sélectionner que les motifs séquentiels clos parmi cet ensemble. Pour aller plus loin dans la présentation de ces différentes solutions d'extraction de motifs, une revue étendue de la littérature, proposée par Fournier-Viger [FVLK+17] présente une classification des différents algorithmes du domaine.

En termes d'application en MP du *pattern mining*, Lim *et al.* [LKK17] ont proposé une méthodologie en deux temps consistant d'abord à extraire les sous-séquences fréquentes à l'aide de l'algorithme `SPADE`. Dans un second temps, les sous-séquences les mieux corrélées à la défaillance sont sélectionnées par une méthode de régression (*Least Absolute Shrinkage and Selection Operator* `LASSO`). Par ailleurs, la recherche de sous-séquences fréquentes en MP a donné lieu à des travaux de recherche dédiés à l'analyse de séries temporelles [LC03], et à l'analyse de données de *logs* [MTB+21, CWDW20].

Une branche plus spécifique du *sequence pattern mining* s'intéresse au délai qui sépare les événements dans un motif, ce qui peut avoir de l'intérêt dans l'analyse de données temporelles pour affiner la prédiction. Cette forme de motif séquentiel, appelée une chronique [DD99], peut être employée pour prédire la survenue d'un événement, ainsi que le délai à l'issue duquel il survient. Les chroniques étant riches en information (chaque événement du motif est associé à une signature temporelle), les algorithmes proposés dans [DD99] ont souffert de problèmes de consommation de mémoire et de temps d'exécution. Une solution moins gourmande en capacité de calcul a été proposée par Cram *et al.* [CMM12] au travers de l'algorithme `HCDA`. Toujours dans le domaine de la recherche de chroniques, Sellami *et al.* [SMS+20] ont introduit l'algorithme `Clasp-CPM` capable d'ajouter une signature temporelle associée aux motifs fréquents obtenus par l'algorithme `ClaSP`. Cette approche consiste à isoler les séquences positives, c'est-à-dire les séquences se terminant par l'événement de défaillance. Les chroniques sont ensuite extraites et identifiées uniquement à partir de cet ensemble. Cependant, ce type d'analyse peut comporter un biais. En effet, les chroniques extraites à partir de l'ensemble des séquences positives peuvent également être présentes dans les séquences négatives. Il pourrait en résulter un taux de faux-positifs importants, diminuer par conséquent la précision de l'approche.
Aussi, Dauxais *et al.* [DGAGH19] ont introduit la notion de chroniques discriminantes, qui sont sélectionnées selon leur capacité à distinguer les séquences négatives des séquences positives.

L'extraction de motifs est donc une stratégie très courante dans la littérature associée à la prédiction de panne et de nombreux autres travaux sont disponibles à ce sujet [DB21, KKKK21, TK21, WLH+17]. En effet, ce type d'approche présente un avantage principal qui est d'apporter une connaissance plus précise des événements antérieurs à la panne et de mieux comprendre leurs inter-dépendances. En MP, ces approches nécessitent une étape supplémentaire qui consiste à étudier la corrélation des motifs extraits avec l'événement de défaillance.
La quantité de motifs identifiés par ces approches dépend entre autres de la valeur du support seuil fixée. Si le nombre de motifs est trop grand, cela peut conduire à des coûts importants en ressource mémoire, et en temps d'exécution. Par ailleurs, la valeur du support seuil a également une importance capitale pour la taille des motifs détectés puisque les algorithmes proposés génèrent habituellement des motifs de très faible taille (moins de 10 événements), ceci étant favorisé par le choix d'une valeur élevée pour le seuil du support. Il est possible de diminuer ce seuil afin de circonscrire l'extraction à des motifs de taille plus grande (plus de 10 événements), mais cela aux dépens de la ressource mémoire. La difficulté à isoler les motifs prédictifs parmi les nombreux motifs candidats ne facilite pas la prise en main de ces techniques, qui nécessitent beaucoup d'investissement en termes d'analyses *post* processus, afin de construire les règles d'associations permettant de relier ces motifs à l'événement de panne.

La section suivante présente des approches de prédiction très différentes de l'extraction de motifs, basées sur l'analyse et la modélisation des données émises par les systèmes mécaniques étudiés.

1.3 Apprentissage automatique

La notion de prédiction dans la littérature est souvent associée à l'utilisation d'un modèle d'apprentissage automatique. Le terme apprentissage automatique, ou *Machine Learning*(ML) en anglais, a été proposé pour la première fois en 1959, par Arthur Lee Samuel [Sam00]. Depuis, cette stratégie a été largement adoptée dans des domaines très variés tels que la santé publique [SR21], l'agriculture [LBM+18], ou encore la finance [LHT12]. Concernant la MP, une revue de la littérature parue en 2020 [CANZ+20] met en évidence une croissance du nombre de publications d'articles scientifiques associés au ML ces dernières années.

Les méthodes de ML permettent de déterminer les valeurs d'une variable réponse (Y) à partir d'un ensemble de **variables explicatives** (X), également appelées **attributs**, caractéristiques ou encore *features*. Chaque exemplaire x de l'ensemble X est appelé

instance et correspond à l'ensemble des valeurs associées aux attributs.
Dans le cadre de la MP, la variable réponse Y peut être continue : le but sera alors de prédire la valeur exacte de cette variable. Nous verrons également dans la suite de ce chapitre d'autres approches, appelées classification, où la variable Y correspond à une valeur booléenne indiquant la survenue (1) ou non (0) de la défaillance (ou erreur critique dans notre cas).
Enfin, le ML est divisé en deux catégories d'approches que sont l'apprentissage supervisé et l'apprentissage non supervisé, chacune d'elles détaillée ci-après.

1.3.1 Apprentissage non supervisé

Les approches non supervisées sont également connues sous le nom *clustering* [SPG+17, AG18] en anglais dans la littérature du ML, car le principe est de regrouper les instances par similarité ou par densité dans l'espace. Une étude des différents algorithmes de *clustering* dans le cadre de la MP a été proposée par Liao *et al.*. [WL05].

Dans un contexte industriel, certains types de données se présentent sous la forme de texte libre, comme les rapports de panne ou de maintenance. Afin de pouvoir extraire les nuances contextuelles de ces données en langage naturel, des méthodes de traitement automatique de la langue ou *Natural Langage Processing* (NLP) [Cho20] peuvent être appliquées pour analyser ces données. Parmi ces approches de NLP, nous pouvons notamment citer l'analyse de sentiments (*Sentiment Analysis*), une approche similaire au *clustering*, qui consiste à déterminer pour un document donné s'il est positif ou négatif, sur la base de son contenu sémantique. La modélisation de thème ou *Topic Modeling* (TM) est l'une des stratégies possibles pour classifier ces documents. Parmi ces méthodes, nous pouvons notamment citer l'allocation latente de Dirichlet (*Latent Dirichlet Allocation* - LDA), introduite par David Blei *et al.* [BNJL03], dont la méthodologie est décrite en détails dans l'annexe B.

Dans les premiers travaux de cette thèse, une méthode LDA a été explorée afin de classer des rapports de panne rédigés en langage naturel par les personnes opérant sur les machines. Le but était de regrouper les rapports afin de déterminer des types de pannes spécifiques. C'est une approche que nous avons voulue tester, malheureusement, nous avons rencontré des difficultés dans la mise en place de la méthode. Les machines étant hébergées dans des pays différents, les rapports étaient rédigés dans des langues différentes. Afin de circonscrire le problème, nous avons donc filtré les données en ne conservant que les rapports rédigés en anglais. Pourtant, même en réduisant le problème, nous nous sommes trouvés confrontés à des habitudes de langages différentes, dépendantes de l'individu. Par exemple, certains opérateurs décrivent le problème sous la forme d'une

demande d'intervention, sans connaissance *a priori* des symptômes. Dans ce cas, des mots approximatifs sont générés tels que "*please*" (s'il vous plaît), "*need*" (besoin), "*service*" (service), "*trouble*" (problème). Parallèlement, un opérateur ayant plus d'expérience avec la machine saura décrire le problème en détails et emploiera des mots plus techniques tels que "*sharpening*" (aiguisage), "*belt*" (bande) ou encore "*blade*" (lame). Dans ce contexte, les groupes générés par la méthode LDA ont par conséquent tendance à s'effectuer sur les opérateurs plutôt que sur le contenu de leurs descriptions.
Faute de résultat probant, cette piste a donc été abandonnée au bout de quelques mois de recherche.

En opposition à l'apprentissage non supervisé, l'apprentissage supervisé s'appuie sur un ensemble d'instances préalablement étiquetées. L'étiquette y de l'instance x correspond à l'une des valeurs possibles de la variable réponse Y. Dans les paragraphes suivants, les algorithmes d'apprentissage supervisé seront abordés en fonction de deux types d'approches : l'apprentissage par régression d'une part et l'apprentissage par classification d'autre part.

1.3.2 Apprentissage supervisé par régression

Les approches de régression ont pour objectif d'ajuster un modèle numérique à la distribution des données. Ce modèle, une fois entraîné, pourra prédire la valeur d'une variable Y à partir d'une instance x. En MP, les stratégies de régression sont particulièrement adaptées pour prédire la durée de vie utile restante du système (ou *Remaining Useful Lifetime* (RUL) dans la littérature). En effet, la survenue d'une défaillance peut parfois s'accompagner de l'évolution progressive de l'état du système vers une forme dégradée. Surveiller la RUL au moyen d'une modélisation des données temporelles permet de déterminer le moment où le système atteindra cet état dégradé [SB16]. Pour mettre en place cette approche, la stratégie consiste à créer un indicateur de dégradation permettant de synthétiser la série de données temporelles. Cet indicateur sera ensuite modélisé au moyen d'un algorithme de régression [WJT+17].

Les processus auto-régressifs sont une variante des modèles régressifs linéaires. La différence principale est que la variable à prédire est la même que la variable explicative. Cette approche, basée sur une série temporelle, fournit des prédictions qui s'appuient sur les observations passées de la série temporelle. Cela signifie concrètement que $Y = X$. Une telle approche a été appliquée par exemple dans Ho et Xi [HX98] pour modéliser une série temporelle d'événements de défaillance. Leur approche est appelée moyenne mobile auto-régressive, ou *Autoregressive Integrated Moving-Average* (ARIMA). L'hypothèse sur laquelle leurs travaux se sont appuyés est que les données peuvent être modélisées de

façon polynomiale. Un échantillonnage est effectué sur la série temporelle afin d'extraire le nombre d'événements de défaillance par intervalle glissant. L'échantillon obtenu est ensuite modélisé au moyen d'une régression linéaire polynomiale afin d'en extraire une moyenne mobile sur chaque intervalle. Les auteurs ont démontré l'intérêt de cette méthode, qui s'avère d'autant plus important si les valeurs de la série temporelle étudiée ne suivent pas une tendance monotone. Cependant, la méthode n'a été testée que pour un système stable et homogène, c'est-à-dire que le phénomène étudié est progressif et régulier, avec une évolution similaire sur les différents équipements. Lorsque les données sont acquises à partir de différents systèmes, la dimension temporelle, si indispensable à la bonne performance des modèles auto-régressifs, pourrait facilement être corrompue par des problèmes de synchronisation.

Les différentes méthodes de régression appliquées dans la littérature s'appuient généralement sur des séries temporelles continues et homogènes, adaptées pour décrire les changements progressifs d'état du système. Dans notre cas d'usage, les données *log* ne permettent pas de traduire un changement progressif et les données sont entachées de grands écart-types de valeurs. Ainsi, nous pensons que l'approche auto-régressive pourrait s'avérer limitée. Dans la partie suivante, nous allons nous concentrer sur les méthodes de classification qui peuvent être employées dans notre cas d'usage.

1.3.3 Apprentissage supervisé par classification

Dans cette partie, la plupart des algorithmes que nous allons décrire peuvent être aussi bien utilisés pour faire de la classification que pour faire de la régression. Notre objectif étant de prédire la survenue d'erreurs critiques au moyen d'une approche de classification, nous nous concentrerons uniquement sur les articles de la littérature qui appliquent cette approche.

L'apprentissage par classification consiste à séparer les exemples dans l'espace des valeurs. L'étiquette Y à prédire appartient à un ensemble fini de valeurs entières, chacune d'elles correspondant à une classe.
Dans cet état de l'art du ML, nous verrons aussi bien des approches de classification binaire (adressées à deux classes) que multi-classes (adressées à plusieurs classes). Rappelons qu'en MP, l'approche multi-classes peut être appliquée lorsqu'il existe plusieurs types de défaillance à prédire. Les approches à deux classes, quant à elles, consistent à mettre en opposition les instances liées à l'occurrence d'une défaillance, à celles liées à la non-occurrence de cette défaillance.

Pour commencer, le paragraphe suivant présente une approche supervisée, dont la stratégie est similaire aux méthodes de *clustering*.

1.3.3.1 K plus proches voisins (KNN)

Une approche de classification supervisée répandue dans la littérature s'appuie sur la répartition des instances dans l'espace. Il s'agit de l'algorithme des k plus proches voisins ou *K-Nearest neighbours* (KNN) [Pet09]. Pour chaque instance x non étiquetée, l'algorithme la classifiera en fonction de l'étiquette des instances étiquetées qui se trouvent dans son voisinage. La limite de ce voisinage est déterminé par un nombre k (impair de préférence) déterminé à l'avance. Ainsi, chaque instance est classifiée en fonction de l'étiquette majoritaire de ses k plus proches voisins étiquetés.
La figure 1.3 illustre cette approche dans le cas d'une classification à deux classes (classes bleue et rouge) où $k = 3$. Les instances non étiquetées sont représentées en gris et reliées à leurs 3 plus proches voisins par des flèches grises.

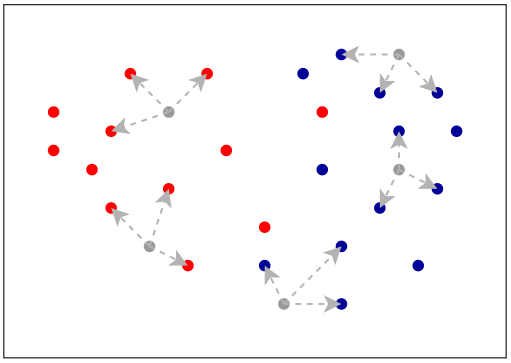

FIGURE 1.3 – Illustration de l'algorithme KNN dans le cas d'une classification à deux classes : classe 0 en bleu et classe 1 en rouge.

Un inconvénient de cette approche est que la détermination du paramètre k, souvent arbitraire, rend l'algorithme difficile à stabiliser. Aussi, certains auteurs ont recours à des méthodes d'optimisation des hyper-paramètres, telle que la méthode *Grid search* [CYLZ21] pour déterminer k. Par ailleurs, le KNN fonctionne très bien lorsque les classes sont bien segmentées dans l'espace. Dans notre cas d'usage, les données étant très hétérogènes, les instances pourraient être difficiles à regrouper ou distinguer simplement à partir des valeurs de leurs attributs, en utilisant l'algorithme KNN. Le paragraphe suivant présente une autre méthode de prédiction basée sur des critères de discrimination statistique.

1.3.3.2 Méthodes Bayésiennes

Les méthodes Bayésiennes, nées de la recherche en statistique, sont des approches probabilistes. Le principe est d'estimer la probabilité d'un événement. Par exemple,

on pourrait chercher à prédire la probabilité qu'une défaillance survienne juste après l'émission d'un ensemble d'événements. Plus particulièrement, la classification naïve Bayésienne repose sur la distribution des variables explicatives, le théorème de Bayes est détaillé dans l'annexe C.

L'algorithme Bayésien est dit naïf, car la probabilité de l'événement étudié est calculée en fonction de chaque variable prise indépendamment des autres. Cette hypothèse forte d'indépendance est mise en place pour faciliter la résolution d'un calcul rendu complexe par le nombre de variables. Dans la pratique, il existe souvent des corrélations entre les variables, ce qui viole cette hypothèse. Pour réduire ce biais, et considérer des groupes d'événements conjoints plutôt qu'un ensemble d'événements indépendants, certains auteurs ont combiné la méthode Bayésienne à une approche d'extraction de motifs [ASSZ17, JSS09], qui a déjà été développée au début de cette section.

1.3.3.3 Chaînes de Markov

À l'instar des approches de diagnostic que nous avons vues dans la première section, une possibilité pour classifier les données est de modéliser les séquences d'événements, au moyen, par exemple, d'une chaîne de Markov. Les chaînes de Markov [Rab89] sont un processus stochastique représenté par un graphe acyclique dirigé qui possède la propriété de Markov, selon laquelle l'information utile pour prédire un événement futur ne dépend que de l'état présent du processus. Chaque étape t, dans le processus stochastique, est représentée par un nœud du graphe et son état détermine la probabilité d'obtenir chacun des différents états futurs possibles à la position $t+1$ du graphe. Les chaînes de Markov cachées (`HMM`) [Rab89] sont une amélioration largement étudiée pour la prise en compte d'observations séquentielles de données [LCDF+15]. Les `HMM` sont construites sur l'hypothèse selon laquelle les observations sont générées par des variables non observées, qui peuvent néanmoins être modélisées par un raisonnement probabiliste.

En MP, les chaînes de Markov ont été par exemple appliquées dans [SM07], pour détecter les précurseurs suspects de l'occurrence d'une défaillance. Largement utilisée pour des problématiques de diagnostic de systèmes mécaniques simples (dont le comportement varie peu), ces approches, tout comme les `RdP` ou les méthodes Bayésiennes, pourraient s'avérer limitées dans les cas de systèmes complexes soumis à des conditions de fonctionnement variables, puisqu'il faudra alors régulièrement mettre à jour la conception du modèle.

D'autres techniques basées sur la distribution des valeurs consistent à modéliser la solution sous la forme d'un arbre. Cette technique sera présentée dans le paragraphe suivant.

1.3.3.4 Arbres de Décision et Forêt Aléatoire

Les Arbres de Décision (AD), dont le fonctionnement est détaillé dans l'annexe C sont basés sur la capacité des attributs à discriminer les instances de chaque classe.
En maintenance prédictive, cet algorithme a par exemple été appliqué par Kolokas *et al.* [KVIT18] pour prédire l'arrêt d'un équipement industriel dans un processus de production d'anode, dans l'industrie de l'aluminium.

L'avantage des AD, tout comme le KNN et la méthode Bayésienne, est qu'ils ne nécessitent que très peu de préparation des données.
En revanche, sur des jeux de données très hétérogènes comprenant un grand nombre d'attributs (plus de 100), ils s'avèrent souvent lents durant la phase d'apprentissage puisqu'il faudra beaucoup de critères de séparation pour discriminer les instances, augmentant de fait la profondeur de l'arbre. Par ailleurs, les AD sont également connus pour leur instabilité, particulièrement en contexte de données hétérogènes puisque chaque processus complet de décision est modélisé à partir d'un même ensemble d'entraînement.
Cette instabilité peut être cependant améliorée par des méthodes ensemblistes telles que les Forêts Aléatoires (FA) [LW+02, GFS+19, SWLY18] ou l'algorithme *Gradient Boosting* [LSW+17]. Ces algorithmes font intervenir un ensemble d'AD, dont l'agrégation des votes permet d'obtenir le modèle solution.

1.3.3.5 Séparateurs à Vaste Marge (SVM)

Les Séparateurs à Vaste Marge (SVM), ou *Support Vector Machine*, sont un ensemble de techniques d'apprentissage supervisé, populaires en classification [CGLRML20]. Le principe est de séparer les instances des différentes classes dans l'espace par un ou plusieurs hyper-plan(s) selon le nombre de classes, comme illustré sur la figure 1.4. L'intérêt du SVM tient dans la maximisation d'une marge de part et d'autre de l'hyper-plan de séparation de manière que la distance entre les instances de chaque classe et l'hyper-plan soit la plus importante possible. Le séparateur peut consister en une fonction linéaire, comme illustré sur la partie gauche de la figure 1.4. Si les données ne peuvent pas être séparées linéairement, des modèles à noyau sont alors adoptés (à droite sur la figure 1.4) pour constituer l'hyper-plan de séparation. Un exemple d'application en MP, est présenté dans Xian *et al.* [Xia10] ou [CTL16] dans une approche multi-classes.

Sur de petits jeux de données (moins de 1000 instances), la capacité du SVM à se concentrer sur les instances les mieux séparées en fait un modèle de prédiction précis. Cependant, ces méthodes souffrent de difficultés à prendre en charge les jeux de données

de grande taille (de l'ordre d'un million d'instances). Par ailleurs, lorsque les attributs sont nombreux (plus de 100), la solution à l'hyper-plan optimal est difficile à ajuster.

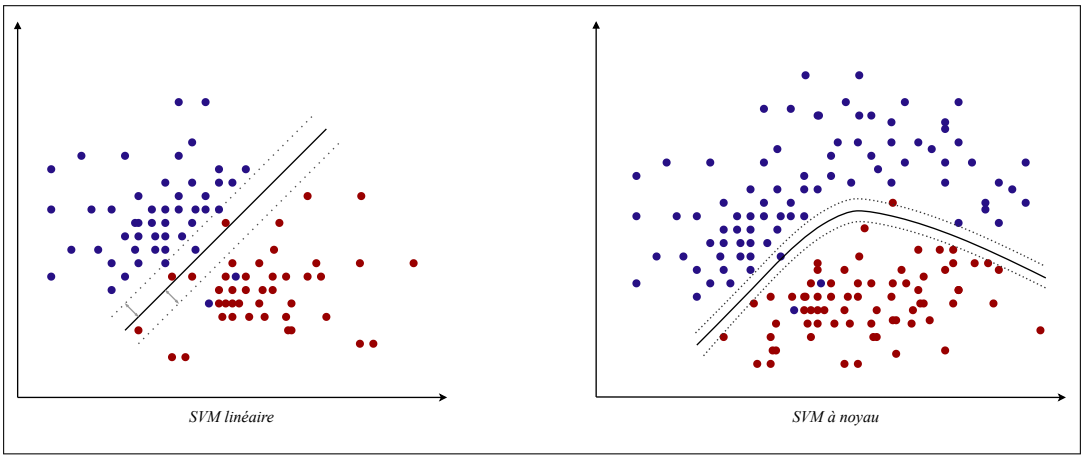

FIGURE 1.4 – Représentation de deux séparateurs `SVM` dans le cas d'une classification à deux classes : classe 0 en bleu et classe 1 en rouge.

Jusqu'à présent, nous avons vu un grand nombre d'approches fondamentalement issues de la recherche en statistiques et aucune ne semble idéale pour résoudre notre problématique. À présent, nous allons nous intéresser à une autre catégorie d'approche de prédiction provenant du domaine purement informatique : l'apprentissage profond, que l'on appelle *deep learning* dans la littérature du domaine.

1.4 Apprentissage profond

Les travaux en apprentissage profond ont débuté avec l'introduction des réseaux de neurones artificiels ou *Artificial Neural Network* (ANN) [JMM96].

1.4.1 Réseau de Neurones Artificiels

Depuis le début des années 2000, les réseaux de neurones artificiels (ANN) connaissent un regain d'intérêt croissant [AJO+18]. Contrairement aux `SVM`, ils sont capables d'apprendre à partir de jeux de données larges et complexes. Inspiré librement de la structure histologique du cerveau, le réseau de neurones est constitué d'unités fondamentales, les neurones, qui prennent en entrée une valeur pondérée et renvoient en sortie une valeur appelée activation. L'architecture des ANN comprend toujours une couche d'entrée, une couche intermédiaire et une couche de sortie, comme illustré sur la figure 1.5. Chacune de ces couches contient un nombre de neurones déterminé souvent de manière arbitraire, en tenant compte des contraintes inhérentes au jeu de données (le volume et le nombre de

features) mais aussi des capacités de calcul du matériel informatique. La dernière couche du réseau dépendra de l'objectif recherché : s'il s'agit d'une approche de classification, elle contient autant de neurones que de nombre de classes. Pour une régression, la dernière couche ne contiendra qu'un seul neurone.
Par extension, un réseau de neurones profond (ou *Deep Neural Network* (DNN) en anglais) correspond à des ANN comportant plus d'une couche intermédiaire.

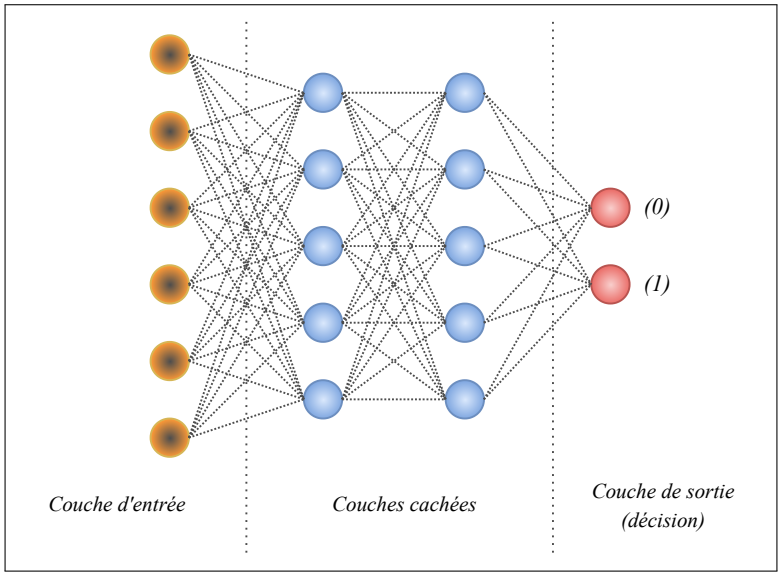

FIGURE 1.5 – Réseau de neurones profond à deux couches, utilisé en tant que modèle de classification à deux classes.

1.4.2 Réseau de Neurones Profond

Depuis quelques années, les DNN sont couramment utilisés en MP [WW18, RFC+20, ZXM+16]. Leur principale différence avec les ANN tient de leur architecture avec un nombre de couches de neurones cachées plus important (plus le nombre de couches est important, plus le réseau est profond), ce qui leur permet de réajuster plus finement les paramètres du modèle nécessaire à la prédiction les rendant plus adaptés pour des jeux de données plus complexes. Le principe fonctionne sur un mécanisme dit de rétro-propagation, expliqué en détails dans l'annexe A, qui permet de mettre à jour les poids attribués à chaque variable. Chaque étape de rétro-propagation des poids s'effectue en fonction de l'erreur de classement évaluée sur un échantillon du jeu d'entraînement. Afin d'approcher la solution non linéaire la plus précise possible, ce processus va être effectué n fois, de manière que le modèle d'apprentissage voit la totalité des instances de l'ensemble d'entraînement.

Pour simplifier la lecture, nous notons les appellations ci-dessous provenant du vocabulaire associé au *deep learning*. Ces appellations sont utilisées au cours des deux chapitres qui vont suivre :

Un *batch* correspond à un échantillon du jeu d'entraînement, de taille fixée en amont par l'utilisateur, souvent une puissance de 2 (2^n) pour des raisons de compatibilité de calcul sur le processeur physique de la machine (CPU). Le nombre de *batch* est défini en fonction du nombre de segmentations nécessaires pour que le réseau de neurones voie la totalité du jeu d'entraînement.

Une *epoch* correspond à une itération complète, durant laquelle l'ensemble du jeu d'entraînement aura été présenté 1 fois au réseau de neurones. Le nombre d'*epochs* est fixé en fonction du nombre d'itérations nécessaires pour que l'erreur de classement soit la plus faible possible.

L'attrait des DNN est également induit par leur capacité à extraire les attributs directement à partir des données sans nécessiter de méthode de construction et de sélection d'attributs (ces méthodes ont été abordées dans la dernière section de ce chapitre). En effet, la capacité à apprendre à partir de jeux de données presque sans pré-traitement des attributs devient indispensable dans l'ère de l'Industrie 4.0 puisque les capteurs sur les équipements sont de plus en plus nombreux, générant ainsi des variables de plus en plus nombreuses [LWL+19, NND+20].
D'un côté, les réseaux de neurones profonds sont reconnus pour leurs performances sur les données de masse, même lorsque les attributs sont nombreux. D'un autre côté, des phénomènes de sur-apprentissage peuvent émerger de ces approches qui nécessitent un travail d'ajustement du modèle. Ce phénomène, décrit en détails dans le chapitre 2, apparaît lorsque le modèle devient trop spécifique au jeu de données d'entraînement. Aussi, pour faire face à ces problèmes, des techniques d'élagage des neurones, appelées *dropout* [SHK+14], sont parfois adoptées pour réduire le sur-apprentissage.

Une autre catégorie bien connue d'apprentissage profond correspond aux réseaux de convolution [RDGC95a]. Ces réseaux, qui sont en général utilisés pour traiter le signal visuel, permettent de faire émerger de façon automatique les caractéristiques essentielles d'une image par compression des valeurs [ZLC+14, ZLC+16, CCC16, LYC+18]. Cependant, ces techniques très répandues en reconnaissance d'image, n'ont de sens que si l'ordre des colonnes dans le jeu de données a de l'importance, ce qui n'est pas le cas pour les données de fichiers *logs*.

Le paragraphe suivant présente une autre approche de *deep learning* encore différente, qui est régulièrement utilisée en MP pour des séries temporelles.

1.4.3 Réseau de Neurones Récurrents

Les réseaux de neurones récurrents ou *Recurrent Neural Network* (RNN) sont réputés pour leur capacité de mémorisation qui leur permet de prédire un événement sur la base des observations précédentes. Pour cette raison, ils sont particulièrement adaptés pour des séquences temporelles variables dont ils peuvent prédire le dernier événement. À la différence du neurone des ANN, le neurone d'un réseau récurrent prend en entrée à la fois la valeur d'activation de la couche précédente, et la nouvelle entrée correspondant à la valeur à l'instant t dans la séquence. De cette manière, l'information vue par le modèle à l'instant $t-1$ est retenue dans la valeur d'activation et contribue au calcul de l'activation qui dépend de t.
Les RNN natifs sont connus pour être soumis à un phénomène de disparition du gradient lors du mécanisme de rétro-propagation, particulièrement lorsque la séquence en entrée est très longue (plus de 100 événements). Aussi, une autre catégorie de neurone récurrent a été introduite pour pallier ce problème : les neurones à mémoire à court-terme et long-terme dits *Long short-term memory* (LSTM) [HS97].
La particularité du neurone LSTM est l'ajout d'un paramètre pour l'état de la cellule en fonction du temps c_t, qui joue un rôle de mémorisation. Cet état c_t est mis à jour à chaque étape en fonction de trois transitions ou portes. Ces portes sont des zones de calcul dans le neurone, qui permettent d'oublier ou d'intégrer de nouvelles informations pour l'état considéré. Des explications plus détaillées sur le fonctionnement du LSTM sont présentées dans l'annexe A. Une application des LSTM peut être retrouvée dans Rivas *et al.* [RFC+20] pour la prédiction de défaillance de moteurs. Les séquences utilisées pour alimenter le réseau correspondent à différentes mesures de capteurs, prises à intervalles de temps réguliers. Zhang *et al.* [ZXM+16] ont également utilisé l'approche LSTM en s'appuyant sur des données de fichiers *log* pour prédire la survenue des défaillances dans les serveurs informatiques.

Une variante des LSTM a été proposée plus récemment : les modèles basés sur des unités récurrentes à porte ou *Gated Recurrent Unit*(GRU) [CvMG+14]. Comparée au neurone LSTM, l'unité récurrente du GRU ne comporte qu'une seule porte, ce qui nécessite moins de paramètres à calculer. Par conséquent, ces modèles sont réputés pour être plus rapides d'exécution et moins gourmands en mémoire et en capacité de calcul que les LSTM. Wang *et al.* [WCN+21] ont utilisé un modèle basé en partie sur un réseau GRU pour apprendre les dépendances locales au sein des séquences de *logs* dans un objectif de détection d'anomalie.

Finalement, l'apprentissage profond, qui a déjà fait l'objet de nombreuses applications dans la littérature de la MP, pourrait s'avérer utile pour notre cas d'usage marqué par des jeux de données hétérogènes. Nous verrons dans le chapitre 3 une application du DNN et discuterons de l'intérêt de l'application d'un RNN de type GRU dans le chapitre 4. Dans la section suivante, nous reviendrons sur la notion d'étiquetage des données et découvrirons une approche de la littérature introduite pour étiqueter des données ambiguës.

1.5 Apprentissage multi-instances

Dans certains cas, les informations permettant d'étiqueter les données sont insuffisantes ou ambiguës et il n'est pas possible d'attribuer une classe précise à chaque instance. Par exemple, dans le cadre de la MP, il peut être difficile à un instant donné de déterminer si les observations sont représentatives d'un cas de défaillance ou non. L'apprentissage multi-instances ou *Multi-Instance Learning* (MIL) permet de répondre à ce problème.

L'approche consiste à mutualiser les étiquettes à un ensemble d'instances, regroupées dans une structure appelée le ***bag***, qui sera alors associé à une seule étiquette. Par exemple, sur des images, Ilse *et al.* [ITW18] ont appliqué le paradigme du MIL pour associer des groupes d'images histo-pathologiques à la présence ou non de cancer.
Appliqué à un contexte de MP, l'avantage est de pouvoir regrouper un ensemble d'observations proches les unes des autres dont la relation individuelle à la défaillance est difficile à déterminer. Au contraire, il sera plus facile de déterminer la relation de cet ensemble d'observations avec la défaillance [KKČ+18, MHKDS05, RBVdW14, GRW16, WLM19].

Le MIL est proposé pour la première fois en 1997 dans les travaux de Dietterich *et al.* [DLLP97] dont la problématique consistait à identifier les protéines ayant une affinité avec le site actif d'une molécule thérapeutique. Sachant qu'une protéine peut avoir différentes conformations spatiales, représentées par des instances, l'idée était de définir chaque protéine par un *bag* dont les instances sont les conformations. En supposant que l'étiquette puisse avoir deux valeurs possibles, par exemple 1 (positive) ou 0 (négative), la conception du MIL telle que l'a décrit Thomas Dietterich introduit l'hypothèse standard suivante sur l'étiquetage des *bags* :

— Un *bag* B prend l'étiquette 1 *si et seulement si* il contient une instance étiquetée 1.
— Un *bag* B prend l'étiquette 0 *si et seulement si* toutes les instances qu'il contient sont étiquetées 0.

À partir de cette hypothèse, les travaux de littérature distinguent deux modes de classification mis en évidence par Herrera *et al.* [HVB+16]. Le premier consiste en une classification ***instance-level*** où l'on cherchera à classer d'abord les instances du jeu de données afin d'attribuer une classe aux *bags* par la suite. Le second mode correspond à la classification ***bag-level*** où les *bags* sont d'abord classifiés, puis les instances sont étiquetées par héritage de l'étiquette du *bag* auquel elles sont associées (toutes les instances d'un *bag* négatif deviennent négatives et toutes les instances d'un *bag* positif deviennent positives).

Pour mettre en place une stratégie d'apprentissage *bag-level*, l'enjeu est de pouvoir trouver une fonction permettant de séparer les *bags*. Couramment, les méthodes sur la similarité, comme l'algorithme `KNN`, présenté plus tôt dans ce chapitre, permettent de répondre à cette question en considérant l'espace des *bags*.
Parallèlement, d'autres solutions consistent à transformer les *bags* par agrégation en une instance unique, appelée la **méta-instance**. Il suffit ensuite d'appliquer un algorithme de classification directement sur les méta-instances.

Dans l'objectif de proposer une solution de maintenance prédictive, Sipos *et al.* [SFMW14] ont exploré une méthode hybride basée à la fois sur la méthode *bag-level* et sur la méthode *instance-level*.
Leur méthodologie, appliquée sur des données de fichiers *log*, propose de regrouper les données *log* collectées chaque jour par intervalles temporels de sept jours. Cet intervalle temporel appelé **intervalle de prédiction** est assimilé au *bag*, dont l'étiquette est ensuite fixée selon sa proximité à l'événement de défaillance. Cette approche, dont laquelle nous nous sommes inspirés dans le cadre de ces travaux de thèse, sera discutée plus en détails dans le chapitre 2.

Nous avons passé en revue un sous-échantillon d'approches appliquées en MP. Pour toutes ces approches, la qualité des données joue un rôle indispensable et plus ou moins important dans la qualité de la prédiction et la robustesse du modèle prédictif. En effet, il est communément admis que des données de mauvaise qualité ne peuvent donner lieu à une information de bonne qualité [RF11]. Dans la section suivante, nous revenons sur ce point et discuterons des différentes méthodes de préparation qui permettent d'améliorer cette qualité.

1.6 Préparation des données

Comme nous venons de le dire, la construction d'un modèle de MP nécessite, pour produire un résultat de qualité, un jeu de données représentatif du système étudié et si possible dépourvu d'éléments parasites. Pour obtenir un tel jeu de données, des étapes de

préparation peuvent être mises en place en amont de la modélisation. Nous distinguons ces étapes en quatre catégories que sont : le nettoyage, la sélection des attributs, la correction du déséquilibre des classes, et enfin la mise à l'échelle des valeurs. Chacune de ces étapes est détaillée ci-après.

1.6.1 Nettoyage des données

Dans le cadre de la MP, les formats de données que l'on peut trouver sont divers et variés :

- Les **données de télémétrie** émises en temps-réel sont des valeurs continues issues par exemple des stations météo ou des sites internet de diffusion de flux audios ou vidéos.
- Les **données de capteurs** disposés sur des équipements mécaniques correspondent à des mesures physiques continues, par exemple l'amplitude de vibration, le régime d'un moteur, la température ou encore les ondes sonores émises par le système étudié.
- Les **données de capteurs logiques** sont des données discrètes qui transmettent une information binaire, par exemple l'activation ou le mouvement de certains composants détectés sur le passage d'une zone précise.
- Les données issues des **rapports de maintenance**, correspondant le plus souvent à du texte libre.
- Les données de **fichiers *logs*** issues des logiciels sont des données événementielles, permettant le suivi des processus d'exécution logiciel (connexion, déconnexion, démarrage du système, etc.). Ces données peuvent également correspondre à l'abstraction de certains signaux issus des capteurs auquel un seuil a été appliqué.
- Les **caractéristiques descriptives** des équipements, par exemple la version logicielle, l'identifiant unique d'une machine, sa localisation géographique ou encore son ancienneté. Ces caractéristiques sont des informations catégorielles statiques (indépendante du temps).

Le nettoyage des données diffère selon la source de données à laquelle on s'adresse. L'étape de nettoyage nécessite des tâches d'ingénierie des données qui permettent à la fois de supprimer les informations inutiles, qui pourraient créer de la confusion, et en même temps de construire des variables explicatives. Concernant les données à caractère continu, ces objectifs peuvent être atteints par l'application de méthodes de traitement du signal d'une part, et de fenêtre glissante d'autre part.

Pour commencer, les données de télémétrie, et les données de capteurs génèrent des séries temporelles de mesures très fines, continues et en général à haute fréquence. Le

signal issu de ces données est souvent trop complexe pour être interprété directement par les algorithmes de prédiction que nous avons présentés ci-dessus. Aussi, des techniques de **traitement du signal** [KSG+21] permettent de synthétiser ces mesures pour extraire le phénomène physique observé et les variations locales de celui-ci. Par exemple, la transformation de Fourrier est une technique classiquement utilisée pour traiter les signaux de vibration [SHIH02].

Par ailleurs, la fréquence d'échantillonnage est également une question qui se pose régulièrement dans l'analyse de séries temporelles soumises à des variations saisonnières [VAH+02]. En effet, la granularité temporelle des données peut soit s'avérer trop fine, ce qui force le modèle à considérer de l'information inutile ou au contraire trop grossière et ne permet pas au modèle de détecter une évolution.
Pour abstraire l'information inutile qui pourrait rajouter du bruit, des techniques de **fenêtre glissante** peuvent être adoptées [TTBS22] et ainsi ignorer les observations d'ordre temporel plus fin, qui n'ont pas d'intérêt pour la prédiction. La mise en place de ces fenêtres glissantes permet par ailleurs de simplifier le jeu de données, la phase d'apprentissage étant par conséquent rendue plus rapide.
Le signal brut issu de plusieurs types de capteurs différents (température, vibration et son) présente des fréquences d'échantillonnage différentes et doit être préparé pour que le modèle d'apprentissage puisse interpréter ces informations sur des échelles de temps identiques. Gutschi *et al.* [GFS+19] appliquent une méthode de fenêtre glissante à deux niveaux. Le premier niveau leur permet d'aligner les observations issues des différents capteurs sur des unités de temps identiques tandis que la seconde étape leur permet de synthétiser l'information via des mesures statistiques. La construction des attributs à partir de mesures statistiques est une stratégie régulièrement employée en analyse de séries temporelles [YB19] et en analyse de données de fichiers *log* [WLH+17]. Le but est d'appliquer une connaissance du domaine pour extraire des représentations analytiques à partir des données brutes afin de rendre les tendances plus faciles à détecter pour l'algorithme d'apprentissage automatique.

En ce qui concerne les données de fichiers *log*, les problématiques majoritairement soulevées sont une question d'hétérogénéité. Une information identique peut être exprimée par des messages *log* dans des formes et/ou des syntaxes différentes. Aussi, des stratégies d'extraction de structures communes à ces messages permettent de pallier ce problème [KWTI15] et ainsi aligner les données issues de ces messages grâce à un modèle commun.
Une particularité courante des messages *logs* dans les systèmes logiciels et mécaniques est leur émission en rafale [OKFE17, LTD13]. Le phénomène de rafale des messages

logs peut correspondre à l'émission en simultané d'un ensemble d'événements, ce qui traduit un problème non résolu qui empêche l'exécution de plusieurs éléments du système. Par ailleurs, ce phénomène peut également correspondre à un même événement dont la fréquence d'émission augmente de façon soudaine (de plus de 50 % de sa fréquence habituelle), ceci pouvant être expliqué par une augmentation (normale ou anormale) de l'activité du système.
Pour réduire la dimension des données induites par ce phénomène, certains auteurs choisissent de filtrer les messages redondants [SOR+03], en supprimant d'une part les doublons successifs dans le temps et d'autre part les redondances liées à des tentatives multiples de tâches. Zheng *et al.* [ZLPG09] utilisent des techniques statistiques de co-occurrence pour fusionner les messages exprimés pour la même indication sémantique.
Par opposition au phénomène de rafale, l'absence de message durant une certaine période peut signifier une perte de communication du système. Ce manque de données dans l'historique peut biaiser l'interprétation des données par le modèle prédictif qui pourrait assimiler ce phénomène à un état normal.

Enfin, rappelons que lorsqu'une défaillance survient, le retour du système à l'état opérationnel ne peut s'effectuer avant la fin d'une période de latence déterminée par le temps nécessaire pour pratiquer la maintenance. Aussi, quelle que soit leur source, les données émises durant cette période de latence ne peuvent présager ni d'un comportement normal, ni d'un comportement anormal (puisque l'état de dégradation est déjà atteint). Aussi, certains auteurs font le choix de mettre à l'écart cette partie des données infectées [KBV18].

Nous avons vu quelques méthodes de préparation des données qui sont parfois appliquées en vue d'améliorer la qualité des prédictions, quel que soit le type d'approche. Nous allons à présent nous concentrer sur la préparation des attributs, qui peut être nécessaire pour appliquer une approche de ML.

1.6.2 Sélection des attributs

Rappelons que les modèles de ML doivent être entraînés sur un jeu de données d'entraînement. Cet ensemble d'entraînement est constitué d'instances qui correspondent à des vecteurs de valeurs, ces dernières étant associées aux variables explicatives du modèle. Dans certains cas d'application, les variables explicatives peuvent se révéler trop nombreuses et limiter la qualité du modèle prédictif. Aussi, réduire le nombre d'attributs [GE03] en apprentissage automatique permet de répondre à plusieurs objectifs :

— supprimer les attributs redondants et ceux qui apportent de la confusion,

— décharger le modèle prédictif de calculs inutiles et ainsi rendre l'exécution plus rapide,
— limiter les phénomènes de sur-apprentissage [1] par des techniques dites de régularisation,
— apporter des éléments de compréhension aux résultats de prédiction obtenus pour améliorer l'interprétabilité[DVK17].

Dans la littérature, les méthodes de sélection d'attributs se départagent couramment en trois catégories : filtrage, emballage et intégration.

Les méthodes de filtrage permettent de sélectionner les attributs sur la base d'un classement. Leur rang dans le classement est obtenu par des scores statistiques via une analyse de variance telle que l'ANOVA, comme dans Gutschi *et al.* [GFS+19]). La contribution de ces attributs peut également être évaluée par des tests de corrélation comme le test du Chi2 ou le coefficient de corrélation de Pearson.

Les méthodes dites d'emballage sont basées sur le score de performance du modèle pour chaque sous-ensemble d'attributs. Le meilleur sous-ensemble de caractéristiques est choisi en fonction du modèle ayant produit les meilleures prédictions. L'élimination récursive des attributs (*Recursive Feature Elimination*-RFE) [YZ15] est l'une des méthodes possibles.

Les méthodes d'intégration pour finir combinent les deux méthodes précédentes. Par exemple, l'algorithme statistique du LASSO [Tib96] est un modèle linéaire basé sur le calcul des poids des attributs obtenus après l'entraînement de chaque modèle. Les attributs sont ensuite ordonnés en fonction de la somme de leurs poids sur chaque modèle linéaire entraîné. L'approche, déclinée par la suite en Bolasso [Bac08] a été utilisée par Sipos *et al.* [SFMW14] pour sélectionner les meilleurs attributs extraits à partir de données logs.

Enfin, lorsque aucun attribut n'est spécifiquement adapté pour aider le modèle à mieux prédire, une alternative consiste à former de nouveaux attributs à partir de ceux existants. Une telle approche peut être menée grâce à l'analyse en composantes principales (ACP).

1. Le phénomène de sur-apprentissage est décrit en détails dans le chapitre 2

1.6.3 Correction du déséquilibre de classes

Le phénomène des classes déséquilibrées est un problème courant dans la littérature dédiée à l'apprentissage supervisé par classification [BTR15]. Les classes sont dites déséquilibrées lorsque l'effectif de l'une des classes supplante celui des autres classes, ou d'une autre manière, lorsque l'effectif d'exemples étiquetés pour l'une des classes est trop insuffisant. Le modèle de classification, entraîné sur un jeu de données déséquilibré, pourrait s'avérer imprécis par sa propension à classifier plus souvent les exemples dans la classe majoritaire. Dans le milieu de maintenance prédictive industrielle, le phénomène est d'autant plus présent que les classes mettent en opposition l'absence de défaillance avec l'occurrence d'une défaillance, qui est normalement un événement rare. Pour pallier ce phénomène, deux types de méthodes de ré-échantillonnage des données peuvent être appliqués. Le premier s'intéresse à réduire l'effectif de la classe surabondante tandis que la seconde consiste à augmenter les données appartenant à la classe sous-représentée [LGMT16].

Le sous-échantillonnage consiste à ne sélectionner qu'un sous-ensemble des données pour l'apprentissage, soit de manière aléatoire [SFMW14, KBV18], soit de manière plus sophistiquée, par exemple avec une méthode de sélection ensembliste [NPF18].

Concernant les méthodes d'augmentation de données, certaines permettent de générer de nouveaux exemples par duplication, par exemple l'algorithme *Synthetic Minority Oversampling Technique* (SMOTE) [CBHK02]. D'autres méthodes, telles que l'algorithme *Adaptative Synthetic* (ADASYN) [HYGS08], consistent à synthétiser de nouveaux exemples, proches des exemples originaux dans l'espace.

Nous avons discuté des différentes méthodes qui peuvent être adoptées en amont de l'apprentissage d'un modèle de ML pour améliorer la qualité de la prédiction. Malgré ces méthodes, la qualité du modèle peut encore être limitée par de grandes différences de valeurs au sein des attributs extraits. Pour terminer notre état de l'art de l'analyse des données en MP, nous allons à présent nous intéresser aux différentes méthodes permettant de ramener les valeurs des attributs à une échelle commune.

1.6.4 Mise à l'échelle

Si les différentes caractéristiques étudiées ont des ordres de grandeur différents, il peut être nécessaire de les normaliser afin de les rendre comparables les unes aux autres pour l'apprentissage. Parmi les nombreuses méthodes possibles existantes, deux techniques statistiques peuvent être appliquées pour ce faire. D'une part, la **normalisation `min-max`**, comme son nom l'indique, utilise les bornes minimum et maximum de la variable. Le

principe consiste à ramener toutes les valeurs de la variable dans un intervalle de valeurs défini sur [0,1]. La technique est appliquée sans modifier le rapport des distances entre les valeurs. La normalisation `min-max` est définie par la formule suivante :

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{1.1}$$

D'autre part, la **normalisation `z-score`** a pour but de ramener la moyenne à 0 et l'écart-type à 1 en s'appuyant sur la formule ci-dessous :

$$X_{stand} = \frac{X - \mu}{\sigma} \tag{1.2}$$

Avec μ la moyenne des valeurs et σ l'écart-type.

Dans le cas d'application de cette thèse, ces techniques de mise à l'échelle ne sont pas utiles. En effet, nous verrons au cours des prochains chapitres que notre modèle est essentiellement fondé d'une part sur l'importance relative de chaque attribut et d'autre part sur les écarts de valeurs au sein de chaque attribut.

Conclusion

Dans ce chapitre, nous avons proposé un état de l'art qui présente dans un premier temps une sélection d'approches possibles pour mettre en place une solution de maintenance prédictive. Dans un second temps, nous avons décrit les différentes méthodes qui peuvent être appliquées pour améliorer la qualité d'un modèle de prédiction.

Parmi les approches de MP, nous avons notamment discuté des méthodes d'extraction de motifs dont l'un des avantages est l'interprétabilité qui peut s'avérer utile lors du processus d'industrialisation. L'exploitation de ces motifs nécessite cependant d'étudier leur corrélation à l'événement de défaillance à prédire, souvent à l'aide de règles d'association. Dans un cas d'usage industriel dynamique, ce double travail devra être effectué à plusieurs reprises pour mettre à jour la solution de MP.
De façon alternative, les approches d'apprentissage automatique nécessitent moins d'analyse manuelle. Parmi ces approches, les méthodes natives d'apprentissage supervisé nécessitent souvent de préparer les attributs afin d'améliorer la qualité de prédiction. Dans le chapitre 3 nous allons voir l'application de certaines de ces méthodes, mais également d'une approche d'apprentissage profond.

Une sous-catégorie particulière de l'apprentissage profond permet de tenir compte du caractère séquentiel des données, l'intérêt étant de pouvoir mémoriser les observations passées afin de prédire l'événement futur : il s'agit des modèles RNN. Dans les travaux

de cette thèse, nous avons exploré ces approches, dont l'application est présentée dans le chapitre 4.

Enfin, les données collectées en milieu industriel sont parfois difficiles à étiqueter, car l'association entre l'instance et l'événement de panne n'est pas connue avec précision. Le MIL permet de tenir compte de ce verrou, par structuration des données sous la forme de *bag*. Dans le chapitre 2, nous présenterons notre méthode de structuration des données de fichiers *logs* basée sur le MIL.

Chapitre 2

Méthode de structuration des données *log* pour la prédiction d'erreur critique

Introduction

Pour certains processus mécaniques, des erreurs associées à une haute sévérité sont un signal fort de la survenue imminente d'une défaillance. Aussi, chercher à anticiper la survenue de ces erreurs de haute sévérité permet de prévenir les défaillances du système. Dans ce cadre, nous proposons une approche de maintenance prédictive basée sur un modèle de classification binaire où la classe positive correspond à la survenue imminente d'une erreur de haute sévérité.

La section 2.2 présente une approche de l'état de l'art, dont nous nous sommes inspirés. Les erreurs sont structurées au moyen d'une méthode de préparation largement inspirée du *Multiple Instance Learning*, expliquée en détails dans la section 2.2 de ce chapitre. Nous présenterons notamment trois paramètres qui permettent d'ajuster le nombre d'observations utilisées pour effectuer une prédiction tout en tenant compte des contraintes de prédiction de l'utilisateur. Cette méthode de préparation des données nous permet de fournir la base nécessaire pour entraîner un algorithme d'apprentissage supervisé par classification, afin d'obtenir un modèle de prédiction.

Enfin, la section 2.3 donne une compréhension complète quant à la méthodologie permettant d'évaluer l'efficacité du modèle prédictif ainsi que des éléments de compréhension nécessaires à l'interprétation des résultats.

2.1 Travaux relatifs

À notre connaissance, peu de contributions scientifiques ont été proposées en MP sur l'analyse des données de fichiers *log*. La contribution la plus proche de nos travaux de recherche est celle de Sipos *et al.* [SFMW14], dont nous nous sommes inspirés et qui se base sur le *Multiple Instance Learning* (MIL), décrit dans le chapitre 1.

Afin de faciliter la compréhension de ce chapitre, rappelons quelques notions fondamentales relatives au MIL dans le cadre de l'application d'une approche de MP par classification. En apprentissage supervisé, un algorithme est entraîné sur un ensemble d'**instances**, chacune correspondant à un vecteur d'**attributs** (ou variables explicatives) associés à un intervalle de temps défini. Par exemple, une instance pourra être la synthèse des observations recueillies durant une journée. Le MIL permet d'attribuer une étiquette à des instances dont l'association directe à la défaillance est difficile à déterminer. Pour cela, les instances sont regroupées dans des ***bags***, qui seront étiquetés positifs s'ils sont corrélés à la survenue de la défaillance et négatifs dans le cas contraire. La stratégie d'apprentissage *bag-level* du MIL consiste à appliquer un algorithme de classification sur les

bags et non pas sur les instances. Pour cela, les instances de chaque *bag* peuvent être agrégées par la suite afin de former un substitut d'instance, que l'on appellera **méta-instance**.

L'objectif des travaux de [SFMW14] est de prédire la survenue d'un événement de défaillance sur des machines, à partir de données de fichiers *log* émis par ces mêmes machines. Les instances de leur cas d'usage reflètent les observations recueillies au cours d'une journée. Dans la démarche du MIL, les données sont regroupées dans des *bags* de 7 jours (contenant 7 instances) qui sont étiquetés positifs ou négatifs, puis un modèle de classification SVM est entraîné pour prédire la classe des instances. La méthode a permis aux auteurs de prédire la survenue d'une défaillance sur les machines avec une précision de 70 %. Toutefois, nous pensons que certains éléments de leur méthode ne sont pas adaptés à notre cas d'usage.

Pour commencer, les auteurs ne traitent pas les *bags* positifs et négatifs de manière similaire. Seuls les *bags* positifs sont agrégés en méta-instances. Par conséquent, l'ensemble de données d'apprentissage est un mélange de méta-instances positives et d'instances négatives. Dans notre cas d'application, l'événement à prédire étant relativement peu fréquent, les instances négatives sont beaucoup plus nombreuses que les instances positives. Aussi, agréger uniquement les *bags* positifs pourrait accentuer ce déséquilibre.

Par ailleurs, dans [SFMW14], lors de la phase de prédiction, le modèle s'appuie sur le contenu des instances et non des méta-instances, puis les prédictions sont propagées aux *bags*. Plus précisément, pendant la phase de prédiction, le modèle entraîné reçoit les instances d'un *bag*. Si l'une de ces instances est classée positive, alors le *bag* entier sera prédit positif. Dans le cas contraire, il sera prédit négatif.
Notons que dans l'approche de [SFMW14], lors de la préparation des données, les *bags* sont d'abord étiquetés, puis leur étiquette est propagée aux instances. En revanche, lors de la phase de prédiction, les classes des instances au sein d'un *bag* sont d'abord déterminées et ensuite la classe du *bag* est déduite en suivant l'hypothèse standard du MIL. Ainsi, fondamentalement, les prédictions sont basées sur les observations collectées durant une journée tandis que nous avons besoin de combiner des événements sur de plus grands intervalles.

Enfin, les *bags* positifs sont agrégés par moyenne des valeurs de chaque attribut. Le raisonnement qui sous-tend l'utilisation de la moyenne comme fonction d'agrégation est que la méta-instance est représentative des observations à l'échelle d'une journée. En réalité, sauf si les valeurs du jeu de données sont soumises à de très faibles écarts-types, les données d'une journée peuvent être très différentes de la moyenne des données récoltées sur 7 jours (à l'échelle du *bag*). Par conséquent, apprendre à partir des moyennes des

bags puis prédire à partir d'instances uniques peut générer une certaine divergence. Les données de notre cas d'application présentent de forts écarts-types, aussi cette manière de faire pourrait ne pas être adaptée.

Comme nous le verrons en détails dans la section 2.2, nous proposons un processus de structuration des données différent de [SFMW14], qui permet de considérer les *bags* positifs et négatifs de manière égale. Par ailleurs, nous conserverons la notion de *bag* pour la phase de prédiction du modèle.

2.2 Méthode de structuration des données basée sur le MIL

En vue de l'apprentissage, les données de fichiers *log* seront préparées et structurées selon une méthode inspirée du MIL. Dans un premier temps, nous allons voir comment construire une base de données à partir d'un ensemble de fichiers *log* puis nous traiterons de la formation d'un jeu de données en vue de l'apprentissage, basée sur le MIL.

2.2.1 Synthèse d'une source de données *log* vers un échantillon représentatif de la réalité terrain

Considérons une collection de fichiers *log* émis par une seule machine. Ces fichiers contiennent un ensemble d'erreurs distinctes par leur type et la sévérité qui leur est attribuée. Pour la méthodologie qui va suivre, les notations suivantes sont utilisés :

- Nous considérons deux niveaux de sévérité : **haut** pour les erreurs h, qui conduisent la plupart du temps à une défaillance, et **bas** pour les erreurs ℓ qui ne conduisent pas à une défaillance. Pour simplifier le discours, les erreurs de sévérité haute seront désignées par l'expression **erreur critique**.
- On note h_j une erreur critique avec $j \in \mathcal{H}$ l'un des différents types possibles d'erreur critique. De la même manière, un type particulier d'erreur de sévérité basse est noté ℓ_j.

Afin d'illustrer les propos de cette partie, un exemple de données issues d'un fichier *log* est présenté dans le tableau 2.1.

Date heure	*Type*	*Sévérité*
$2018-05-23T12:47:39.4689125$	ℓ_2	basse
$2018-05-23T17:17:12.9044737$	ℓ_1	basse
$2018-05-25T07:04:10.4582192$	ℓ_6	basse
$2018-05-25T07:08:06.1726735$	ℓ_2	basse
$2018-05-25T07:08:42.5641681$	ℓ_2	basse
$2018-05-25T07:12:24.4774971$	ℓ_3	basse
$2018-05-25T10:01:09.0527228$	ℓ_1	basse
$2018-05-26T00:35:08.2569332$	ℓ_4	basse
$2018-05-26T00:35:52.2354945$	ℓ_4	basse
$2018-05-26T04:59:22.2366952$	ℓ_3	basse
$2018-05-26T12:08:12.2369695$	ℓ_1	basse
$2018-05-27T08:32:45.8745351$	ℓ_4	basse
$2018-05-27T08:36:01.8965238$	ℓ_4	basse
$2018-05-27T08:41:18.8456163$	ℓ_3	basse
$2018-05-27T08:53:03.3565254$	ℓ_1	basse
$2018-05-29T16:28:19.9833822$	h_1	haute
$2018-05-30T08:10:26.3672368$	ℓ_4	basse
$2018-05-30T08:16:32.0671892$	ℓ_3	basse
$2018-05-30T19:58:58.7359758$	ℓ_6	basse
$2018-05-31T08:47:16.6764364$	ℓ_1	basse
$2018-05-31T10:10:27.4326789$	ℓ_1	basse
$2018-05-31T19:50:19.5946553$	h_1	haute
$2018-05-31T19:53:08.5985546$	h_2	haute

TABLEAU 2.1 – Exemple d'un fichier *log* et des erreurs qu'il contient.

Nous rappelons que notre objectif est de mettre au point un modèle de classification supervisée permettant de prédire la survenue ou la non survenue d'un erreur critique h_j. Comme nous l'avons vu dans le chapitre 1, l'apprentissage d'un modèle s'effectue sur un jeu de données composé d'**instances** qui portent une étiquette, associée à leur classe d'appartenance. Rappelons également qu'une instance est un vecteur d'attributs, ces attributs sont les variables explicatives du modèle de prédiction.
Dans un contexte de MP, pour que le modèle de prédiction ait du sens, les instances nécessaires à l'entraînement doivent être représentatives d'un échantillon de données type du cas d'application. Par exemple, si les fichiers *logs* sont émis toutes les 24 heures, la logique est de créer des instances représentatives des observations recueillies sur un intervalle de 24 heures. Ainsi, lors de la phase de prédiction, le modèle sera alimenté chaque jour par les nouvelles données émises par la machine et chaque jour une nouvelle prédiction pourra être effectuée par le modèle.

Par ailleurs, la survenue d'une défaillance, (ou d'une erreur critique), ne dépend pas uniquement de l'occurrence ou non de chacune des autres erreurs. Le nombre d'occurrences de chacune d'entre elles, au moment où elle survient, est un signal qui peut avoir de l'importance pour expliquer la survenue de la défaillance future. Afin de conserver cette information quantitative, les attributs seront élaborés dans l'inspiration de la méthode *Bag-of-Words* issue de la recherche en NLP [Gol17], c'est-à-dire que pour chaque instance, le nombre d'erreurs de chaque type est comptabilisé. De cette manière, le comptage de chaque erreur ℓ_j où h_j est considéré comme une variable explicative du modèle de prédiction. Le tableau 2.2 représente les instances du jeu de données obtenues avec la méthode *Bag-of-Words*. Chaque instance représente l'ensemble des observations recueillies sur un intervalle de 24 heures.

Date	*Unité de temps* (i)	ℓ_1	ℓ_2	ℓ_3	ℓ_4	ℓ_5	ℓ_6	h_1	h_2
$2018-05-23$	1	1	0	0	0	0	0	0	0
$2018-05-24$	2	0	0	0	0	0	0	0	0
$2018-05-25$	3	1	2	1	0	0	1	0	0
$2018-05-26$	4	1	0	1	2	0	0	0	0
$2018-05-27$	5	0	1	1	2	0	0	0	0
$2018-05-28$	6	0	0	0	0	0	0	0	0
$2018-05-29$	7	0	0	0	0	0	0	1	0
$2018-05-30$	8	0	0	1	1	0	1	0	0
$2018-05-31$	9	2	0	0	0	0	0	1	2

TABLEAU 2.2 – Exemple d'instances extraites à partir d'un fichier *log*.

Afin de mieux illustrer les étapes de la méthode présentée tout au long de cette section, un historique de données plus important est développé dans le tableau 2.3. Cet exemple de données est doté d'un historique de 13 unités de temps et contient un ensemble $\mathcal{E} = \{\ell_1, \ell_2, \ell_3, \ell_4, h_1, h_2\}$ de 6 types d'erreurs, dont deux erreurs critiques h_1 et h_2. Chaque ligne de ce jeu de données est une instance associée à une unité de temps i et à un ensemble d'attributs. Par exemple, la première instance du jeu de données, associée à l'unité de temps $i = 1$, rapporte 0 occurrence des erreurs ℓ_1, h_1 et h_2, et respectivement 12, 6 et 1 occurrence(s) des erreurs ℓ_2, ℓ_3 et ℓ_4.

Unité de temps (i)	ℓ_1	ℓ_2	ℓ_3	ℓ_4	h_1	h_2
1	0	12	6	1	0	0
2	0	0	3	2	0	0
3	0	1	4	1	1	1
4	1	0	1	2	0	0
5	0	1	1	2	0	0
6	0	1	1	1	0	0
7	0	1	1	0	0	1
8	1	0	1	8	0	1
9	0	0	6	1	1	0
10	1	0	7	1	1	0
11	2	0	0	1	3	0
12	0	0	0	2	0	1
13	2	1	0	0	2	1

TABLEAU 2.3 – Exemple d'un jeu de données de 13 unités de temps.

Pour l'entraînement du modèle de prédiction, les instances ci-dessus doivent être associées à une étiquette de classe qui sera positive si l'instance est reliée à l'occurrence d'une erreur h_j ou négative dans le cas contraire. Cette méthodologie peut également être appliquée dans un cas de classes multiples. Dans ce cas, chaque classe correspondra à l'occurrence de l'une des erreurs critiques possibles et une dernière classe sera ajoutée pour la non-occurrence de toute erreur critique.

2.2.2 Paramètres de prédiction

En considérant une erreur critique h_j particulière, l'objectif est de prédire la prochaine ou les prochaines occurrence(s) de cette erreur, compte tenu de l'observation de ses occurrences précédentes et de celles de toutes les autres erreurs. Pour répondre à l'objectif, il est important de déterminer le cadre d'observation de ces erreurs. Deux questions majeures sont exprimées derrière ce besoin : (1)"Combien d'observations passées doit-on considérer ?" (2)"Avec quel délais le modèle doit-il être capable de prédire ?" Les paragraphes qui suivent traitent ces deux questions et présentent trois paramètres nécessaires pour l'application d'une méthode de MP.

2.2.2.1 Intervalle de prédiction

Nous partons des deux hypothèses suivantes :

— La fréquence et la variété d'occurrences des différentes erreurs au cours d'un intervalle donné pourrait former un motif.
— La présence de motifs similaires apparaissant régulièrement en amont de la survenue de l'erreur critique constitue un signal précurseur.

La question que l'on peut à présent se poser est "quelle est la taille de l'intervalle idéale qui permet de circonscrire ce motif ?"

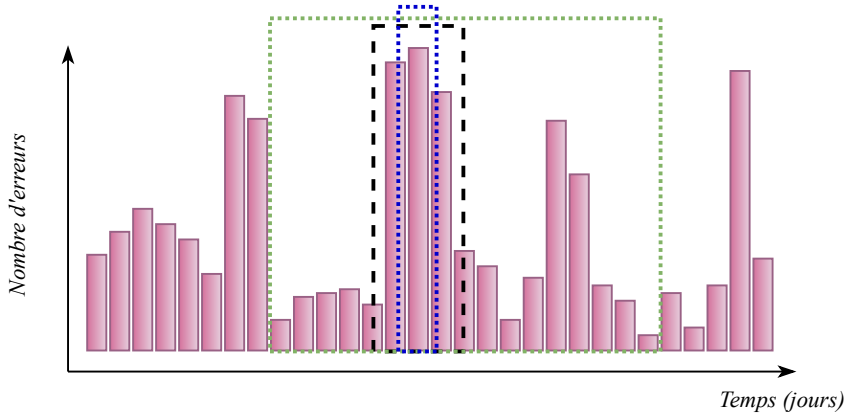

FIGURE 2.1 – Illustration du nombre d'erreurs émises en fonction du temps

Sur l'illustration de la figure 2.1, la fréquence d'émission des données issues de fichiers *log* est représentée par le diagramme en bâton. Trois fenêtres d'observation de l'historique de données sont positionnées sur la courbe.
La fenêtre noire encadre une zone de l'historique contenant un motif qui se détache suffisamment sur le profil du graphique pour être exploité en vue de prédire le prochain événement de défaillance.
La réduction de cette fenêtre, pour obtenir le cadre bleu en pointillé, entraîne le risque de ne prendre qu'une partie des observations permettant de détecter ce motif prédictif. Au contraire, une fenêtre plus grande, représentée en vert, ajoutera de nombreuses informations inutiles qui pourraient noyer le signal présent dans le motif du cadre noir.

Exemple 1. Supposons que $PI = 3$ et que l'erreur critique à prédire soit h_i. Cela signifie que la prédiction de h_i sera basée sur l'observation des données de fichiers *log* émises au cours de 3 unités de temps consécutives.

Aussi, le paramètre PI décrit une structure de données appelée le *bag*, en référence au paradigme du MIL [DLLP97] décrit dans le chapitre précédent. Pour rappel, un *bag* correspond à un ensemble d'instances. Dans notre cas d'usage, une instance est l'ensemble

des observations relatives à 24 heures d'usage de la machine et le *bag* est l'ensemble des instances successives utilisées pour effectuer la prédiction.

2.2.2.2 Intervalle d'erreur

Étant donné une erreur critique h_i, notre objectif est de construire un modèle de classification capable de prédire la survenue de h_i (classe positive) ou la non survenue de h_i (classe négative). Nous pensons qu'il peut être difficile d'obtenir un modèle satisfaisant à la fois en termes de finesse (on connaît l'instant précis auquel l'erreur critique surviendra) et de qualité (avec un faible nombre de fautes). Aussi, nous proposons d'assouplir la restriction liée à la finesse de prédiction pour ainsi favoriser la qualité du résultat. À ce titre, nous introduisons un troisième intervalle appelé l'intervalle d'erreur, dont la taille est exprimée par le paramètre EI avec $EI >= 1$.

Prédire la survenue de h_i au cours de l'intervalle d'erreur signifie que h_i peut survenir une ou plusieurs fois au cours de cet intervalle.

Exemple 2.

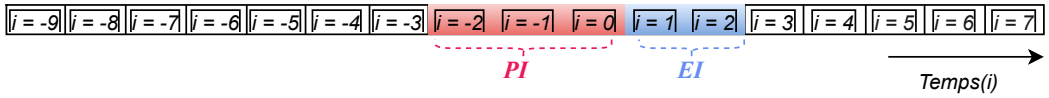

FIGURE 2.2 – Illustration de l'intervalle de prédiction, et de l'intervalle d'erreur associés respectivement aux paramètres $PI = 3$ et $EI = 2$.

La figure 2.2 schématise la situation décrite dans cet exemple. Reprenons $PI = 3$ et l'erreur critique h_i. Si $EI = 2$, cela signifie qu'à partir du moment présent ($i = 0$), nous cherchons à prédire la survenue de h_i, entre $i = 1$ et $i = 2$ en se servant des observations recueillies dans la période de $i = -2$ à $i = 0$. Si le modèle prédit la survenue future d'une erreur h_i, cela signifie que h_i peut avoir lieu ou bien à $i = 1$, ou bien à $i = 2$ ou bien à chacun de ces deux instants.

Enfin, nous allons voir à présent le dernier paramètre qui, d'un point de vue maintenance prédictive, nous permettra d'ajuster la capacité d'anticipation du modèle.

2.2.2.3 Intervalle de réactivité

Nous avons expliqué la signification et l'importance de l'intervalle de prédiction vis-à-vis de l'ensemble des observations utilisées pour prédire l'erreur critique. La méthode décrite jusqu'à présent nous permet de prédire la survenue d'une erreur critique dans un intervalle de temps EI qui suit directement l'intervalle de prédiction. Dans une

problématique de MP, les prédictions ont plus de sens à être effectuées avec un délai afin de laisser le temps nécessaire pour organiser et appliquer une opération de maintenance qui peut parfois prendre plusieurs jours. Par exemple, dans notre cas d'usage, les contraintes industrielles imposent la nécessité d'un délai d'anticipation de 5 jours. Ainsi, nous allons à présent nous intéresser à un troisième paramètre, qui permet d'ajuster ce délai d'anticipation. Concrètement, cet intervalle permet de forcer le modèle à prédire avec un délai de n unités de temps. Nous allons donc construire un jeu de données dont les instances sont constituées d'observations étiquetées positives si elles sont suivies, n unités de temps plus tard, par la survenue de l'erreur critique, sinon elles seront étiquetées négatives. Les n unités de temps se traduisent par un intervalle appelé l'**intervalle de réactivité**, contrôlé par le paramètre RI, dont les valeurs appartiennent à $\mathbb{N}^+$. Cet intervalle se situe juste après l'intervalle de prédiction et juste avant l'intervalle d'erreur. Les observations contenues dans cet intervalle sont ignorées pour construire les instances du jeu de données d'entraînement.

Exemple 3. Considérons encore $PI = 3$, $EI = 2$ et l'erreur critique h_i. Si $RI = 5$, cela suggère que l'on souhaite prédire si h_i aura lieu entre 6 et 7 unités de temps plus tard. Dans ce cas, les instances du jeu de données sont construites sur les observations survenues au minimum 6 unités de temps avant l'occurrence de h_i. Les paramètres PI RI et EI sont illustrés sur la figure 2.3.

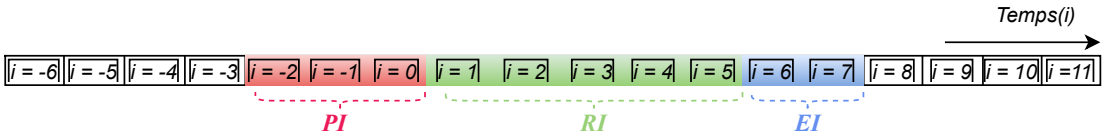

FIGURE 2.3 – Illustration de l'intervalle de prédiction, de l'intervalle de réactivité et de l'intervalle d'erreur associés respectivement aux paramètres $PI = 3$, $RI = 5$ et $EI = 4$.

Jusqu'à présent, nous avons décrit trois paramètres qui peuvent généralement être appliqués dans un contexte de MP. Notre approche étant largement inspirée du MIL, nous allons à présent nous intéresser à la partie de la méthodologie propre au MIL.

2.2.3 Apprentissage multi-instances

Notre approche, inspirée du MIL compte trois étapes fondamentales pour préparer un jeu de données d'entraînement. Ces trois étapes consistent à créer des ***bags***, leur attribuer une **étiquette**, puis les transformer en une **méta-instance**.

2.2.3.1 Construction des *bag*

L'intervalle de prédiction et son usage ont été présentés dans la section précédente. Dans l'esprit du MIL, cet intervalle de prédiction est assimilé à un ***bag***. À chaque unité de temps, un nouveau *bag* sera formé et contiendra toutes les instances comprises dans l'intervalle défini par PI.

Exemple 4.

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=6$	$i=7$	$i=8$	$i=9$	$i=10$	$i=11$	$i=12$	$i=13$
ℓ_1	0	0	0	1	0	0	0	1	0	1	1	2	0	2
ℓ_2	12	0	1	0	1	1	1	0	0	0	0	0	0	1
ℓ_3	6	3	4	1	1	1	1	1	6	1	7	0	0	0
ℓ_4	1	2	1	2	2	1	0	8	1	8	1	1	2	0
h_1	0	0	2	0	0	0	0	0	2	0	1	3	0	2
h_2	0	0	6	0	0	0	1	3	0	3	0	0	1	1

FIGURE 2.4 – Préparation des données. Les *bags* sont obtenus à l'aide des paramètres suivants : $PI = 3$; $RI = 5$; $EI = 2$.

La figure 2.4 représente l'exemple de jeu de données de la table 2.3. À partir de cet historique fictif, 4 *bags* ont été obtenus en tenant compte des paramètres suivants : $PI = 3; RI = 5; EI = 2$. Il est à noter que le dernier *bag* commence à l'unité de temps $i = 4$. En effet, si un *bag* était créé à l'unité de temps $i = 5$ il ne serait pas possible d'appliquer l'intervalle d'erreur associé.

2.2.3.2 Attribution de l'étiquette

Le processus d'étiquetage des *bags* dépend des paramètres EI et RI. Attribuer l'étiquette **positive (1)** ou **négative (0)** revient respectivement à dire :

— (1) le *bag* considéré a été suivi **au moins une fois**, RI unités de temps plus tard de la survenue de l'erreur critique. Cette erreur critique a eu lieu au cours des EI unités de temps qui suivent l'intervalle de prédiction RI.

— (0) On se projette RI unités de temps après la fin du *bag* considéré. Durant l'intervalle d'erreur associé à ce *bag*, il n'y a pas du tout eu d'occurrence de l'erreur cible.

Exemple 5. Rappelons les paramètres utilisés jusqu'à présent : $PI = 3$; $RI = 5$; $EI = 2$. Nous cherchons à prédire la survenue de l'erreur cible h_2. En se basant sur

l'exemple de données de la table 2.3, la table 2.5 présente les 4 *bags* obtenus et leur étiquette associée.

Bag	*Unité de temps* (i)	ℓ_1	ℓ_2	ℓ_3	ℓ_4	h_1	h_2	*Étiquette*
1	1	0	12	6	1	0	0	1
1	2	0	0	3	2	0	0	1
1	3	0	1	4	1	1	1	1
2	2	0	0	3	2	0	0	0
2	3	0	1	4	1	1	1	0
2	4	1	0	1	2	0	0	0
3	3	0	1	4	1	1	1	1
3	4	1	0	1	2	0	0	1
3	5	0	1	1	2	0	0	1
4	4	1	0	1	2	0	0	1
4	5	0	1	1	2	0	0	1
4	6	0	1	1	1	0	0	1

FIGURE 2.5 – *Bags* formés à partir d'un historique de données fictif associé à une machine, compte tenu des paramètres $PI = 3$, $RI = 5$ et $EI = 4$.

2.2.3.3 Synthèse des *bags*

Après obtention des *bags*, nous cherchons à baser les prédictions sur l'ensemble des informations contenues dans le *bag*. Dans le vocabulaire du MIL cela correspond à une méthode d'apprentissage *bag-level* comme nous l'avons vu dans le chapitre 1. Chaque *bag* sera donc transformé en une **méta-instance** qui servira à entraîner le modèle. Les méta-instances, tout comme les instances, sont des paires [vecteur-étiquette] où chaque vecteur représente les valeurs des attributs et l'étiquette représente l'appartenance de chaque instance à une classe. On appellera "*synthèse*" la procédure qui permet de passer d'un *bag* contenant plusieurs instances à une méta-instance. Nous distinguons deux méthodes pour synthétiser les *bags* : l'agrégation et la concaténation. Dans les deux cas, l'étiquette attribuée à la méta-instance obtenue sera héritée du *bag* à partir duquel elle a été formée.

Synthétiser les *bags* en méta-instances peut se faire simplement en créant pour chaque attribut une valeur représentative du contenu du *bag* pour l'attribut considéré. Plus exactement, soit un *bag* B_i, la méta-instance s_i associée s'obtient à l'aide d'une fonction d'agrégation : $s_i[j] = Agr(\{r[j] | r \in B_i\})$ où Agr correspond à toute fonction statis-

tique ou mathématique permettant d'obtenir une valeur unique à partir d'un ensemble fini de valeurs et $r[j]$ est l'ensemble des valeurs du *bag* B_i pour une variable explicative $r[j]$. À titre d'exemple, les fonctions statistiques `moyenne()`, `somme()`, `maximum()` et `minimum()` peuvent être utilisées pour cette tâche. Dans le chapitre 3 nous comparerons ces trois méthodes d'agrégation. Considérons les *bags* obtenus dans la table 2.5, avec le paramètre $PI = 3$ et $Agr =$`somme()`. Le *bag* B_1 est synthétisé par le vecteur $\langle 0, 13, 13, 4, 1, 1\rangle$. En effet, pour chaque erreur (ou chaque variable explicative) la valeur obtenue correspond à la somme des valeurs du *bag*, associée à la variable explicative considérée. Notons que l'agrégation s'applique aussi bien aux erreurs de faible sévérité (ℓ_i) qu'aux erreurs de haute sévérité (h_i).

Exemple 6. Sachant que $PI = 3; RI = 5$ et $EI = 3$, en prenant h_1 comme erreur critique, l'exemple de jeu de données de la table 2.6a est constitué de 4 méta-instances : deux positives et deux négatives. Ces méta-instances, rapportées dans la figure 2.6 peuvent ainsi être directement exploitées afin d'entraîner un algorithme de classification binaire.

Lorsque les *bags* sont synthétisés par agrégation, les relations temporelles entre les instances d'un *bag* ne sont pas prises en compte. Aussi, nous pouvons envisager une méthode de synthèse alternative qui consiste à concaténer les attributs des instances d'un même *bag* de manière à conserver la succession de ces instances.

Exemple 7. Soient les paramètres $PI = 3; RI = 5; EI = 2$. Le jeu de données obtenu par une méthode de concaténation est présenté dans le tableau 2.4. Pour chaque unité de temps i et chaque erreur de faible ℓ_i ou haute sévérité h_i, les attributs respectifs ℓ_{ij}/h_{ij} avec $1 \leq j \leq PI$ et $i = |\mathcal{E}|$ représentent les occurrences de ces erreurs, dans un *bag*.

Bag	ℓ_{11}	ℓ_{21}	ℓ_{31}	ℓ_{41}	h_{11}	h_{21}	ℓ_{12}	ℓ_{22}	ℓ_{32}	ℓ_{42}	h_{12}	h_{22}	ℓ_{13}	ℓ_{23}	ℓ_{33}	ℓ_{43}	h_{13}	h_{23}	*Étiquette*
B_1	0	12	6	1	0	0	0	0	3	2	0	0	0	1	4	1	2	6	0
B_2	0	0	3	2	0	0	0	1	4	1	2	6	1	0	1	2	0	0	0
B_3	0	1	4	1	2	6	1	0	1	2	0	0	0	1	1	2	0	0	1
B_4	1	0	1	2	0	0	0	1	1	2	0	0	0	1	1	1	0	0	1

TABLEAU 2.4 – Synthèse des *bags* par concaténation.

Le jeu de données obtenu par l'application de la méthodologie ci-dessus sera ensuite divisé en deux parties. La première partie, appelée jeu d'apprentissage, est utilisée pour déterminer les paramètres du modèle de prédiction. Quant à la seconde partie du jeu de données, appelée ensemble de test, elle est utilisée au cours de la phase d'évaluation, qui nous permettra de juger de la qualité du classifieur et dont les mesures sont présentées dans la section suivante.

Unité de temps (i)	ℓ_1	ℓ_2	ℓ_3	ℓ_4	h_1	h_2
1	0	12	6	1	0	0
2	0	0	3	2	0	0
3	0	1	4	1	2	6
4	1	0	1	2	0	0
5	0	1	1	2	0	0
6	0	1	1	1	0	0
7	0	1	1	0	0	1
8	1	0	1	8	0	3
9	0	0	6	1	2	0
10	1	0	7	1	1	0
11	2	0	0	1	3	0
12	0	0	0	2	0	1
13	2	1	0	0	2	1

(a) Historique fictif de 13 unités de temps associé à une machine

Bag	ℓ_1	ℓ_2	ℓ_3	ℓ_4	h_1	h_2	*Étiquette*
B_1	0	13	13	4	2	6	0
B_2	1	1	8	5	2	6	0
B_3	1	2	6	5	2	6	1
B_4	1	2	3	5	0	0	1

(b) *Bags* obtenus à partir du jeu de données fictif (tableau 2.6a), synthétisés par la fonction `SOMME`. Les paramètres utilisés sont $PI = 3; RI = 2; EI = 2$.

FIGURE 2.6 – Synthèse des *bags* par agrégation avec la fonction `somme()`.

2.3 Évaluation du modèle

Pour rappeler la procédure courante de l'apprentissage supervisé, lors de la phase d'évaluation du modèle, le jeu de test contient des instances étiquetées n'ayant jamais été présentées à l'algorithme d'apprentissage. Chaque instance dans le jeu de test sera classée à l'aide du modèle obtenu lors de la phase d'apprentissage. Pour évaluer la capacité à classifier correctement, les classes obtenues sont ensuite comparées aux étiquettes initialement attribuées aux instances du jeu de test. Les paragraphes suivants développent quelques mesures permettant de rendre compte de cette qualité de classification, que nous utiliserons dans les chapitres 3 et 4 pour présenter les résultats.

2.3.1 Métriques de classification

Pour simplifier la compréhension, et se concentrer sur l'objectif de prédiction, nous nous plaçons dans une problématique de classification à deux classes uniquement, où la classe positive correspond à la prédiction de la survenue de l'erreur h_i et la classe négative l'absence d'occurrence future de cette erreur. Comme dans toute application d'apprentissage supervisé à deux classes, nous désignerons par Vrais Positifs (VP) et Vrais Négatifs (VN), les instances appartenant respectivement à la classe positive et à la classe négative, qui sont classées correctement. Dans le cas contraire, un exemple mal classé peut être décrit par deux situations possibles :

Faux positif (FP) : le modèle a prédit la survenue de h_i, dont l'occurrence n'est en réalité pas observée. D'un point de vue applicatif, le FP est considéré comme une alerte superflue. L'erreur du modèle ne porte pas atteinte à l'objectif de prédiction de panne, mais une solution de prédiction qui émet des avertissements sur-numéraires peut s'avérer handicapant d'un point de vue utilisation.

Faux négatif (FN) : un exemple classé négatif par le modèle tandis que l'occurrence de l'erreur h_i est observée. Les FN sont les erreurs de classement les plus critiques dans l'évaluation d'un modèle de prédiction. Les raisons majeures pouvant les expliquer sont un mauvais ajustement des paramètres du modèle, un effectif du jeu de données d'entraînement trop faible ou encore un sur-ajustement du modèle sur l'ensemble d'entraînement. Ces points seront abordés plus loin dans ce chapitre.

Pour évaluer l'efficacité globale du modèle prédictif, le nombre de VP, de VN, de FP et de FN est comptabilisé. La matrice de confusion, présentée dans la figure 2.7 permet de résumer les résultats de classification par la présentation de ces quatre dénombrements.

Étiquette réelle \ Résultat de prédiction	1	0
1	Vrai Positif	Faux Negatif
0	Faux Positif	Vrai Negatif

FIGURE 2.7 – Matrice de confusion

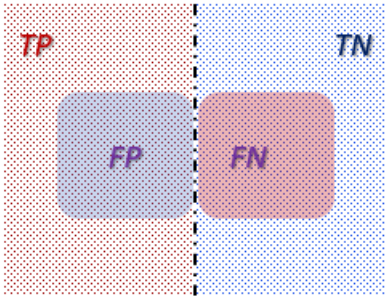

FIGURE 2.8 – Illustration des composantes d'une matrice de confusion

Par ailleurs, l'usage des mesures statistiques suivantes complète l'utilisation d'une matrice de confusion :

Sensibilité $= \frac{VP}{VP+FN}$. La sensibilité d'un modèle correspond à la capacité à détecter correctement les éléments de la classe positive.

Spécificité $= \frac{VN}{VN+FP}$. Elle se mesure parmi les exemples négatifs uniquement. En complément de la sensibilité, la spécificité permet d'évaluer la capacité du modèle à détecter les exemples négatifs.

Anti-spécificité $= 1-$ Spécificité $= \frac{FP}{VN+FP}$

À partir de ces mesures statistiques, plusieurs métriques d'évaluation sont appliquées dans la littérature. Le choix d'une métrique plutôt qu'une autre dépend en partie des exigences liées aux besoins de maintenance prédictive. Les paragraphes suivants décrivent une partie des métriques possibles en MP, auxquelles nous nous sommes intéressés.

2.3.1.1 Exactitude

La mesure d'exactitude (ou *accuracy* en anglais), donne à l'utilisateur une vue globale de la performance du modèle, toute classe confondue. Il s'agit d'établir la proportion d'exemples correctement classifiés, toutes classes confondues, parmi l'ensemble des instances.

$$exactitude = \frac{VP + VN}{VP + VN + FP + FN} \tag{2.1}$$

2.3.1.2 Précision

La mesure de précision permet d'évaluer à quel point le modèle est correct sur ses prédictions, quelle que soit la classe. Concrètement, cette mesure, définie par la

formule ci-dessous, donne la proportion d'exemples correctement classifiés, toutes classes confondues.

$$précision = \frac{VP}{VP + FP} \tag{2.2}$$

Dans un contexte de déséquilibre des classes, un modèle peut avoir un score de précision élevé pour la classe la plus fréquente, tout en étant mauvais pour la classe la moins fréquente. Par exemple, si un ensemble de test ne contient que 5 % d'échantillons positifs, un modèle qui classifie toujours les exemples dans la classe négative aura une précision de 95 %. Cependant, les prédictions peuvent contenir un nombre très faible de positifs correctement classés. La mesure de *f1-score* détaillée plus loin permet d'évaluer l'efficacité du modèle sur la classe positive, même si elle est en faible effectif.

2.3.1.3 Rappel

Le rappel correspond à la proportion d'exemples positifs correctement prédits par le modèle. Plus il est élevé, c'est-à-dire proche de 1, plus le modèle est capable de classer les positifs comme tels.

$$rappel = \frac{VP}{VP + FN} \tag{2.3}$$

2.3.1.4 *F1-score*

La définition formelle du *F1-score* par rapport à la classe positive est donnée par la moyenne harmonique de la précision et du rappel, définie par la formule suivante :

$$F1\text{-}score = \frac{2 \times VP}{2 \times VP + FP + FN} = \frac{2(precision \times recall)}{(precision + recall)} \tag{2.4}$$

2.3.1.5 Aire sous la courbe

La fonction *Receiver Operating Characteristic* (ROC), représente la sensibilité par rapport à la spécificité. Cette mesure permet d'évaluer la performance d'un classifieur binaire. Graphiquement, l'allure de la courbe donne une indication visuelle sur la performance du modèle de classification. Comme l'illustre la figure 2.9, lorsque la performance du modèle est au plus bas, la courbe ROC prend l'allure d'une fonction linéaire (dont l'ordonnée à l'origine est 0). Au contraire, l'utilisateur pourra d'autant plus faire confiance au modèle de classification que la courbe ROC associée se rapproche du point de coordonnées (0,1).

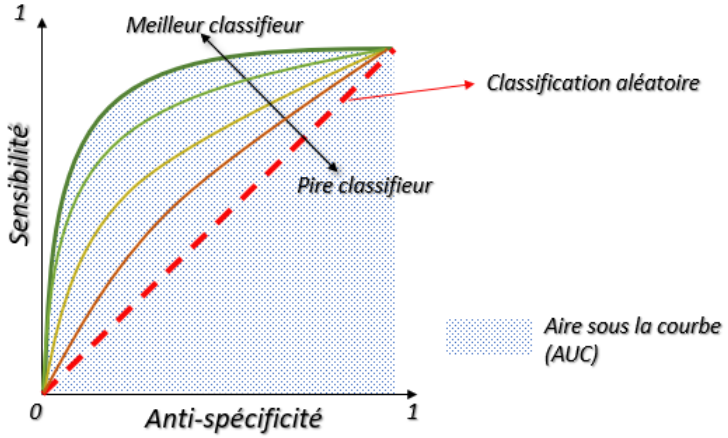

FIGURE 2.9 – Courbes ROC et AUC

2.3.2 Surveillance du sur-ajustement dans un réseau de neurones

Si l'ensemble d'apprentissage est trop large par rapport à l'ensemble de test ou si les attributs sont trop nombreux ou trop spécifiques à l'ensemble d'entraînement, le modèle peut avoir des difficultés à généraliser la solution. Cela signifie qu'il montrera de très bonnes performances pour la classification des exemples de l'ensemble d'entraînement tout en se montrant mauvais pour classifier les exemples de l'ensemble de test. Ce phénomène est appelé **sur-ajustement** ou **sur-apprentissage**.

L'utilisation des réseaux de neurones en particulier est souvent accompagnée de ces phénomènes de sur-apprentissage. Comme détaillé plus tôt dans ce chapitre, le réseau de neurone met à jour ses paramètres par un mécanisme de rétro-propagation, itéré sur un certain nombre d'*epochs*. Trop entraîner un réseau de neurones profond (DNN) peut favoriser l'apparition de ce phénomène de sur-apprentissage. Au cours de chaque *epoch*, le DNN est entraîné sur la totalité de l'ensemble d'entraînement puis évalué sur un ensemble de test appelé la **validation**. L'erreur de classification (sur laquelle le DNN s'appuie pour mettre à jour les paramètres du modèle), est tracée. Aussi, pour contrôler le phénomène de sur-apprentissage, il est possible de visualiser l'écart à chaque *epoch* entre l'erreur de classement évaluée sur le jeu d'entraînement et celle évaluée sur le jeu de validation. La figure 2.10 présente le phénomène de sur-ajustement. Les images du bas représentent les courbes d'erreurs de classement associées à l'ensemble d'entraînement (courbe bleue) et à l'ensemble de validation (courbe rouge). La situation d'apprentissage idéale est représentée par les deux illustrations de droite : le modèle est capable de séparer les deux classes dans l'espace et l'écart entre les deux courbes d'erreur de classement

est très mince. Les deux images de gauche quant à elles illustrent le cas où le réseau ne parvient pas à ajuster correctement la fonction discriminante idéale pour la classification. Enfin, un phénomène typique de sur-apprentissage est présenté sur les deux figures du milieu. La courbe bleue associée au jeu d'apprentissage présente un faible nombre d'erreurs de classement. La courbe associée au jeu de validation présente une diminution puis une augmentation progressive du nombre d'erreurs de classement, ce qui traduit le sur-ajustement du modèle aux données de l'ensemble d'entraînement.

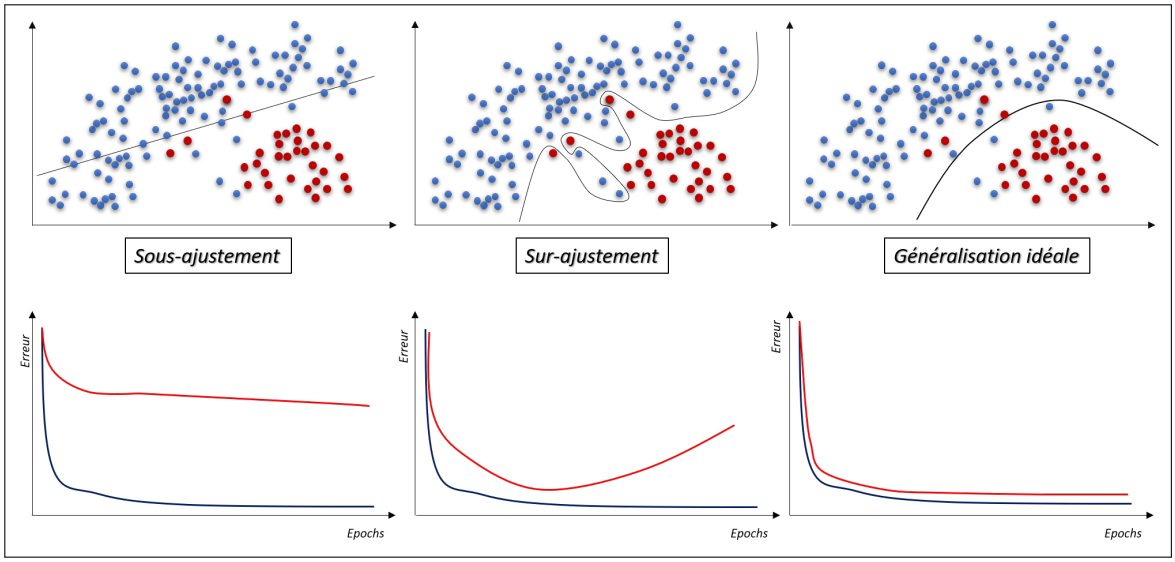

FIGURE 2.10 – Les différents phénomènes d'apprentissage. Les images du haut représentent la fonction de discrimination des valeurs approchée par l'algorithme. Les images du bas sont les courbes d'erreur de classification d'un réseau de neurones (DNN) en fonction de l'itération (*epoch*).

2.3.3 Validation croisée

Dans notre cas d'usage, les jeux de données ont des effectifs trop faibles pour valider les résultats de performance et nous ne pouvons pas faire d'augmentation de données pour atténuer ce phénomène. Aussi, pour corriger cette faiblesse d'effectif, nous avons effectué l'ensemble de nos expériences au travers d'une **validation croisée comptant 10 itérations** afin de présenter des résultats de performance les plus représentatifs possibles de la réalité.

La validation croisée est une méthode permettant de normaliser le score de performance du modèle sur les variations produites par les données. Comme le schéma 2.11 le décrit, cela consiste à itérer n fois l'entraînement et l'évaluation du modèle de classification. À chaque itération, un nouveau tirage des ensembles d'entraînement et de test est effectué

sur le jeu de données. La performance finale correspond à la moyenne des scores obtenus à chaque itération.

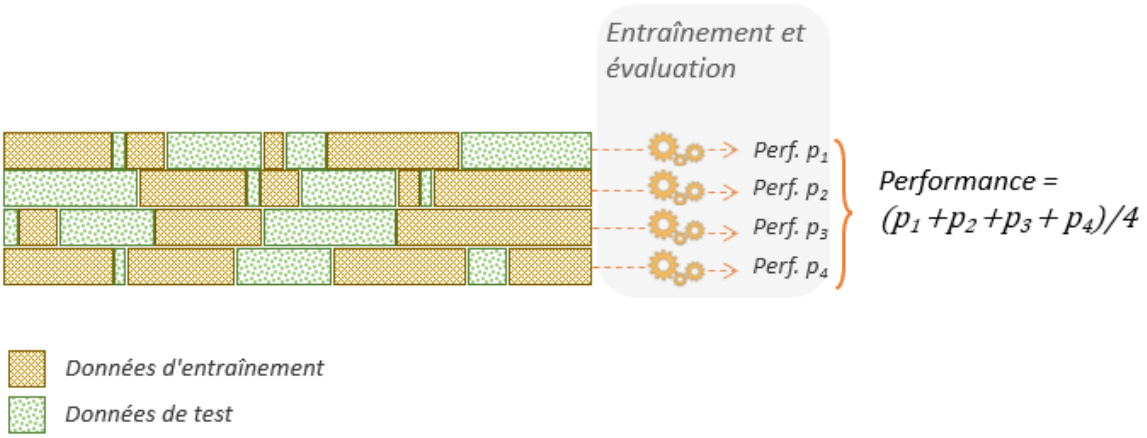

FIGURE 2.11 – Méthode de validation croisée tenant compte de 4 itérations

Conclusion

Nous avons décrit dans ce chapitre une approche permettant d'exploiter un historique de fichiers *logs* afin de construire un jeu de données dans un contexte de prédiction. Les données obtenues peuvent ainsi alimenter l'entraînement de divers algorithmes pour construire un modèle de classification binaire. L'approche proposée fait intervenir trois paramètres que l'utilisateur peut modifier pour contrôler la précision et la portée de la prédiction souhaitée. Relativement simple à mettre en place, cette méthode ne nécessite pas ou très peu de retraitement des données et peut s'appliquer en MP sur des cas d'usage similaires au nôtre. Les travaux présentés dans ce chapitre ont conduit à la publication d'articles de conférences scientifiques [LBADM21b, LBADM21a] et d'un article de journal [LBADM22].

Le chapitre suivant présente en détails l'application de cette approche, en combinaison de plusieurs algorithmes de classification.

Chapitre 3

Prédiction d'erreur critique : application à un cas d'usage industriel

Introduction

Les travaux de cette thèse ont été réalisés dans le cadre d'une application industrielle où l'objectif est de prédire la survenue d'une **erreur critique**.

Dans le chapitre 2, nous avons vu en détails la méthodologie proposée en vue de prédire ces erreurs de haute sévérité. Le présent chapitre s'attache à présenter les résultats de l'application de notre méthodologie sur notre cas d'usage industriel.

Les données utilisées pour cette application sont décrites dans la première section. Cette section expose d'une part les choix que nous avons faits pour sélectionner un échantillon candidat en vue de l'objectif de prédiction d'erreur critique. D'autre part, nous discuterons des contraintes liées aux données de **logs**. Enfin, les sections 3.2 et 3.3 sont consacrées aux résultats de l'évaluation des modèles de prédiction. Plus précisément, les résultats de la section 2.2 sont associés à une méthode de l'état de l'art, tandis que la section 3.3 présente les résultats que nous avons obtenus à partir de la méthode proposée dans le chapitre 2.

3.1 Présentation des données

3.1.1 Collecte et sélection des données

Afin d'appliquer et évaluer la méthode présentée dans le chapitre 2, nous avons collecté un échantillon de fichiers *logs*, à partir d'un **historique de 15 mois**. Dans le cadre de notre objectif industriel, nous nous sommes restreints aux machines de modèles similaires, adressées au marché de l'automobile uniquement.

Constitution des séquences

L'émission des fichiers *logs* est entachée par des interruptions non fortuites de la transmission des données, ce qui génère des manques dans le jeu de données. Les raisons de ces interruptions peuvent être expliquées par un problème de connexion de la machine au réseau, ce qui empêche la réception de l'information. De plus, pour chaque machine, une limite de capacité quotidienne de transmission restreint le volume de données reçu et stocké.

Quelles que soient les raisons pour lesquelles la collecte à un instant t n'a pas eu lieu, cette ellipse dans la série de données pourrait générer des biais lors de l'apprentissage d'un modèle de prédiction. Aussi, dans le cadre d'application de cette thèse, seules les séquences continues sur une période de 30 jours ou plus ont été conservées pour construire le modèle prédictif.

Dans l'ensemble de ce manuscrit, le terme **séquence** désigne un historique de fichiers *logs* émis par une même machine sur une durée de 30 jours minimum, sans discontinuité.

Extraction des erreurs

À partir des séquences de *logs*, seuls les événements appelés **erreur** sont extraits pour les besoins de notre objectif. Pour rappel, ces erreurs sont associées à une catégorie de sévérité qui distingue les erreurs de sévérité basse (qui n'entraînent pas de défaillance) des erreurs de haute sévérité (ou erreurs critiques). L'échantillon de données brut ainsi collecté compte **611 834** erreurs, de **332 types** différents dont **61 types de haute sévérité**, générés par **une flotte de 296 machines**.

Filtrage des données

En fonction des usines et des pays, les plannings hebdomadaires de production diffèrent. Par exemple, une usine A pourrait travailler 7 jours sur 7 tandis qu'une usine B en parallèle ne mettrait pas en route les machines durant les week-ends. De fait, les machines n'étant pas utilisées à la même fréquence dans la même semaine, les données de deux machines associées à deux politiques de production différentes pourraient ne pas être alignées sur l'échelle du temps.

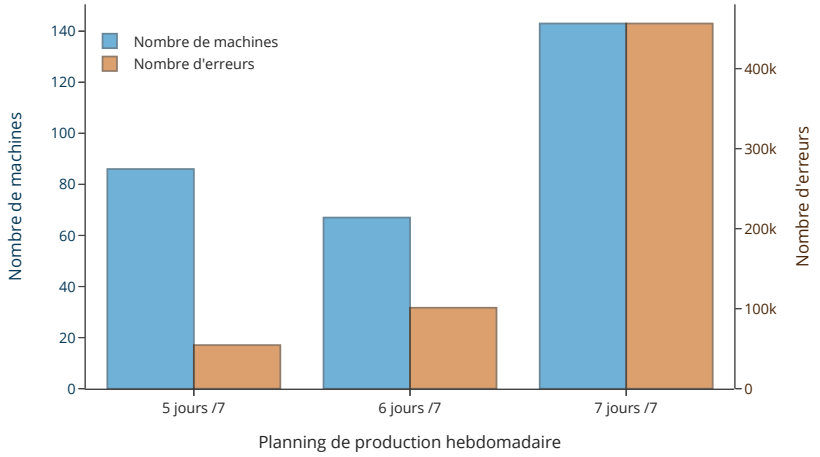

FIGURE 3.1 – Nombre d'erreurs et de machines par type de planning hebdomadaire de production.

Par exemple, une machine $M1$ sollicitée 7 jours sur 7 émettra des données sur deux journées de plus qu'une machine $M2$ utilisée seulement du lundi au vendredi. Le risque d'une telle situation, lors de la phase de l'analyse des données, est de considérer que la machine $M2$ n'a pas émis d'erreur durant ces deux jours. Pour contourner ce problème,

nous avons fait le choix de ne conserver que les données issues des machines utilisées 7 jours sur 7. La figure 3.1 présente le nombre d'erreurs et le nombre de machines en fonction du type de planning hebdomadaire de production. Comme en atteste le diagramme en bâtons, notre choix s'est porté sur le type de planning le plus représenté dans l'historique total, ce qui nous a permis de travailler avec une flotte de 140 machines et un volume d'environ 450 000 erreurs, ce qui représente plus de la moitié du volume de départ.

Étape de la collecte	*# de séquences*	*# de machines*	*# de jours*	*# de types d'erreurs*	*# d'erreurs*
Extraction des erreurs	458	296	21032	332	611834
Filtrage des données	213	143	11018	246	453050
Proportions finales	**46 %**	**48 %**	**52 %**	**74 %**	**74 %**

TABLEAU 3.1 – Volumes de données obtenus lors de la collecte.

Le tableau 3.1 présente les volumes de données dont nous disposons en termes de nombre de séquences, de machines (installées dans les différentes usines), de types d'erreurs, d'erreurs et le nombre de jours contenus dans l'historique, toutes machines confondues.
Le nombre de journées est un bon indicateur pour visualiser globalement le volume d'instances dont les jeux de données d'entraînement seront constitués plus tard. En effet, comme expliqué dans le chapitre 2, les données sont rassemblées par intervalles de 24h pour former des instances avant d'être transformées en *bags*. La proportion de chaque caractéristique par rapport aux chiffres obtenus lors de la phase d'extraction des erreurs est indiquée en gras dans la dernière ligne.

Dans le cadre de notre objectif, nous nous sommes concentrés sur la prédiction de 11 erreurs critiques, appelées **erreurs cibles**. Ces 11 erreurs ont été sélectionnées pour leur fréquence de survenue plus importante que toutes les autres ainsi que les types de pannes auxquels elles sont associées. Les pannes de notre cas d'usage correspondent le plus souvent à des problèmes de surtension des différents moteurs de la machine, des problèmes d'alimentation électrique ou d'alignement des pièces mécaniques.
Pour chacune de ces erreurs, un jeu de données a été construit selon la méthode qui a été détaillée dans le chapitre 2, et plusieurs modèles d'apprentissage, que nous décrirons dans la section 3.3 ont été entraînés sur ce jeu de données.

3.1.2 Hétérogénéité des données

L'analyse de données de type *log* en milieu industriel, est souvent associée à des contraintes d'hétérogénéité, qui peuvent rendre difficile l'élaboration d'un modèle mathématique de classification. Dans notre cas d'usage, ces contraintes d'hétérogénéité sont retrouvées à trois niveaux, que sont la matière première, la machine et l'usager, détaillés ci-après.

Matière première

Les matières traitées par les machines de découpe présentent différentes caractéristiques, générant des contraintes mécaniques variées sur la tâche de découpe. L'ensemble de ces contraintes pourra influencer de manière différente l'usure dont la machine fait l'objet. Par conséquent, les types d'erreurs (toutes sévérités confondues), émis par ces machines dépend de ces contraintes, qui sont elles-mêmes influencées par :

- La **densité** d'un support textile : il détermine l'effort nécessaire pour la découpe. Elle dépend du "grammage" [1] de la matière première, de sa composition ainsi que de la structure des fibres.
- L'**élasticité** du tissu génère une résistance, nécessitant un effort de pression plus important sur la lame de découpe.
- Les matières **thermocollantes** : elles abîment plus rapidement le fil de la lame de découpe, qui, avec la chaleur du frottement, fait fondre le tissu.
- Le **matelassage** : lors de la découpe à échelle industrielle, les matières sont assemblées en matelas et comprimées par aspiration. Selon l'épaisseur du matelas, la force d'aspiration nécessaire est plus ou moins importante, exposant par conséquent le moteur d'aspiration à des variations de contraintes. Par ailleurs, certains de ces matelas forment des structures en escaliers, ce qui suppose une pression de découpe différente selon la zone de coupe considérée.
- Les matériaux **composites** : certaines matières sont des assemblages composites de plusieurs matériaux empilés et collés, rendant la lame plus adhérente aux tissus et donc moins fluide à la découpe.

Machine

Les données ont été collectées à partir d'une flotte de machines d'ancienneté variée, ayant par conséquent une endurance variable selon les équipements.

1. Poids par m^2

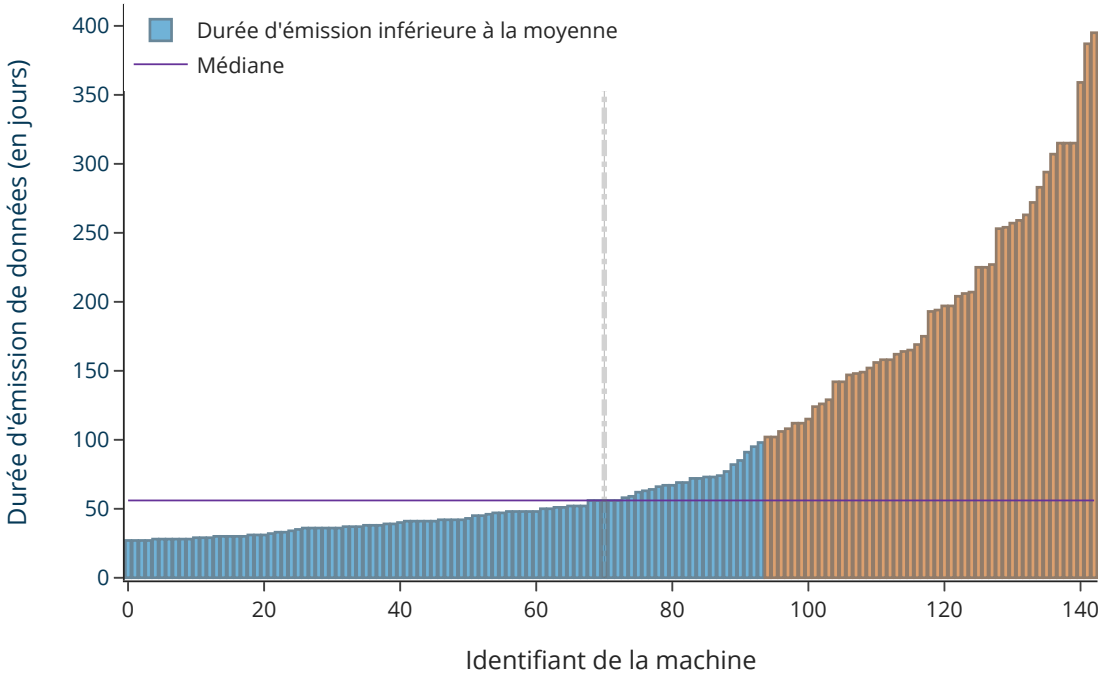

FIGURE 3.2 – Représentation pour chaque machine de sa durée d'émission de données *logs* sur la période de 15 mois.

Parallèlement, les machines étant plus ou moins anciennes, leur représentativité sur les 15 mois de collecte est également variable. En effet, une machine installée bien avant la collecte que nous avons mise en place sera plus représentée qu'une machine installée au cours des derniers mois de collecte. La figure 3.2 représente pour chaque identifiant de machine de l'axe des abscisses, le volume de données collecté sur 15 mois, traduit en nombre de jours d'émission de fichiers *log* dans l'intervalle d'activité de la machine. Les bâtons colorés en bleu représentent l'ensemble des machines dont la durée d'émission est inférieure ou égale à la moyenne du nombre de jours d'émission. Le constat le plus marquant apporté par ce graphique est que plus de la moitié des machines ont émis un nombre de fichiers *log* inférieur à la moyenne. Par ailleurs, comme nous pouvons le voir à l'aide de la médiane tracée horizontalement et la ligne pointillée, 50 % des machines ont émis durant moins de 50 jours tandis que l'autre moitié des machines ont, au contraire, une durée d'émission de fichiers allant jusqu'à 400 jours.

L'un des objectifs de cette thèse est de proposer un modèle de prédiction pour chacune des 11 erreurs cibles que nous souhaitons prédire. Ces 11 modèles doivent être génériques, c'est-à-dire qu'ils doivent pouvoir prédire la survenue de l'erreur cible quelle que soit la machine. Pour chaque erreur cible à prédire, le modèle devra déceler les motifs précurseurs

récurrents à travers les données. Si ces données sont très hétérogènes en fonction des machines, un risque évident pourrait être de ne pas parvenir à déterminer un motif commun à toutes les machines. Aussi, travailler avec des échantillons de données de petite taille (moins de 100 jours) rend cette tâche difficile. La figure 3.2 montre que le jeu de données global est constitué d'un grand nombre de petits échantillons de données, ce qui le rend par conséquent très hétérogène.

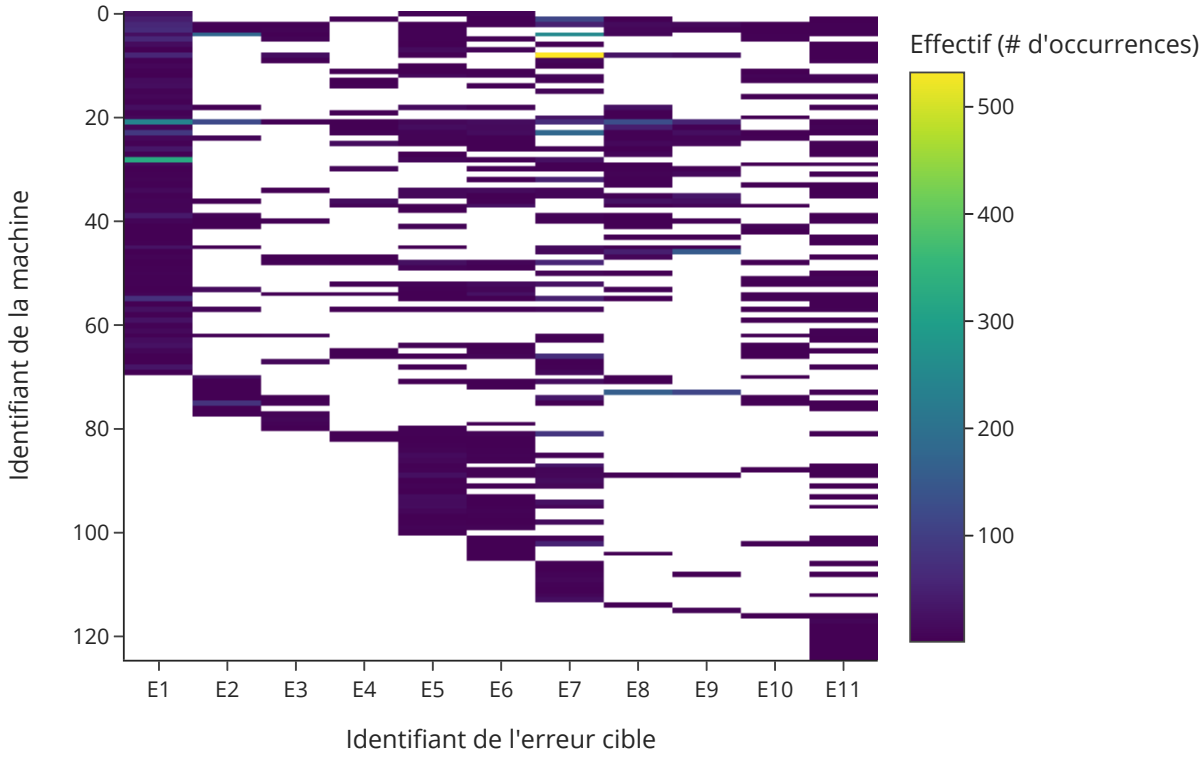

FIGURE 3.3 – Effectif des erreurs cibles par machine.

Nous partons de l'hypothèse que les différentes erreurs cibles à prédire ne sont pas associées à des motifs précurseurs identiques. Il est donc difficile de proposer une solution commune à toutes les erreurs cibles. C'est pour cette raison que nous avons fait le choix de mettre au point une solution de prédiction par erreur critique h_i. En fonction de chaque erreur critique, l'effet de l'hétérogénéité sur la qualité résultante du modèle prédictif sera plus ou moins important. Un indicateur possible pour visualiser cette différence est d'étudier la répartition des erreurs cibles par machine. La figure 3.3 présente pour chaque paire machine/erreur critique, le nombre d'occurrences de l'erreur cible, représenté par un segment de couleur dont l'échelle du gradient est indiquée sur la légende de droite. L'identifiant de la machine est représenté en ordonnée et l'identifiant de l'erreur cible

sur l'axe des abscisses. Lorsque l'erreur cible n'a jamais été émise par la machine, le segment de couleur n'apparaît pas, se traduisant par des zones blanches sur le graphique. De prime abord, nous pouvons voir que, excepté quelques cas isolés, lorsqu'une erreur critique a été émise par la machine, sa fréquence est inférieure à 100.

En termes de répartition, seules les erreurs $E1$ et $E11$ semblent concerner une grande partie de la flotte de machines. Les erreurs telles que $E2$, $E3$ et $E4$ étant très faiblement réparties à travers la population de machines, les jeux de données associés devraient donc être plus homogènes. En effet, les résultats de la figure portent à croire que ces jeux de données seront formés par un effectif plus réduit de machines. Cette observation nous amène à penser que les volumes de ces jeux de données pourraient être trop faibles pour entraîner efficacement le modèle.

Usager

L'usure et l'endurance des machines dépendent des matières premières traitées par ces équipements, mais également de la façon dont ils sont sollicités par les usagers. Étant donné l'ensemble des contraintes mécaniques régies par les matières, de nombreux paramètres peuvent être ajustés sur la machine par les utilisateurs. Par exemple, la puissance d'aspiration, la vitesse de vibration (montée et descente de la lame), la vitesse de progression de la lame, encore la pression exercée par la lame. Ces paramètres jouent à la fois sur la qualité de la découpe, l'usure à long-terme de la machine, mais également sur la productivité de l'usine. Pour un jeu de paramètres dédié à une qualité optimale, la vitesse de production peut être dégradée. À l'opposé, un jeu de paramètres dédié à une production rapide peut provoquer une usure plus rapide du matériel. Le jeu de paramètres associé à cet équilibre entre la qualité de découpe, la vitesse de production et l'usure de l'équipement est propre à chaque usine de production et dépend également des contraintes fixées par la politique de production. Par ailleurs, ces jeux de paramètres sont susceptibles d'être régulièrement modifiés au fil du temps par les ingénieurs de chaque usine en fonction de leur expérience de découpe. Par conséquent, à matière identique et machine identique, deux usines différentes pourraient ne pas utiliser les mêmes paramètres de découpe.

Les contraintes de découpe induites par la matière, les sollicitations des usagers et l'ancienneté des machines génèrent un phénomène d'hétérogénéité des données. En effet, en fonction de ces différentes contraintes, la quantité et la diversité des erreurs émises pourra varier d'une machine à l'autre.

3.1.3 Déséquilibre de classes

Rappelons les étapes de la construction des données, détaillées dans le chapitre 2. Sachant une erreur cible h_i et trois paramètres (PI, RI, et EI), associés respectivement à trois intervalles de temps (**Intervalle de Prédiction**, **Intervalle de Réactivité** et **Intervalle d'Erreur**). À partir de ces paramètres, un jeu de données d'entraînement, et de test, constitués de ***bags*** seront construits pour entraîner puis évaluer le modèle de prédiction dédié à h_i. Ces *bags*, qui correspondent à une structure de données contenant l'ensemble des échantillons compris dans l'Intervalle de Prédiction, sont par la suite transformés en **méta-instances**. La méta-instance porte la même étiquette que le *bag* et correspond à sa forme synthétisée par une fonction $\mathcal{F}$. Ces méta-instances sont étiquetées positives si l'Intervalle d'Erreur contient une occurrence de l'erreur cible h_i, sinon elles sont étiquetées négatives. Les attributs (également appelés variables explicatives ou *features*) du jeu de données correspondent aux types d'erreurs, toute sévérité confondue, y compris l'erreur critique à prédire. En plus de l'identifiant de chaque type d'erreur, nous avons inclus l'identifiant de la machine dans les attributs, afin que cet élément de variabilité soit pris en compte par le modèle de prédiction.

Pour une erreur cible h_i donnée, certaines séquences de *logs* contiennent très peu, voire pas du tout, d'occurrence de l'erreur h_i en question. À l'évidence, si la séquence contient très peu d'occurrences de l'erreur h_i, le nombre de *bags* positifs résultants pourrait s'avérer très faible par rapport au nombre de *bags* négatifs (parfois moins de 10 % du jeu de données total). Pour limiter un tel problème de déséquilibre des classes, nous avons fait le choix de retirer du jeu de données les séquences de *logs* dans lesquelles E_i n'a jamais été observée. Cette stratégie permet de limiter le déséquilibre des classes par la réduction du nombre de *bags* négatifs. Par ailleurs, certains types d'erreurs (critiques ou non) n'ont été observés que dans moins de 10 % de ces séquences de fichiers *log*. Aussi, le fait de retirer ces dernières, expose le jeu d'entraînement à un problème statistique de sparsité [2], puisque de nombreuses variables explicatives seront nulles, du fait de la présence insuffisante des erreurs associées.

Le tableau 3.2 expose à titre d'information pour chaque erreur cible les volumes de données, en termes de nombre de séquences, de nombre de machines et de nombre de *bags*, qui ne sont pas pris en compte pour la construction des jeux de données finaux. Les *bags* sont à chaque fois construits à l'aide des paramètres $PI = 7$, $RI = 7$ et $EI = 4$ et la fonction d'agrégation des *bags* $\mathcal{F} = maximum()$. Dans la section 3.3, nous discuterons

2. En statistique, la sparsité d'un jeu de données désigne le fait que beaucoup d'attributs sont absents.

des choix de sélection de ces différents paramètres de structuration des données. Il est important de noter que ce volume de *bags* correspondra plus tard aux méta-instances qui alimentent le modèle de prédiction. Pour chacun des chiffres présentés dans le tableau, les pourcentages par rapport aux volumes de l'ensemble d'entraînement complet sont indiqués en gras. Les statistiques descriptives sur les jeux d'entraînements finaux sont présentées dans le tableau 3.3.

Erreur cible	*# de séquences*	*# de machines*	*# de bags*
$E1$	117 - **55 %**	89 - **62 %**	3445 - **48 %**
$E2$	184 - **86 %**	126 - **88 %**	6275 - **87 %**
$E3$	187 - **88 %**	131 - **91 %**	6314 - **88 %**
$E4$	188 - **88 %**	133 - **93 %**	6210 - **86 %**
$E5$	145 - **68 %**	110 - **76 %**	4622 - **64 %**
$E6$	146 - **68 %**	107 - **74 %**	4477 - **62 %**
$E7$	132 - **62 %**	103 - **72 %**	4383 - **61 %**
$E8$	161 - **75 %**	114 - **79 %**	5466 - **76 %**
$E9$	180 - **84 %**	126 - **88 %**	5983 - **83 %**
$E10$	176 - **83 %**	126 - **88 %**	5665 - **79 %**
$E11$	130 - **61 %**	102 - **71 %**	3866 - **54 %**

TABLEAU 3.2 – Volumes de données non pris en compte pour la création des jeux de données, suite au retrait des séquences dépourvues de l'erreur cible.

Globalement, le fait d'avoir retiré les séquences dépourvues de l'occurrence de l'erreur cible a conduit à une réduction considérable du jeu de données. Dans 4 cas sur 11 ($E2$, $E3$, $E4$ et $E9$), plus de 80 % des *bags* ont été retirés du jeu d'entraînement. Les chiffres apportés par le tableau 3.3 indiquent que les jeux de données d'apprentissage correspondant aux erreurs $E2$, $E3$ et $E4$ sont constitués de moins de 1000 *bags* ce qui pourrait s'avérer trop peu pour entraîner les modèles de prédiction correspondants. De manière générale et en écho à ce qui a été dit précédemment à propos de l'hétérogénéité, les volumes et les contraintes des jeux d'apprentissage diffèrent complètement d'une erreur cible à l'autre. L'idéal serait de paramétrer aussi finement que possible chaque modèle en fonction de chaque erreur critique à prédire, c'est pour cette raison que nous avons fait le choix d'entraîner un modèle de prédiction par erreur cible. Cependant, modifier les paramètres de construction des données ainsi que les hyper-paramètres du modèle pour chacune de ces erreurs n'est pas l'objectif de cette thèse, qui est en réalité d'élaborer

une approche générique pour l'ensemble des erreurs critiques. Nous avons donc ajusté les hyper-paramètres des modèles en nous basant sur les résultats de la performance de prédiction pour l'erreur $E8$.

Erreur cible	*# de machines*	*# de bags*	*proportion de bags positifs*
$E1$	70	3706	22 %
$E2$	22	876	27 %
$E3$	20	837	12 %
$E4$	21	941	9 %
$E5$	57	2529	14 %
$E6$	56	2674	12 %
$E7$	58	2768	30 %
$E8$	39	1685	37 %
$E9$	21	1168	30 %
$E10$	33	1486	8 %
$E11$	65	3285	9 %

TABLEAU 3.3 – Statistiques descriptives des ensembles d'entraînement.

La suite de ce chapitre expose les résultats des expérimentations que nous avons obtenues, lors de l'élaboration des différents modèles de prédiction.

3.2 Application d'une méthode de l'état de l'art

Nous avons voulu reproduire la méthode proposée dans Sipos *et al.* [SFMW14] pour prédire la survenue de chacune des 11 erreurs cibles d'intérêt. L'implémentation originale de l'auteur n'étant pas disponible directement, nos expérimentations ont été réalisées en suivant le plus fidèlement possible les indications fournies dans l'article scientifique.

Pour reproduire le même protocole, les méta-instances positives ont été synthétisées à l'aide de la fonction `moyenne()` et les ensembles de données ont été divisés en 80 % / 20 % respectivement pour l'entraînement et le test, en suivant un processus de validation croisée en 10 itérations. Pour chaque erreur cible, le modèle de prédiction, basé sur un `SVM`, a été entraîné au moyen du logiciel `RapidMineR`[3] avec l'algorithme `libSVM`[CL11] et une méthode de sélection des variables appelée `BoLASSO` [Bac08] a permis de filtrer les

3. https://rapidminer.com

attributs sur la base de leur contribution au modèle, dont la mesure est comprise entre 0 (pas de contribution du tout) et 1 (le modèle s'appuie en totalité sur le contenu de cette variable explicative). Dans le cas d'usage de cette thèse, les meilleures performances ont été obtenues avec un seuil de sélection des variables à 0.1 (seuls les attributs ayant démontré une contribution supérieure à 0.1 sont sélectionnés). La construction des données proposée de [SFMW14] passe par l'application des paramètres PI, et RI que nous avons décrits dans le chapitre 2. Pour nos expérimentations, ces paramètres ont été chacun fixés à 7 jours, une valeur qui nous semble adaptée à notre cas d'usage, et donc aux jeux de données avec lesquels nous avons travaillé. En effet, rappelons que d'un point de vue industriel, le temps nécessaire pour réparer une machine est en moyenne d'une semaine minimum, de même que les informations qui précèdent une panne s'étendent sur un ordre de grandeur d'une semaine. Les performances obtenues sont détaillées dans le tableau 3.4 où chaque chiffre est présenté sous forme de la moyenne des valeurs du F1-score sur les 10 itérations de la validation croisée.

Malgré nos nombreuses tentatives pour ajuster la solution de [SFMW14] aux données de notre cas d'usage, nous n'avons pas pu obtenir de performances supérieures à 0.6 en termes de F1-score.
En moyenne, les scores des différents modèles de prédiction sont autour de 0.3 et en ce qui concerne les erreurs $E3$, $E4$ et $E6$ ces scores sont même inférieurs à 0.1 : nous en concluons que le modèle n'est pas adapté, en particulier pour prédire ces 3 erreurs.

Erreur cible	E1	E2	E3	E4	E5	E6	E7
F1score [SFMW14]	0.344	0.368	0.057	0.015	0.315	0.16	0.487

Erreur cible	E8	E9	E10	E11	*moyenne*
F1-score [SFMW14]	0.457	0.553	0.033	0.164	**0.268**

TABLEAU 3.4 – Performances en termes de moyenne des F1-score obtenus avec une validation croisée de 10 itérations, relatives à chaque erreur cible lorsque le modèle de prédiction est élaboré avec la méthode de [SFMW14]. Les hyper-paramètres utilisés sont $PI = 7, RI = 7, EI = 1$ et la régularisation $\lambda = 0.15$.

Nous avons appliqué une méthode de l'état de l'art sur notre cas d'usage. Cette méthode ne nous a pas permis d'obtenir des résultats suffisamment satisfaisants au vu de notre besoin de maintenance prédictive. La section suivante présentera l'application de notre solution de MP, décrite dans le chapitre 2.

3.3 Proposition d'amélioration et application

Pour toutes les expérimentations qui suivent, les paramètres de construction des *bags* utilisés par défaut sont les suivants : $PI = 7$; $RI = 7$; $EI = 4$. Nous avons choisi d'utiliser la fonction $\mathcal{F} = maximum()$ pour synthétiser les *bags*, de manière à favoriser dans un premier temps les hautes différences de valeurs entre les instances. Chaque jeu de données a été partitionné en 10 fois par validation croisée en un ensemble d'entraînement et un ensemble de test avec un ratio de 80 % / 20 %. Afin de s'assurer que les jeux d'entraînement et de test conservent une proportion similaire d'instances positives, le partitionnement est effectué par échantillonnage stratifié.

3.3.1 Comparaison des algorithmes traditionnels d'apprentissage

Pour chaque erreur cible, un classifieur binaire est entraîné, à l'aide du logiciel RapidMiner[4], pour prédire l'occurrence ou non de l'erreur cible au cours des EI jours qui suivent la période RI. Dans cette optique, les quatre algorithmes de classification ci-dessous ont été explorés. Leur fonctionnement a été présenté de façon détaillée dans le chapitre 2.

Un modèle `SVM` [Vap00], implémenté dans l'algorithme `libSVM` [CL11]. Il est entraîné avec un noyau non linéaire de type fonction de base radiale, dit RBF pour l'anglais *Radial Base Function*.

Un modèle Bayésien naïf [DH73], associé à une correction de Laplace : chaque valeur est augmentée de 1 pour supprimer les valeurs nulles, qui peuvent biaiser le résultat du modèle.

Un Arbre de Décision (`AD`) [Wu75], de profondeur limitée à 10. Un élagage de l'arbre est appliqué, et le rapport de gain décrit dans le chapitre 2 a été utilisé en tant que critère de sélection des attributs dans le processus décisionnel.

Une Forêt Aléatoire (`FA`) [Bre01], constituée de 100 `AD` de profondeur limitée à 10 et sur lesquels l'élagage est permis.

Les performances de prédiction pour chacune des 11 erreurs critiques sont présentées dans les tableaux 3.5 pour l'exactitude, 3.6 pour le score de rappel et 3.7 pour la mesure du F1-score. Pour chaque erreur, le score obtenu le plus élevé parmi les quatre classifieurs est représenté en gras dans les tableaux.

4. https ://rapidminer.com

Concernant le score d'exactitude dans le tableau 3.5, l'`AD`, la `FA` et l'algorithme `libSVM` contribuent aux meilleures performances avec une valeur autour de 0.8. Ce score signifie que dans 80 % des cas, ces modèles de classification sont capables de prédire correctement l'occurrence ou la non-occurrence de l'erreur E_i.

	Exactitude			
Erreur cible	Bayésien Naïf	Arbre de décision	Forêt aléatoire	libSVM
E1	0.296238	**0.825364**	0.821723	0.824636
E2	0.480928	0.837629	0.839691	**0.851031**
E3	0.512366	**0.902151**	0.898925	0.895699
E4	0.527404	0.917308	**0.918269**	0.914904
E5	0.313036	**0.882321**	0.876250	0.881964
E6	0.317905	**0.895270**	0.889696	0.893581
E7	0.403420	0.800651	0.791531	**0.810749**
E8	0.448267	0.789333	0.795733	**0.835733**
E9	0.473563	0.796935	0.793487	**0.862452**
E10	0.407576	**0.926364**	0.920909	0.918182
E11	0.323056	**0.918690**	0.916508	0.917735

TABLEAU 3.5 – Scores d'exactitude, relatifs à 5 classifieurs différents utilisés pour prédire individuellement 11 erreurs cibles.

	Rappel relatif à la classe positive			
Erreur cible	Bayésien Naïf	Arbre de décision	Forêt aléatoire	libSVM
E1	**0.976974**	0.083553	0.044737	0.074342
E2	**0.969048**	0.259524	0.280952	0.416667
E3	**0.947619**	0.314286	0.109524	0.104762
E4	**0.922222**	0.333333	0.122222	0.194444
E5	**0.983333**	0.111111	0.041667	0.136111
E6	**0.962687**	0.114925	0.040299	0.092537
E7	**0.943357**	0.152448	0.111189	0.221678
E8	**0.925490**	0.251961	0.259804	0.509804
E9	**0.985246**	0.183607	0.137705	0.547541
E10	**0.907692**	0.150000	0.038462	0.015385
E11	**0.978689**	0.072131	0.000000	0.014754

TABLEAU 3.6 – Scores de rappel, relatifs à 5 classifieurs différents, utilisés pour prédire individuellement 11 erreurs cibles.

Les meilleurs scores de rappel associés à la classe positive (tableau 3.6) sont obtenus par l'`AD`, la `FA` et le classifieur `SVM`. Pourtant, le score reste en-dessous de 0.54 : parmi toutes les occurrences de l'erreur critique attendues, au maximum 54 % d'entre elles sont détectées par l'un de ces trois modèles.
Le modèle Bayésien Naïf est meilleur en termes de score de rappel que d'exactitude. En effet, le score de rappel est en général supérieur à 0.9, il s'avère donc meilleur pour prédire la survenue de l'erreur cible que pour prédire la non-survenue.

Intéressons-nous maintenant aux performances des modèles, traduites par la mesure du F1-score. Comme expliqué dans le chapitre 2, la valeur du F1-score est représentative à la fois du score de rappel et de celui de la précision. Plus la valeur de cette mesure est proche de 1, plus la classification est correcte. Parmi les 11 erreurs cibles d'intérêt, seules les erreurs $E2$, $E8$ et $E9$ ont été prédites avec un F1-score supérieur à 0.5. Cette performance a été obtenue par l'utilisation de l'algorithme `SVM`

	F1-score			
Erreur cible	Bayésien Naïf	Arbre de décision	Forêt aléatoire	libSVM
E1	**0.338711**	0.149524	0.084147	0.134486
E2	0.4471	0.407954	0.429805	**0.545993**
E3	0.3051	**0.413308**	0.191240	0.179745
E4	0.2533	**0.405576**	0.199790	0.277620
E5	**0.2690**	0.194414	0.078897	0.226055
E6	**0.2422**	0.196806	0.076008	0.164167
E7	**0.4242**	0.261390	0.198023	0.352383
E8	0.478	0.392925	0.407459	**0.627501**
E9	0.467	0.295606	0.235831	**0.649217**
E10	0.1944	**0.242484**	0.070101	0.028378
E11	**0.194014**	0.125803	0.000000	0.028776

TABLEAU 3.7 – Performances en termes de F1-score, relative à 5 classifieurs différents, utilisés pour prédire individuellement 11 erreurs cibles.

Comme nous l'avons vu dans le tableau 3.3, seuls les jeux d'entraînement pour les erreurs $E2$, $E8$ et $E9$ présentent une proportion de la classe positive supérieure à 20 %. En termes de F1-score, les meilleures performances obtenues par le modèle `SVM` sont associées à ces trois mêmes erreurs. Cela démontre simplement que le modèle `SVM` est très sensible au déséquilibre des classes. Si le déséquilibre des classes est effectivement la raison de ces mauvais résultats de F1-score, on pourrait s'attendre à obtenir un gain substantiel par ré-échantillonnage du jeu de données.

3.3.2 Correction du déséquilibre de classes

La prédiction de défaillances à l'aide de techniques d'apprentissage automatique est couramment confrontée au problème du déséquilibre des classes. En effet, les défaillances (dans notre cas, les erreurs critiques) ne se produisent pas fréquemment, ce qui pourrait générer beaucoup plus d'échantillons négatifs que d'échantillons positifs. Comme nous avons pu le voir précédemment, les algorithmes de classification sont plus ou moins sensibles à cet aspect. Traditionnellement, un important déséquilibre de classes peut être corrigé soit en sous-échantillonnant la classe majoritaire (négative dans notre cas d'usage), soit en sur-échantillonnant la classe minoritaire (ici, la classe positive).

Nous avons pu observer dans la section précédente que les meilleures performances des modèles de prédictions analysés, ont été atteintes pour les erreurs cibles dont les ensembles d'apprentissage sont moins déséquilibrés. Cette section présente les résultats des expérimentations effectuées en suivants ces techniques de sur- et/ou sous-échantillonnage, qui semblent être une bonne piste.

3.3.2.1 Sous-échantillonnage

Le sous-échantillonnage de la classe majoritaire consiste à sélectionner de manière aléatoire un sous-ensemble d'exemples appartenant à cette classe. Dans notre cas, la classe négative est la classe majoritaire, nous avons donc sous-échantillonné l'ensemble d'apprentissage en prélevant respectivement 25 %, 50 % et 75 % des échantillons négatifs. Rappelons qu'une forme de sous-échantillonnage a déjà été effectuée lors de la construction des jeux d'entraînement, puisque pour chaque erreur E_i spécifique, nous avons fait le choix de retirer les séquences de *logs* dépourvues de l'occurrence de E_i.

Les performances du modèle en termes de F1-score, après l'application de cette méthode, sont présentées dans la figure 3.4.

Globalement, la méthode de sous-échantillonnage améliore les performances des algorithmes de classification. Cependant, et particulièrement en ce qui concerne le Bayésien naïf et l'`AD`, le gain résultant du sous-échantillonnage n'est pas significatif (moins de 0.1). Par ailleurs, les gains de performance induits par le sous-échantillonnage sont largement variables en fonction des erreurs critiques, ceci étant dû à la différence de fréquence de chaque erreur cible (le tableau 3.3 démontre des proportions d'instances positives très variables d'une erreur cible à l'autre).
Les expérimentations suivantes présentent une méthode de sous-échantillonnage permettant de s'adapter à la proportion de positif du jeu de données.

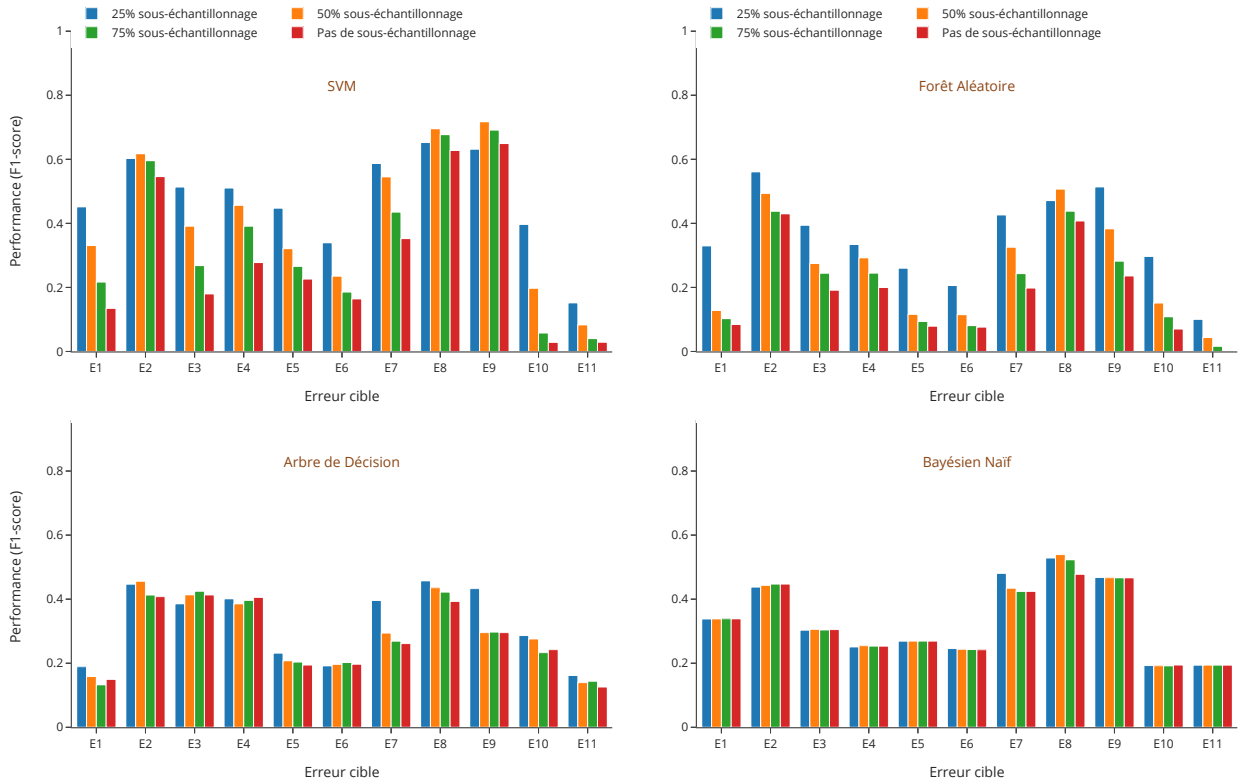

FIGURE 3.4 – Effet du sous-échantillonnage sur 4 classifieurs ($PI = 7; RI = 7; EI = 4$)

3.3.2.2 Sous-échantillonnage adaptatif

Comme nous venons de le dire, la difficulté de notre cas d'usage réside à la fois dans l'hétérogénéité des données et le déséquilibre des classes. En effet, la quantité d'instances positives diffère selon les jeux de données associés aux erreurs cibles. C'est pourquoi une autre méthode de sous-échantillonnage a été explorée afin de maintenir des proportions d'instances négatives homogènes à travers les différents jeux de données, et ainsi rendre nos résultats comparables.

Soient les paramètres r, P et N respectivement associés à un ratio d'instances négatives, au nombre d'instances positives et au nombre d'instances négatives contenues dans l'échantillon initial.
La proportion d'instances négatives r' à conserver est obtenue à l'aide de la formule du

sous-échantillonnage adaptatif suivante :

$$r' = r \times (P/(N \times (1 - r)))$$

Dans le cas où la proportion de négatifs effective est plus faible que la proportion fixée par l'utilisateur, r' sera supérieur à 1. Le sous-échantillonnage ne sera alors pas appliqué.

Cette procédure est pourtant à considérer avec prudence. En effet, si la faiblesse des performances s'avère être dûe à un manque d'information discriminante, la suppression aléatoire des instances négatives réduit certainement le poids de cette classe majoritaire sans pour autant résoudre le problème sous-jacent. Une alternative potentielle consiste au contraire à multiplier les instances positives afin d'augmenter le poids de la classe minoritaire, et donc appliquer un sur-échantillonnage.

3.3.2.3 Sur-échantillonnage

Le sur-échantillonnage de la classe minoritaire peut se faire de plusieurs façons. Naïvement, il suffirait, de répliquer en plusieurs exemplaires chacune des instances de la classe minoritaire. Cette méthode ne permettant pas de diversifier les valeurs des exemples positifs, d'autres techniques plus sophistiquées sont, au contraire, pensées pour générer de nouveaux exemples synthétiques, similaires aux originaux, mais pas identiques.

Par exemple, `SMOTE` [CBHK02] génère de nouveaux exemples synthétiques par interpolation. Un exemple réel p est choisi au hasard parmi les positifs, puis un autre exemple q est sélectionné parmi ses k plus proches voisins. Placé à mi-chemin entre p et q, dans l'espace des variables, un nouvel exemple synthétique est généré. Ce processus est répété jusqu'à ce que le nombre d'exemples positifs soit égal au nombre d'exemples négatifs : l'ensemble de données obtenu est alors complètement équilibré.

Plusieurs extensions de `SMOTE` ont été proposées dans la littérature, parmi lesquelles `ADASYN` [HYGS08], qui est également souvent utilisé pour traiter ces problématiques. Intuitivement, l'approche `ADASYN` se concentre sur certains échantillons spécifiques qui sont jugés difficilement prévisibles et génère de nouveaux exemples synthétiques similaires à ceux-ci.

Une troisième alternative est celle de Liu *et al.* [LAH06], qui a montré les avantages de la combinaison de la méthode `SMOTE` avec un sous-échantillonnage.
Pour évaluer le gain attendu des méthodes de sur-échantillonnage par rapport à l'absence de correction, nous avons comparé `SMOTE`, `ADASYN` et avons également combiné 3 taux de sous-échantillonnage avec `SMOTE` :

— `SMOTE` + un sous-échantillonnage à 25 % ;

— `SMOTE` + un sous-échantillonnage à 50 % ;
— `SMOTE` + un sous-échantillonnage à 75 % ;

Les performances en termes de F1-score sont présentées à l'aide des histogrammes de la figure 3.5, pour chaque algorithme de classification candidat et pour chaque erreur cible d'intérêt. Il est important de noter que pour chaque expérience, les ensembles de données obtenus sont toujours parfaitement équilibrés.

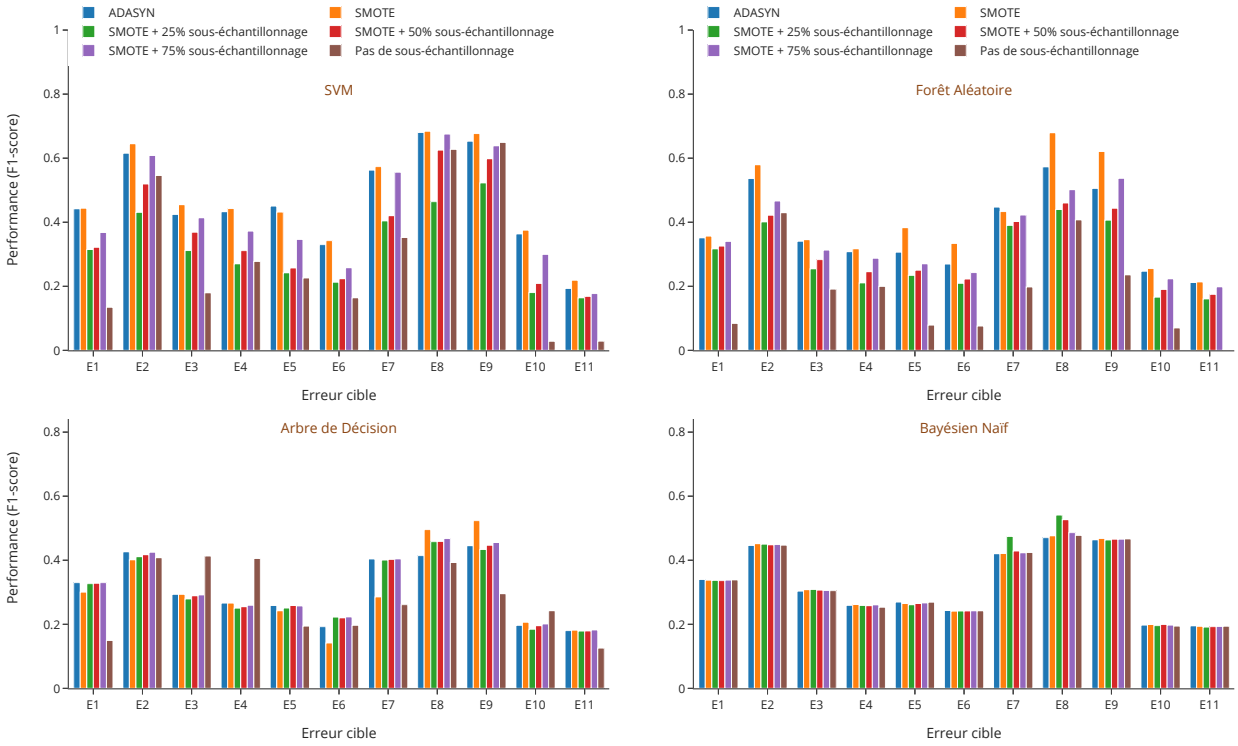

FIGURE 3.5 – Évaluation de l'effet du sur-échantillonnage avec quatre algorithmes de prédiction, lorsque les paramètres des jeux de données sont $PI = 7; RI = 7; EI = 4$.

Pour la plupart des erreurs, le gain apporté par le ré-échantillonnage de l'ensemble d'entraînement ne permet pas d'atteindre une performance suffisante de notre point de vue. En effet, à quelques exceptions près (voir par exemple les erreurs $E2$, $E8$ et $E9$), la valeur du F1-score reste inférieure à 0.6 pour tous les algorithmes candidats.
Les résultats spécifiques à l'`AD` et au modèle Bayésien démontrent même qu'il n'y a pas de différence significative, lorsque le jeu d'entraînement est sur-échantillonné, même si l'on combine le sur-échantillonnage à un sous-échantillonnage.

À ce stade de nos explorations, nous n'avons pas pu obtenir de modèle capable de prédire l'occurrence d'une erreur critique avec une F1-score supérieur à 0.6. Aussi, nous avons expérimenté un autre modèle de prédiction, basé sur un réseau de neurones artificiels.

3.3.3 Apprentissage au moyen d'un réseau de neurones artificiels

Le nouveau modèle de prédiction que nous avons exploré dans l'objectif de cette thèse correspond à un réseau de neurones artificiels profonds multi-couches, aussi connu en anglais sous le terme *multi-layer feed-forward artificial neural network*. Dans la suite de ce manuscrit, nous utiliserons l'acronyme `DNN` (*Deep Neural Network*) pour désigner ce nouveau modèle. L'architecture du DNN est constituée de quatre couches de 100 neurones. L'algorithme d'optimisation correspond au gradient de descente stochastique, réadapté et fournit directement par le *framework* H2o [CPLA16], et la fonction de coût utilisée dans ces expérimentations correspond au calcul de l'entropie croisée. Chaque expérience est réalisée avec 1000 *epochs*, en conservant les valeurs de paramètres $PI = 7$, $RI = 7$, $EI = 4$ et la fonction d'agrégation des *bags*, `maximum()`.

Erreur	*ré-échantillonnage*	$F1$-*score* [SFMW14]	$F1$-*score* (1)	$F1$-*score* (2 -DNN)
E1	`SMOTE`	0.344	0.444	0.688
E2	`SMOTE`	0.368	0.645	**0.731**
E3	`SMOTE`	0.057	0.455	**0.737**
E4	`SMOTE`	0.015	0.443	0.616
E5	`ADASYN`	0.315	0.451	0.637
E6	`SMOTE`	0.160	0.343	0.678
E7	`SMOTE`	0.487	0.574	**0.783**
E8	`SMOTE`	0.457	0.684	**0.795**
E9	`SMOTE`	0.553	0.677	**0.804**
E10	`SMOTE`	0.033	0.375	0.591
E11	`SMOTE`	0.164	0.219	0.552

TABLEAU 3.8 – Les meilleurs scores de performance obtenus pour chaque erreur critique E_i, associés au modèle de classification et à la technique de ré-échantillonnage utilisée, comparés aux performances du modèle `DNN`.

Le tableau 3.8 confronte les meilleurs résultats précédemment obtenus (colonne intitulée "$F1$-*score* (1)"), aux performances du modèle `DNN` (colonne intitulée "$F1$-*score* (2-DNN)") ainsi qu'aux performances obtenues avec la méthode de [SFMW14]. Les chiffres sont obtenus par moyenne des performances issues de la validation croisée en 10 itérations. La seconde colonne indique la méthode de ré-échantillonnage éventuellement

utilisée pour obtenir les chiffres de la "$F1\text{-}score$ (1)". Les chiffres de F1-score supérieurs à 0.7 sont présentés en gras, la valeur à partir de laquelle nous considérons que l'efficacité du modèle prédictif est suffisante pour notre cas d'usage. À titre de comparaison, la dernière colonne rapporte les résultats relatifs à la méthode de [SFMW14] (colonne "$F1\text{-}score$ [SFMW14]").

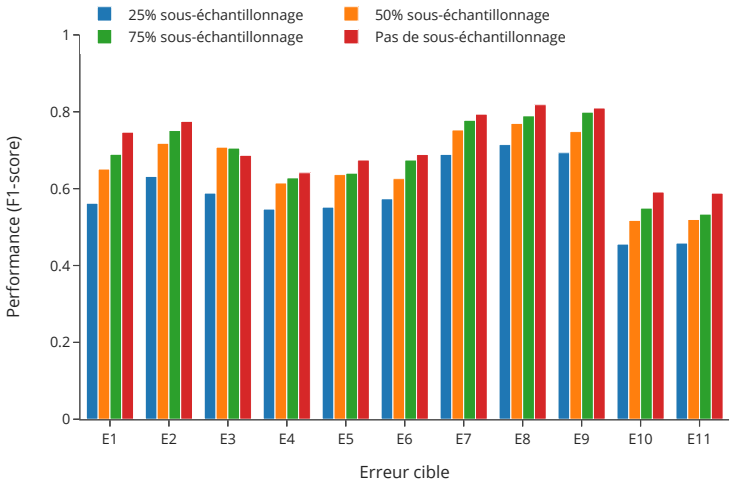

FIGURE 3.6 – Évaluation de l'effet du sous-échantillonnage sur la performance du modèle `DNN`.

Le constat global de cette comparaison est que quelle que soit la correction de déséquilibre de classes utilisée en combinaison des algorithmes traditionnels, le modèle `DNN` est significativement plus performant sans cette correction et pas seulement en moyenne, mais également pour chaque erreur cible concernée. Par conséquent, à ce stade, le meilleur candidat en tant que solution de maintenance prédictive est le modèle `DNN`.

De la même manière que pour les quatre autres classifieurs utilisés dans les expériences précédentes, nous avons cherché à améliorer les performances du modèle `DNN` en corrigeant le déséquilibre des classes par des techniques de ré-échantillonnage. La figure 3.6 présente les performances obtenues après un sous-échantillonnage des exemples négatifs de 25 %, 50 % et 75 %. Parallèlement, la figure 3.7, présente les performances du modèle `DNN` spécifiques à la prédiction de l'erreur $E8$, en fonction de différentes valeurs de **sous-échantillonnage adaptatif**. La constante représentée en orange est la valeur de performance du modèle lorsque aucune technique de ré-échantillonnage n'est appliquée.

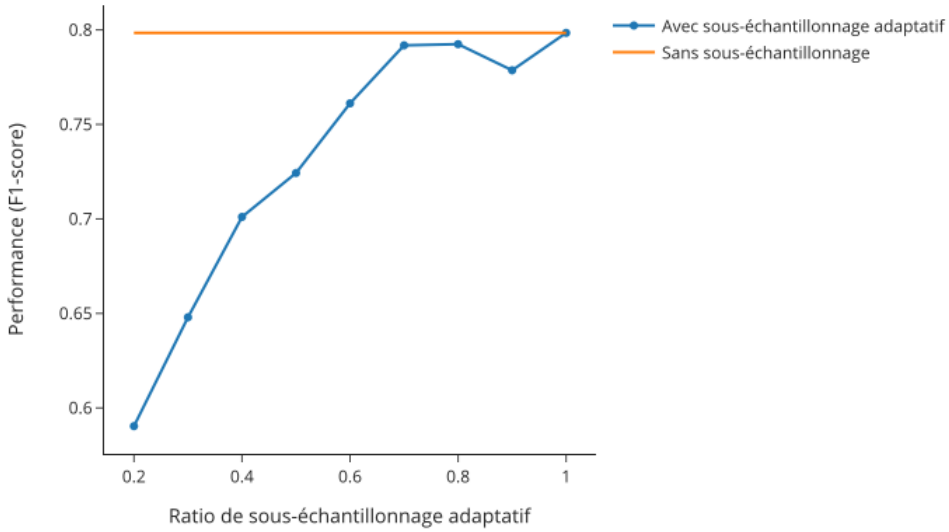

FIGURE 3.7 – Évaluation de l'effet du sous-échantillonnage adaptatif sur la performance du modèle `DNN` associé à l'erreur $E8$.

Sans surprise, non seulement la performance du modèle ne s'améliore pas avec le sous-échantillonnage, mais l'effet obtenu tend plutôt vers une dégradation. Cela confirme que le modèle `DNN` est capable, avec peu d'échantillons positifs, de déceler suffisamment les motifs précurseurs de la survenue de l'erreur cible.

Le fait de retirer les échantillons négatifs n'a pas permis d'améliorer significativement les performances de prédiction pour la classe positive. Nous pensons que cette méthode pourrait globalement écarter des *bags* utiles pour améliorer la qualité de prédiction. Nous avons choisi de poursuivre nos analyses avec le modèle `DNN`, qui s'avère être le meilleur candidat, sans appliquer de correction du déséquilibre de classe.

3.3.4 Synthèse des *bags*

La manière dont les *bags* sont transformés en méta-instances modifie la topologie des données. Par conséquent, selon le choix effectué, les motifs précurseurs de l'erreur cible seront plus ou moins difficiles à discriminer. Aussi, nous avons comparé l'application des fonctions d'agrégation `somme()`, `minimum()`, `moyenne()` et la méthode de concaténation, qui ont été détaillées dans le chapitre 2, à celle de la fonction d'agrégation `maximum()`. Les résultats obtenus sont présentés dans la figure 3.8. Les fonctions `maximum()`, `somme()` et `moyenne()` fournissent les meilleures performances, mais la variation des valeurs pour les scores obtenus reste négligeable. Aussi, nous avons choisi de poursuivre les expérimentations suivantes à l'aide de la fonction d'agrégation `maximum()`.

Il reste encore à évaluer l'effet des paramètres PI, RI et EI sur les performances du modèle `DNN` et déterminer la combinaison permettant d'obtenir les meilleures performances.

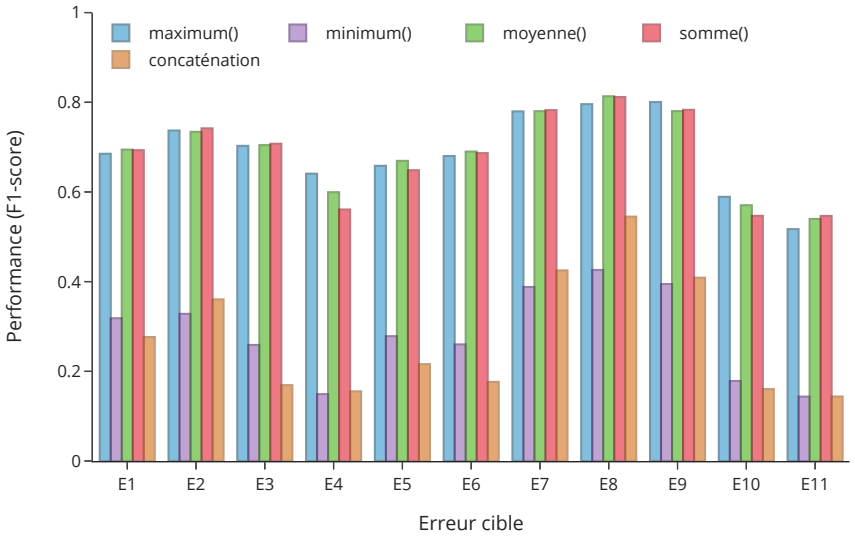

FIGURE 3.8 – Performances du `DNN` (F1-score) pour chaque erreur E_i, en fonction de cinq méthodes de synthèse des *bags*, lorsque $PI = 7; RI = 7; EI = 4$

3.3.5 Impact des paramètres de prédiction

Comme nous l'avons expliqué dans le chapitre 2, l'intervalle de prédiction définit l'historique des données utilisées pour la phase d'entraînement et la phase de prédiction. En pratique, il est difficile de savoir quelle valeur doit être attribuée à PI pour obtenir les meilleures performances de prédiction.

Pour toutes les expériences précédentes, nous avons fixé $PI = 7$ sans certitude que la valeur 7 soit le meilleur choix pour notre cas d'usage. En effet, la durée sur laquelle les erreurs en rapport avec la défaillance sont émises est un élément déterminant pour la qualité des prédictions. Rappelons que la taille minimale d'une séquence de fichiers *log* est de 30 jours. Aussi, pour analyser l'impact de PI, nous avons mené une expérience en faisant varier les valeurs de ce paramètre dans un intervalle de 1 à 29 jours. De plus, afin de comparer les performances du modèle à différentes valeurs de PI, nous avons introduit un autre paramètre temporel appelé H où $H = PI + RI$ qui tient compte en parallèle du paramètre RI. Comme le montre la Figure 3.9, lorsque PI est augmenté, RI diminue et inversement.

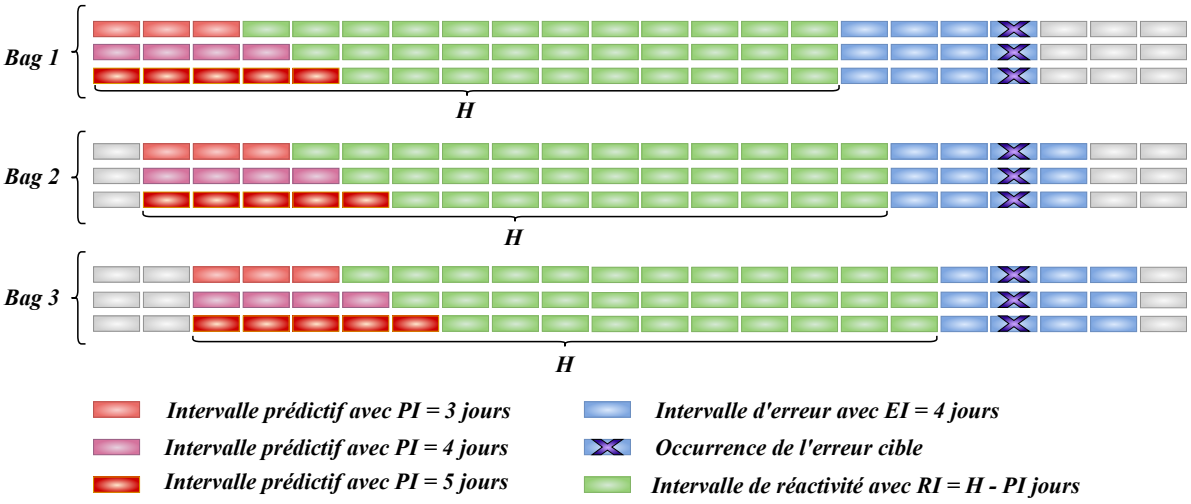

FIGURE 3.9 – Méthode utilisée pour faire varier la taille de l'intervalle PI au sein de l'intervalle H.

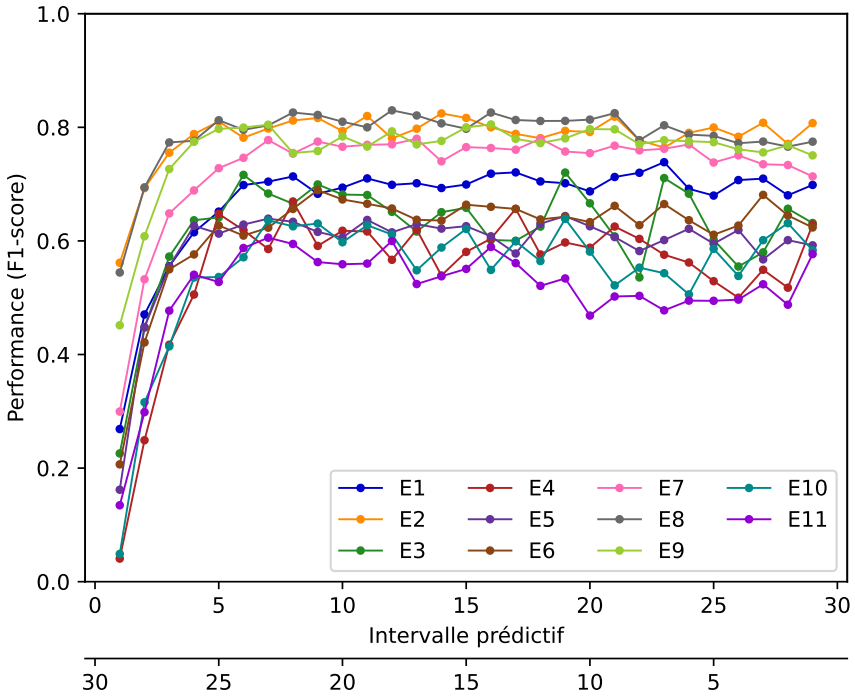

FIGURE 3.10 – Évolution des valeurs de F1-score en fonction de PI.

La figure 3.10 montre, pour chaque modèle associé à une erreur cible, la valeur du F1-score obtenue en fonction de l'intervalle PI. Jusqu'à environ 5 jours, la valeur du F1-score augmente d'autant plus que l'intervalle PI est grand. Les performances sont

donc satisfaisantes à partir d'une valeur de PI supérieure à 5. Au-delà de cette valeur, les performances ne semblent pas être dépendantes d'une valeur spécifique du paramètre PI et se stabilisent.

Il est à noter que parmi les 11 erreurs cibles, $E2$, $E8$ et $E9$ présentent les meilleures performances. Cette dernière observation rejoint la remarque effectuée dans le paragraphe 3.3.1, à propos de la sensibilité du `SVM` (tableau 3.7) vis-à-vis du déséquilibre des classes. En effet, ces meilleures performances semblent être liées à un déséquilibre des classes moins marqué (voir le tableau 3.3 pour rappel des proportions de la classe positive).

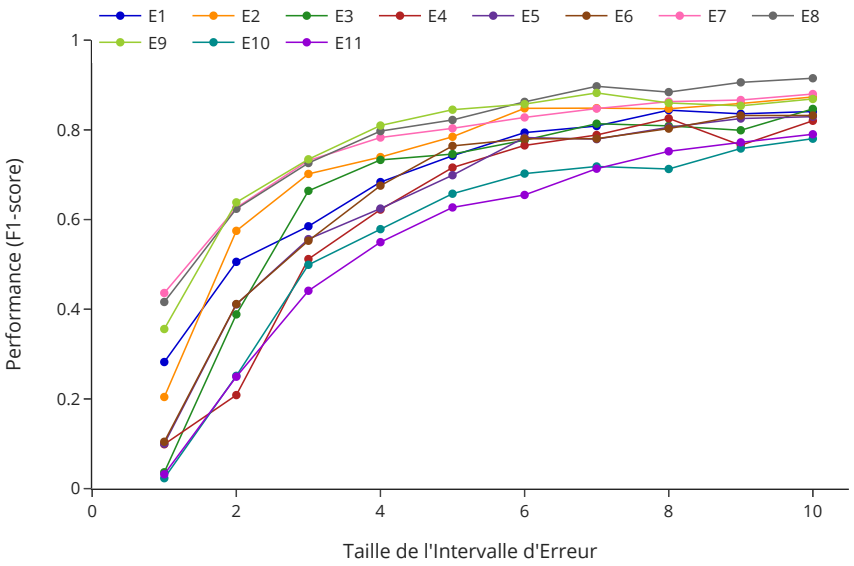

FIGURE 3.11 – Évolution du F1-score en fonction de EI.

Comme nous l'avons expliqué dans le chapitre 2, le paramètre EI introduit une flexibilité par rapport aux prédictions positives, de même qu'une rigidité pour les prédictions négatives. En effet, une prédiction positive signifie que durant un intervalle de temps dans le futur, l'erreur E_i surviendra, peu importe le nombre de fois. Au contraire, prédire une instance comme appartenant à la classe négative indique que, au cours de ce même intervalle de temps, il n'y aura pas du tout d'occurrence de cette erreur critique. Rappelons que les *bags* sont étiquetés positifs *ssi* l'erreur E_i à prédire a été observée au moins une fois au cours de EI. Par conséquent, plus la taille de cet intervalle est importante, plus il y a de chance pour que le *bag* considéré soit étiqueté positif. Intuitivement, concernant les prédictions positives, l'hypothèse serait que les prédictions sont d'autant plus correctes que la valeur du paramètre EI est élevée. En considérant les prédictions négatives, le

comportement attendu est inversé : plus EI est élevé, moins les prédictions négatives sont correctes. Pour analyser cette hypothèse, nous avons fait varier le paramètre EI de 1 à 10 jours et évalué la performance du modèle en fonction de ces valeurs pour chaque erreur cible. Pour cette expérience, les paramètres PI et RI sont tous deux fixés à 7 jours. Les résultats obtenus sont représentés sur la figure 3.11.

De manière générale, notre hypothèse initiale est confirmée : le F1-score augmente en fonction de EI : non seulement nous obtenons de meilleures prédictions pour la classe positive, mais celles de la classe négative par ailleurs ne sont pas dégradées.

Conclusion

Nous avons adressé une solution générique permettant de prédire chacune des 11 erreurs cibles d'intérêt. Les données sont très hétérogènes, du fait de notre cas d'usage industriel et des machines qui sont sollicités de différentes façons. Aussi, nous avons cherché à proposer un modèle de MP générique capable de tenir compte de cette hétérogénéité afin d'être adressé à une flotte de différentes machines. En plus du problème d'hétérogénéité, certaines erreurs cibles sont peu représentées à travers les machines, ce qui a généré un problème de déséquilibre des classes. Nous avons cherché à résoudre le problème au moyen de différentes méthodes de sous-échantillonnage. Nos analyses ont cependant démontré une dégradation de la qualité des modèles lorsque ces techniques sont appliquées.

Parmi les 5 méthodes de classification que nous avons explorées (Bayésien naïf, `AD`, `SVM`, `FA` et `DNN`), le modèle basé sur un réseau de neurones profond s'est avéré le plus performant avec une très bonne qualité de prédiction (un F1-score de 0.8).

Concernant les paramètres de structuration des données, nos analyses nous ont permis de vérifier qu'un intervalle de prédiction de taille inférieure à 5 jours était insuffisant pour prédire avec certitude la survenue d'une erreur critique. L'analyse des performances du modèle en fonction de la valeur du paramètre EI a révélé que la qualité du modèle peut être optimisée en augmentant la taille de cet intervalle, mais ceci conduit à des prédictions moins précises sur l'indication temporelle de la survenue de l'erreur critique.

Nous avons vu dans le chapitre 2 que les erreurs dans les fichiers *log*, sont émises de façon successive. Les modèles de classification que nous avons explorés dans ce chapitre ne nous permettent pas de tenir compte de la forme séquentielle de ces erreurs. Dans le chapitre suivant, nous allons présenter une autre approche de prédiction des erreurs critiques basée sur un réseau de neurones récurrents afin de prendre en compte le caractère séquentiel des fichiers *logs* émis par les machines.

Chapitre 4

Modèle de prédiction basé sur un réseau de neurones récurrents

Introduction

La méthodologie que nous avons présentée dans le chapitre 2 et appliquée dans le chapitre 3 nous a permis de prédire la survenue d'une erreur critique à partir d'un ensemble d'observations contenues dans l'intervalle PI, qui précède la survenue de l'erreur critique. Le contenu du *bag*, défini par PI est transformé en une méta-instance qui peut être vu comme un instantané de l'ensemble des observations émises durant PI jours. L'hypothèse que nous avons mis en avant dans cette méthode est que cet instantané forme un motif qui peut être interprété comme un signal fort pour la prédiction de l'erreur critique.

Dans certaines conditions, la succession des événements pourrait traduire l'évolution d'un état, et constituer un signal supplémentaire pour la prédiction. Cette succession n'a pas été prise en compte dans l'approche que nous avons appliquée jusqu'ici puisque chaque prédiction est effectuée à partir d'un instantané du *bag*. Des efforts ont été faits dans le chapitre 2, pour tenir compte de la succession des observations, puisque nous avons tenté de prédire les erreurs critiques à partir de la concaténation des instances du *bag*. Ces efforts n'ont pourtant pas apporté de résultat probant, notamment dû au fait que cela augmente considérablement le nombre de variables explicatives (ou ***features***, dans le vocabulaire de l'apprentissage profond).
Nous avons vu dans le chapitre 1 que certaines approches d'apprentissage profond appelées réseaux de neurones récurrents (RNN) fournissent une capacité de mémorisation permettant de tenir compte des observations passées pour prédire l'événement futur.

Dans ce chapitre, nous allons étendre l'approche proposée dans le chapitre 2 avec deux modèles RNN, à savoir un réseau ***Gated Recurrent Unit* (GRU)** et un réseau ***Long Short Term Memory* (LSTM)**. L'objectif de cette approche, plus sophistiquée que la méthode de concaténation, est d'assimiler le contenu d'un *bag* à une séquence afin de tenir compte de la succession des observations sans augmenter le nombre *features* dans le jeu de données.

La section 4.1 présente les deux modèles RNN que nous avons utilisés dans cette nouvelle approche, leur architecture, les hyper-paramètres ainsi que les étapes de préparation des données qui ont été nécessaires pour l'entraînement. Enfin, les résultats obtenus seront présentés dans la section 4.2 et nous discuterons de ces résultats dans la dernière section de ce chapitre.

4.1 Méthode d'apprentissage profond

4.1.1 Réseau de neurones récurrents (RNN)

Comme nous l'avons vu dans le chapitre 1, la vaste famille des réseaux de neurones compte une catégorie d'approches appelée Réseau de Neurones Récurrents (RNN) capable de prendre en compte la succession des événements au sein d'une séquence. Parmi ces méthodes, les plus couramment utilisées sont les réseaux *Long Short Term Memory* (LSTM) [HS97], et les réseaux *Gated Recurrent Unit* proposés en 2014 par Cho *et al.* [CvMG+14]. Rappelons que le LSTM et le GRU sont basés sur un principe de fonctionnement commun : le neurone est doté d'un cycle permettant de mettre à jour l'information passée pour la transmettre aux neurones suivants. La différence entre ces deux modèles relève du nombre de paramètres que le modèle doit ajuster puisque le LSTM en contient trois de plus que le GRU.

Les réseaux de neurones récurrents sont des modèles dont la tâche est de prédire le dernier événement d'une séquence. Concrètement, le dernier événement constitue la variable Y à prédire. Dans le cas d'usage présenté dans ce chapitre, la valeur de Y indique l'absence (0) ou la présence (1) de l'occurrence de l'erreur cible. L'objectif d'application est le même que celui présenté dans le chapitre 3, c'est-à-dire que nous nous sommes intéressés à 11 erreurs critiques appelées, **erreurs cibles**, et nous avons construit un modèle par erreur cible.

4.1.2 Architecture et hyper-paramètres du modèle

L'architecture du modèle GRU que nous avons conçue est constituée de 5 couches, chacune d'elles contient 3 fois plus d'unités récurrentes que de *features* dans le jeu de données (c'est-à-dire plusieurs centaines). De cette manière, la dimension des couches récurrentes sera adaptée au nombre de *features* dont les dépendances temporelles seront prises en compte par le modèle. Concernant le LSTM, l'architecture est composée d'une seule couche contenant autant d'unités récurrentes que de *features*.

Ces deux architectures ont été sélectionnées sur la base d'expérimentations multiples effectuées sur la prédiction de l'erreur $E8$ (pour laquelle nous avions obtenu les meilleurs résultats avec le modèle DNN). Au cours de ces expérimentations, nous avons notamment modifié le nombre de couches et le nombre de neurones par couche, ou encore appliqué une méthode de *dropout*. Nous avons conservé la configuration pour laquelle la performance

de prédiction est la plus importante. Ces deux modèles ont été implémentés avec la librairie `Pytorch` du langage `Python`[1].

	Modèle GRU	*Modèle LSTM*
Nombre d'*epochs*	250	1000
Taille du *batch*	64	
Optimisation	`Adadelta` [Zei12]	
learning rate	0.5	0.8
Régularisation ρ	0.99	
Régularisation ϵ	1e-6	

TABLEAU 4.1 – Hyper-paramètres associés à la construction des modèles GRU et LSTM.

Le tableau 4.1 présente les hyper-paramètres d'entraînement utilisés pour les deux modèles. Le taux d'apprentissage, ou *learning rate* en anglais, est un paramètre permettant de contrôler le pas avec lequel les poids du réseau évoluent. L'ensemble de ces hyper-paramètres a été stabilisé en fonction de la meilleure performance de prédiction obtenue pour l'erreur $E8$. Nous nous sommes également assurés que l'entraînement ne faisait pas l'objet d'un phénomène de sur- ou de sous-apprentissage en analysant les courbes de l'erreur de classification en fonction des *epochs* (l'explication sur ce phénomène a été détaillée dans le chapitre 2).

Lors de l'entraînement des deux modèles, le critère d'évaluation du coût de l'erreur de classification à chaque **epoch** correspond à la fonction ***binary cross-entropy with logit loss***, détaillée dans l'annexe A.

À présent, nous allons nous intéresser à la présentation des données au modèle.

4.1.3 Préparation des jeux de données

Pour rappel et comme illustré dans la figure 4.1, le *bag*, introduit dans le chapitre 2, est une structure qui contient l'ensemble des événements survenus avant l'erreur critique. Dans notre méthodologie, le *bag* est agrégé en une méta-instance. Si l'on ne fait pas cette transformation, les valeurs des *features* seront conservées à l'échelle de la journée, et la succession des instances dans le *bag* pourra être prise en compte pour la prédiction.

1. https://github.com/AnonymousUser49/HighErrorPrediction.git

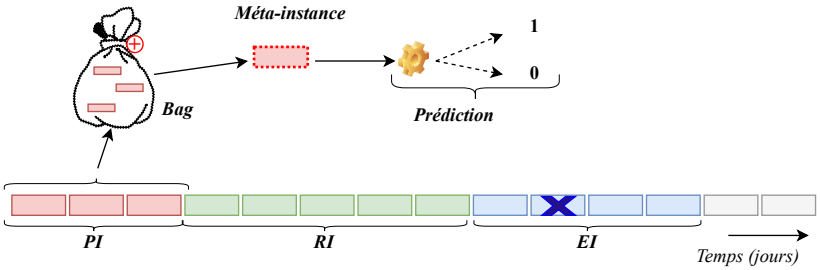

FIGURE 4.1 – Synthèse illustratrice de la méthodologie de prédiction relative au chapitre2.

À la différence de l'approche présentée dans le chapitre 2, les *bags* sont donc transmis au RNN sous leur forme native et non pas sous leur forme agrégée. Rappelons qu'une instance dans un *bag* est un vecteur de *features* dont les valeurs sont les nombres d'occurrences de chaque erreur, au cours d'une unité de temps de l'ordre de la journée. Tout comme pour les expérimentations présentées dans le chapitre 3, nous avons intégré aux *features* le nombre d'occurrences de l'erreur cible, observées par le passé, ainsi que l'identifiant des machines. Afin de présenter des résultats comparables, les paramètres de structuration des données, sont fixés par défaut avec les mêmes valeurs que celles utilisées dans le chapitre 1, à savoir : $PI = 7; RI = 7; EI = 4$. Au cours des expérimentations que nous ferons, ces valeurs pourront changer de manière ponctuelle.

L'ensemble des *bags* construits en suivant la méthode décrite dans le chapitre 2 pourront par la suite alimenter les modèles pour les entraîner. Le paragraphe suivant décrit le protocole que nous avons suivi pour la phase d'entraînement.

4.1.4 Entraînement

Toutes les expérimentations dans ce chapitre se déroulent en suivant la même procédure que celle appliquée dans le chapitre 3. Nous avons appliqué une validation croisée en 10 itérations, avec un échantillonnage des ensembles d'entraînement et de test respectant un ratio respectif de 80 % / 20 %.
En apprentissage profond, l'entraînement s'effectue à chaque *epoch* sur un ensemble d'entraînement et un ensemble de validation. Aussi, 20 % des données du jeu d'entraînement sont sélectionnés aléatoirement à chaque *epoch* pour constituer le jeu de validation. Finalement, sur l'ensemble des données que nous avions au départ, 72 %, 18 % et 10 % sont respectivement dédiés aux ensembles d'entraînement, de validation et de test.

La section suivante présente l'application de deux types de modèles de prédiction basés sur un réseau de neurones récurrents : le GRU et le LSTM. Chaque expérimentation a été effectuée sur un processeur Xeon(R)-E-2146G 3.50Ghz.

4.2 Expérimentations et résultats

Pour commencer, nous allons comparer les performances obtenues pour les deux modèles RNN (un réseau GRU et un réseau LSTM), aux performances des modèles que nous avons mis au point dans le chapitre 3. Pour rappel, nous avions obtenu au mieux un **F1-score de 0.8** sur les modèles basés sur un DNN. Aussi, nous avons réalisé les expériences de ce chapitre dans l'hypothèse que considérer le caractère séquentiel des instances nous permettrait d'obtenir les mêmes résultats en termes de F1-score, voire meilleurs dans l'idéal.

4.2.1 Comparaison à la méthode précédente

Le tableau 4.2 présente les résultats obtenus par le modèle DNN, le modèle GRU et le modèle LSTM.
Pour faciliter la comparaison avec les résultats du chapitre 3, pour l'ensemble de ces résultats, aucune correction du déséquilibre des classes n'a été appliquée et nous avons gardé le même jeu de paramètres de structuration des données, à savoir $PI = 7$, $RI = 7$, $EI = 4$.

	Approche méta-instance	Approche RNN	
Erreur cible	*DNN*	*GRU*	*LSTM*
E1	**0.688** ± 0.03	0.645 ± 0.05	0.648 ± 0.08
E2	0.731 ± 0.02	0.769 ± 0.07	**0.779** ± 0.06
E3	**0.737** ± 0.07	0.660 ± 0.11	0.659 ± 0.11
E4	0.616 ± 0.10	0.723 ± 0.15	**0.768** ± 0.15
E5	**0.637** ± 0.03	0.564 ± 0.08	0.606 ± 0.06
E6	**0.678** ± 0.05	0.668 ± 0.05	0.662 ± 0.10
E7	**0.783** ± 0.01	0.748 ± 0.04	0.745 ± 0.04
E8	0.795 ± 0.04	**0.833** ± 0.03	0.829 ± 0.04
E9	0.804 ± 0.03	0.822 ± 0.07	**0.831** ± 0.06
E10	**0.591** ± 0.12	0.577 ± 0.10	0.578 ± 0.11
E11	**0.552** ± 0.04	0.492 ± 0.05	0.470 ± 0.09
Moyenne	0.69 ± 0.08	0.68 ± 0.09	0.69 ± 0.09

TABLEAU 4.2 – Performances (F1-score) des méthodes GRU et LSTM comparées au modèle DNN.

Globalement, **il n'y a pas de gagnant absolu** parmi ces trois modèles de prédiction. Au vu des chiffres, le DNN reste le meilleur candidat, mais les différences de F1-score entre un DNN, un GRU ou un LSTM ne sont pas suffisamment significatives pour affirmer cette observation.

Concernant les deux approches de RNN que nous avons appliquées, les résultats sont très similaires. La différence la plus élevée est obtenue pour les erreurs $E4$ et $E5$, avec la méthode LSTM, mais la différence avec la méthode GRU pour ces deux erreurs s'élève seulement à 0.04.

Ces résultats montrent que l'approche RNN reste tout de même un bon candidat comme solution de prédiction et les chiffres pourraient être encore améliorés en corrigeant le déséquilibre des classes ou en réajustant les paramètres PI, RI et EI, comme nous l'avons fait dans le chapitre 3.
Comme nous avons pu le noter, il n'existe pas de meilleur candidat entre un modèle LSTM et un modèle GRU, mais le LSTM nécessite un plus grand nombre d'*epochs* (1000) pour converger comparé au GRU, et peut donc se révéler beaucoup plus gourmand en temps et en capacité de calcul. Aussi, nous avons choisi de poursuivre nos expérimentations avec le modèle GRU. Pour améliorer la qualité de prédiction, la correction du déséquilibre des classes est la première piste que nous allons étudier.

4.2.2 Correction du déséquilibre des classes

Dans le chapitre 3, lors de la préparation des données, nous avons fait le choix de retirer les séquences de fichiers *log* dans lesquelles l'erreur cible n'a jamais été observée. Le paragraphe suivant nous permettra de discuter de ce choix et des améliorations que nous avons apportées lors de l'application de ce nouveau modèle.

4.2.2.1 Suppression des séquences dépourvues de l'erreur cible

Le retrait des séquences dépourvues de l'erreur cible nous a permis de réduire une grande partie des *bags* négatifs en amont de la préparation de l'ensemble d'entraînement. Cette façon de faire constitue, de fait, une forme de correction du déséquilibre des classes. Cependant, le fait de retirer ces séquences avant de partitionner les données en un ensemble d'entraînement et un ensemble de test pourrait nous empêcher d'évaluer la performance du modèle de prédiction dans le cas où les séquences traitées appartiennent à des machines "saines", c'est-à-dire qui n'émettent pas l'erreur cible. Théoriquement parlant, nous ne pourrons pas complètement évaluer le nombre de Faux Positifs que le modèle pourrait générer, car une partie des séquences sans erreur cible n'est pas traitée par le modèle, donc le nombre de FP réel pourrait être plus élevé que celui qui est constaté.
Afin de vérifier cet effet, nous avons modifié le protocole de préparation en ne retirant ces séquences **que dans l'ensemble d'entraînement**, puis comparé les résultats obtenus pour chacun de ces deux protocoles.

Pour clarifier, les deux protocoles que nous allons évaluer par l'application d'un GRU sont :

— Ancien protocole : Les séquences dépourvues de la survenue de l'erreur cible sont retirées du jeu de données d'entraînement et du jeu de données de test.

— Nouveau protocole : Les séquences dépourvues de la survenue de l'erreur cible sont retirées du jeu d'entraînement uniquement.

Sur ces deux modes de préparation, le même modèle GRU est appliqué, pour chaque erreur cible puis évalué sur l'ensemble de test. Ces deux protocoles sont illustrés dans la figure 4.2

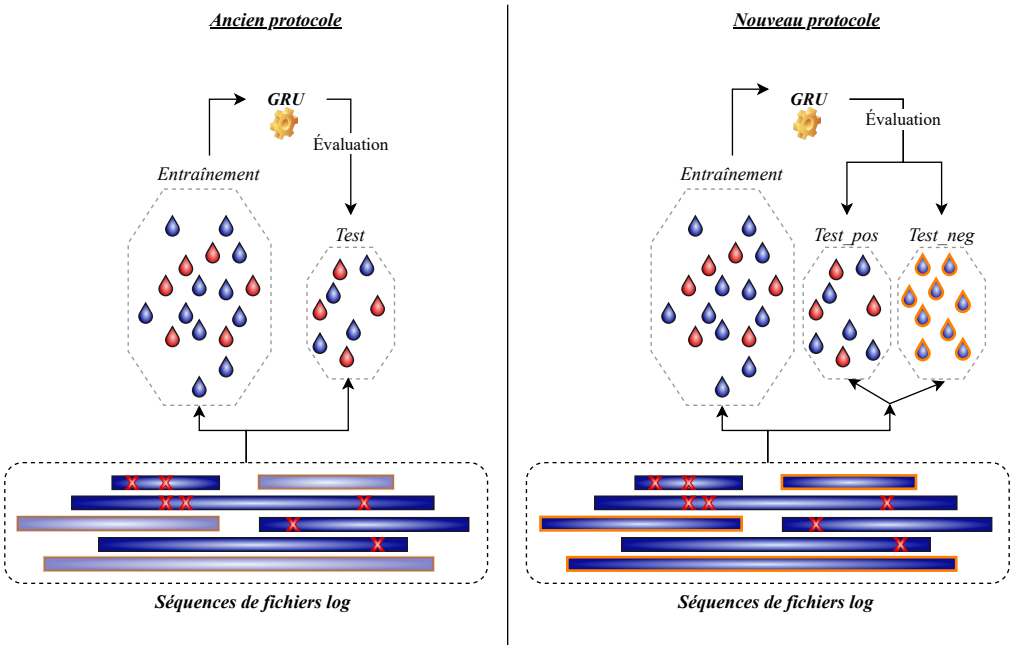

FIGURE 4.2 – Illustration des deux protocoles d'entraînement du modèle de prédiction.

Pour présenter les résultats dans le cas du nouveau protocole, nous avons choisi de scinder le jeu de test associé à chaque erreur cible en deux parties afin de mieux évaluer la performance du modèle sur les machines "saines" :

— Le jeu TEST_{NEG} qui correspond au jeu de test issu des séquences dans lesquelles l'erreur cible n'a jamais été observée. Comme nous l'avons expliqué dans le chapitre 3, ces séquences n'ont permis de générer que des *bags* négatifs.

— Le jeu TEST_{POS} qui correspond au jeu de test issu des séquences dans lesquelles l'erreur cible a été observée au moins une fois. Cet ensemble contient donc forcément des *bags* positifs.

À titre d'information, la figure 4.3, donne une idée des effectifs des *bags* pour chaque classe, dans l'ensemble TEST_{POS}, ainsi que l'effectif des *bags* négatifs dans le jeu TEST_{NEG} en fonction de l'erreur cible considérée.
Le nombre de *bags* positifs dans l'ensemble TEST_{POS} peut être lu sur l'axe de gauche. Au minimum, l'effectif des *bags* positifs est de 10 pour l'erreur $E4$, ce qui représente une très faible valeur pour la phase d'évaluation et confirme la nécessité d'appuyer l'analyse des performances du modèle à l'aide d'une matrice de confusion. Nous pouvons également noter la grande taille des jeux de test TEST_{NEG}, particulièrement pour les erreurs $E2$, $E3$ et $E4$. Ainsi, pour la suite de nos analyses, séparer les évaluations du modèle sur les ensembles TEST_{POS} et TEST_{NEG} nous permettra de mieux évaluer la qualité de la prédiction.

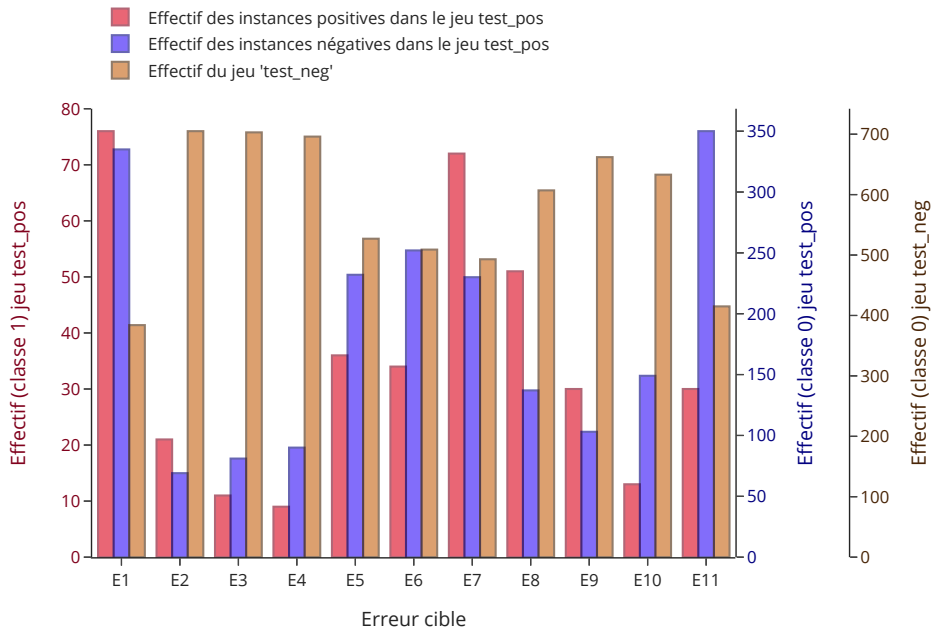

FIGURE 4.3 – Effectifs de chaque classe dans le jeu de test TEST_{POS} et effectif du jeu TEST_{NEG}.

La mesure du F1-score est utilisée pour évaluer l'efficacité du modèle sur le jeu TEST_{POS}, comme dans toutes les expériences précédentes. En ce qui concerne le jeu TEST_{NEG}, étant donné qu'il ne contient que des *bags* négatifs, l'évaluation du modèle sera

effectuée à l'aide de la mesure de rappel pour la classe négative [2]. La figure 4.4 présente les performances du modèle, évaluées sur les deux jeux de test présentés ci-dessus.

De manière générale, nous pouvons voir que les scores de rappel (en bleu sur l'axe de droite) de la classe négative sur l'ensemble TEST_{NEG} sont tous au-dessus de 0.9. Cela nous permet donc de nous rassurer sur le fait que **le modèle crée peu de Faux Positifs**. En parallèle, les performances sur l'ensemble TEST_{POS} varient d'une erreur critique à l'autre. Le meilleur modèle atteint une valeur de f1-score de 0.8. Pour les modèles aux performances les plus faibles, le score est à peine plus élevé que 0.5. Cette mauvaise performance pourrait être améliorée par une technique de sous-échantillonnage.

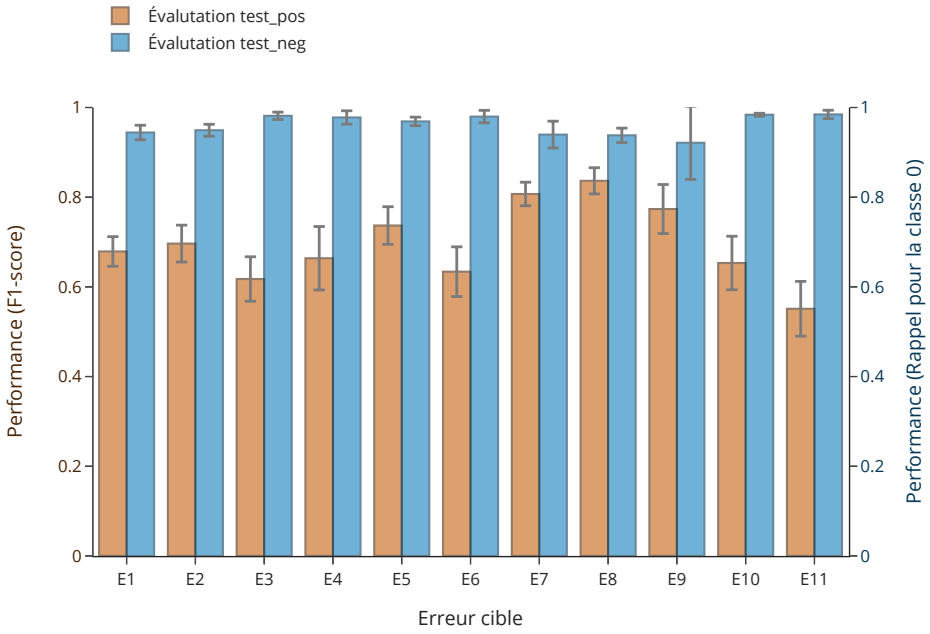

FIGURE 4.4 – Comparaison des performances des modèles en fonction de l'erreur cible à prédire.

4.2.2.2 Sous échantillonnage adaptatif

Les expériences de correction du déséquilibre des classes réalisées dans le chapitre 3 ont démontré l'inefficacité du sous-échantillonnage et du sous-échantillonnage adaptatif sur les performances du modèle. Cependant, pensant que tous les modèles d'apprentissage ne se comportent pas de la même manière face au déséquilibre des classes, nous avons

2. Rappel (classe 0) = $\frac{VN}{VN+FP}$

réitéré ces expériences de sous-échantillonnage en combinaison du modèle GRU, pour tenter d'améliorer encore la qualité de prédiction.

Au vu de la variabilité du déséquilibre observée et pour rendre les résultats comparables d'une erreur cible à l'autre, nous avons fait le choix de n'appliquer que la stratégie de sous-échantillonnage adaptatif présentée dans le chapitre 3. Pour rappel, sachant le paramètre r', une certaine proportion r de *bags* négatifs est retirée de l'ensemble d'entraînement, de manière à atteindre une proportion finale r'. Par exemple, lorsque $r' = 0.4$, si le jeu de données initial contient 60 % de *bags* négatifs et 40 % de *bags* positifs, alors environ 40 % des instances négatives seront aléatoirement retirées du jeu de données.
Afin de déterminer une valeur idéale r' de sous-échantillonnage, nous avons fait varier la valeur du paramètre r' de 0.2 à 1 puis analysé les résultats obtenus, en termes de F1-score.
Notons que lorsque le paramètre r' choisi nécessite de retirer des *bags* positifs pour respecter la proportion finale, aucun sous-échantillonnage n'est appliqué. Par exemple, si $r' = 1$ cela signifie que l'on souhaite avoir une proportion finale de 100 % de *bags* négatifs, ce qui suppose de retirer tous les *bags* positifs. En effet, le but étant de corriger la trop grande influence de la classe négative, il serait contre-productif d'appliquer un sous-échantillonnage sur les *bags* positifs.
Enfin, à titre de référence, nous pouvons noter également que lorsque $r' = 0.5$ cela signifie que le jeu de données résultant sera parfaitement équilibré.

La figure 4.5 représente les résultats obtenus sur le jeu de test TEST_{POS} pour les différentes valeurs de sous-échantillonnage adaptatif SSA. Globalement, et tout comme pour les résultats du DNN vus dans le chapitre 3, l'application du sous-échantillonnage adaptatif ne semble apporter aucun bénéfice à la performance du modèle GRU. Les résultats démontrent même une dégradation des résultats, proportionnelle au volume de *bags* négatifs retirés du jeu de données. Pour l'erreur $E4$, le modèle se comporte un peu différemment, avec une légère amélioration des résultats lorsque la proportion de *bags* négatifs autorisée dans le jeu de données est autour de 80 %. Cette erreur est associée à l'un des jeux de données les plus déséquilibrés de notre cas d'usage. Aussi, nous pouvons émettre l'hypothèse que la survenue de cette erreur critique est mieux détectée par le modèle lorsque le déséquilibre des classes est corrigé, à condition toutefois que cette correction soit faible.

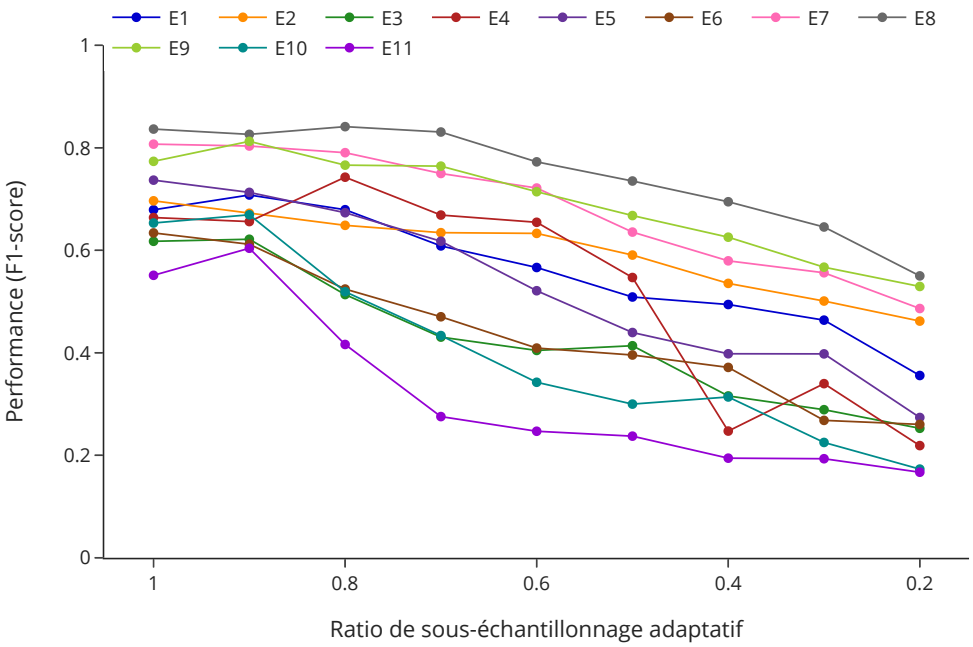

FIGURE 4.5 – Comparaison des performances de chaque modèle associé à l'erreur cible E_i, en fonction du rapport de sous-échantillonnage adaptatif. Évaluation effectuée sur l'ensemble TEST_{POS}.

Pour compléter ces résultats, la figure 4.6 présente les performances obtenues sur le jeu de test TEST_{NEG}. Les scores de performance sont obtenus avec la mesure de rappel pour la classe négative. Étant donné que ce jeu de test ne contient que des éléments négatifs, un score de rappel élevé démontre un faible nombre de faux positifs, contrairement à un score de rappel faible. Globalement, les résultats de la figure 4.6 confirment les observations précédentes : plus la proportion de sous-échantillonnage de la classe négative est importante, plus le nombre de faux positifs augmente, ce qui se traduit par un score de rappel de plus en plus faible.

La représentativité des erreurs cibles dans le jeu de données diffère d'une erreur à l'autre, ce qui se traduit par un déséquilibre de classe variable. La correction de ce déséquilibre, bien qu'il s'attache à réguler les variations d'une erreur cible à l'autre, a révélé, sauf cas particulier (erreur $E4$), une dégradation des performances du modèle.

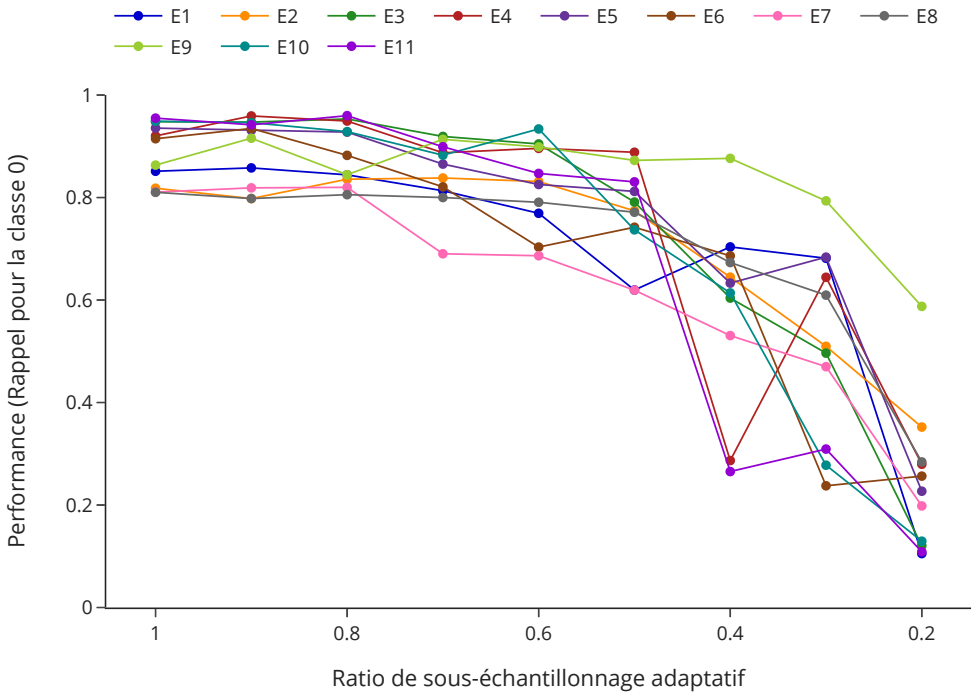

FIGURE 4.6 – Comparaison des performances de chaque modèle associé à l'erreur cible E_i, en fonction du rapport de sous-échantillonnage adaptatif. Évaluation effectuée sur l'ensemble test issu des séquences de données dépourvues de l'occurrence de l'erreur cible.

Que ce soit pour prédire la survenue d'une erreur critique ou prédire la non survenue de cette erreur, en écho à nos observations du chapitre 3, la correction du déséquilibre des classes dégrade les résultats de la méthode GRU. Nous en concluons donc que, dans notre cas d'usage, cette technique ne doit pas être appliquée pour un modèle de prédiction basé sur un réseau de neurones.
À présent, nous allons nous intéresser aux paramètres de structuration des données, tout comme nous l'avons fait au cours du chapitre 3.

4.2.3 Influence des paramètres de structuration des donnés

Dans le chapitre 3, nous avons discuté de l'importance du choix de PI afin de circonscrire la taille idéale des motifs prédictifs (assimilés aux *bags*). Cette nécessité est d'autant plus vraie avec l'application d'une approche de type RNN, qui tient compte de la succession des observations.

4.2.3.1 Influence du paramètre PI

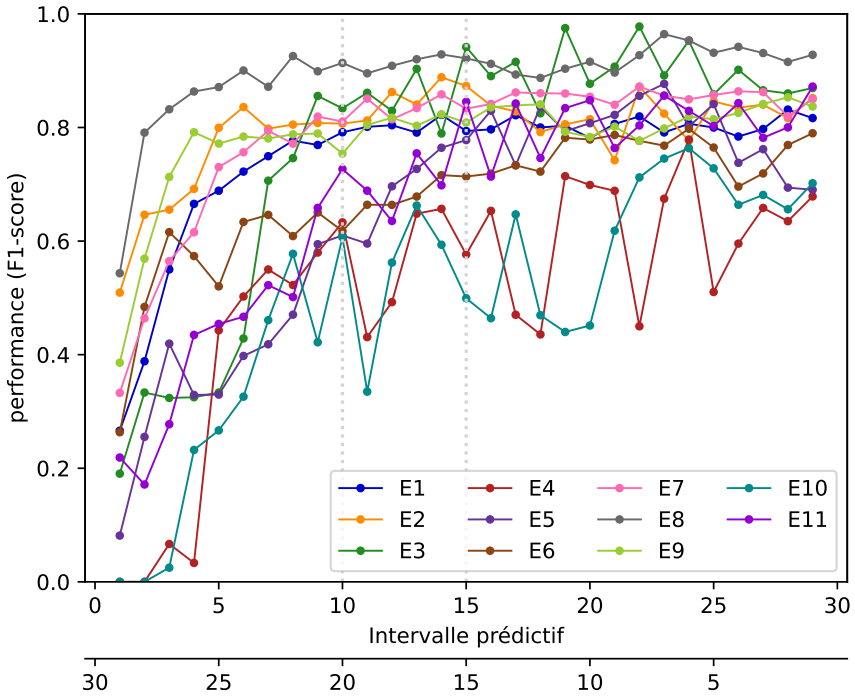

FIGURE 4.7 – Comparaison des performances de chaque modèle associé à l'erreur cible E_i, en fonction de la valeur des paramètres PI et RI. Évaluation effectuée sur l'ensemble TEST_{POS}.

Étant donné le caractère hétérogène de notre cas d'usage, la taille de l'intervalle prédictif idéale pour obtenir les meilleurs F1-score devrait être différente selon les erreurs cibles. Afin de mettre ce phénomène en exergue, nous avons fait varier les valeurs des paramètres PI et RI de 1 à 29 jours au sein d'un intervalle fixe de 30 jours, sachant que $RI = 30 - PI$.

Sur la figure 4.7, les performances sur l'ensemble TEST_{POS} sont globalement croissantes lorsque la taille de l'intervalle PI augmente jusqu'à un certain point cependant. En effet, pour toutes les erreurs cibles d'intérêt, les courbes atteignent un plafond de performance, au-delà duquel l'augmentation de la taille de l'intervalle prédictif ne semble pas avoir d'effet (ni positif ni négatif). Ce plafond est atteint avec une valeur PI qui diffère selon les erreurs cibles.

L'augmentation de la taille de l'intervalle PI est concomitante à la réduction de la taille de l'intervalle RI. Aussi, nous avons émis l'hypothèse que les valeurs de performance situées au début de la phase stationnaire pourraient être représentatives d'un jeu de paramètres PI et RI optimal pour prédire l'erreur cible considérée.

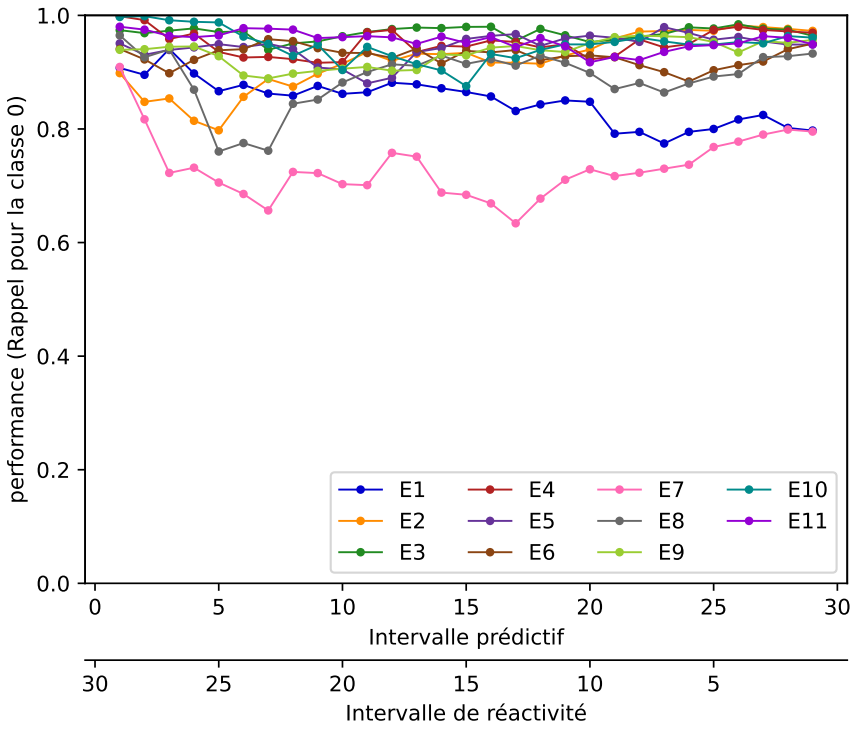

FIGURE 4.8 – Comparaison des performances de chaque modèle associé à l'erreur critique E_i, en fonction de la valeur des paramètres PI et RI. Évaluation effectuée sur l'ensemble TEST_{NEG}.

Sur cette hypothèse, les courbes de la figure 4.7 semblent présenter deux groupes distincts. Le premier groupe de modèles, associé aux erreurs E_1, E_2, E_3, E_7, E_8 et E_9, semble fournir les meilleures performances lorsque la taille de l'intervalle PI est proche de 10 jours. Le second groupe de modèles (E_4, E_5, E_6, E_{10} et E_{11}) semble atteindre une forme stationnaire autour de $PI = 15$.
Enfin, les modèles associés aux erreurs $E4$ et $E10$ démontrent des écarts de F1-score en fonction de PI qui semblent plus importants que pour les autres erreurs cibles. Ceci peut être expliqué par le fait que les jeux de données associés à ces modèles font partie de ceux qui présentent les plus faibles effectifs associés à un déséquilibre de classes important (voir les statistiques de la figure 4.3). Cependant, les caractéristiques de ces jeux de données ne sont pas l'unique explication des mauvaises performances parfois observées et de l'instabilité du modèle. En effet, l'erreur $E3$ dont les modèles de prédiction présentent de très bons résultats est pourtant associée à des jeux de données qui présentent les mêmes caractéristiques statistiques que pour l'erreur $E4$.

Comme nous l'avons relevé plusieurs fois au cours de ce manuscrit, le problème de MP que nous cherchons à traiter s'applique sur des données très hétérogènes, c'est ce qui rend les résultats si complexes et les hyper-paramètres si difficiles à déterminer. Les courbes de variation du F1-score en fonction de la valeur de PI nous ont permis de distinguer deux profils d'erreur cible, chacun associé à une valeur minimale du paramètre PI. Cette observation n'avait pas été faite dans le chapitre 3. Cela démontre simplement que la méthode de prédiction basée sur des séquences de *logs* est plus sensible à la taille des séquences, que le DNN ne l'était sur la taille des *bags* agrégés.

Pour aller un peu plus loin dans l'analyse de ces groupes, la section suivante permet d'évaluer l'effet du paramètre RI pour chacun des deux profils.

4.2.3.2 Influence du paramètre RI

La performance du modèle prédictif pour une erreur cible donnée dépend entre autres de la spécificité du motif prédictif qui la précède : à quel point le motif formé par une séquence d'erreurs est associé à une erreur cible particulière. La spécificité du motif peut être ajustée grâce au paramètre PI que nous avons étudié dans la partie précédente. D'autre part, un autre élément important à ajuster correspond au délai d'anticipation. Les analyses effectuées avec la méthode du chapitre 3 ont révélé que ce délai devait être fixé avec une valeur minimale de 5 jours. Cependant, ces analyses ne nous ont pas permis de définir une limite à ce délai. D'un point de vue MP, il pourrait être intéressant de fournir une solution capable de prédire avec un délai plus long qu'une semaine, avec la même qualité de résultat. Pour chaque profil d'erreur cible que nous avons distingué lors des analyses précédentes, nous avons fait varier le paramètre RI avec la valeur de PI idéale pour ce profil. Concrètement, pour le groupe 1 (erreurs E_1, E_2, E_3, E_7, E_8 et E_9), $PI = 10$ et pour le groupe 2 (E_4, E_5, E_6, E_{10}, E_{11}), $PI = 15$.
Les performances en fonction du paramètre RI sont représentées par le graphique de la (figure 4.9) pour le profil 1, et celui de la figure 4.10 pour le profil 2. Tout d'abord, nous pouvons voir que sur les deux groupes de résultats obtenus, les erreurs $E4$ (figure 4.10) et $E3$ (figure 4.9) présentent des écarts de performance d'autant plus importants que la valeur de RI est grande. La stabilité qui est observée au contraire sur les autres erreurs, quelle que soit la taille de RI, pourrait être expliquée par la présence régulière de *bags* positifs dans le jeu de test, associée la présence régulière de motifs séquentiels d'erreurs suffisamment distincts et homogènes pour être détectés par le modèle.

Finalement, si le but recherché est d'avoir un résultat de qualité, en ce qui concerne les erreurs $E4$ et $E3$ le paramètre RI **doit être inférieur ou égal à 7 pour** $E4$ et **inférieur ou égal à 10 pour** $E3$. Au-delà de ces valeurs, la stabilité du modèle ne

permet pas d'assurer un résultat fiable 100 % du temps. Pour toutes les autres erreurs cibles, ce paramètre n'a pas d'influence sur la performance, donc le délai d'anticipation peut être largement allongé.

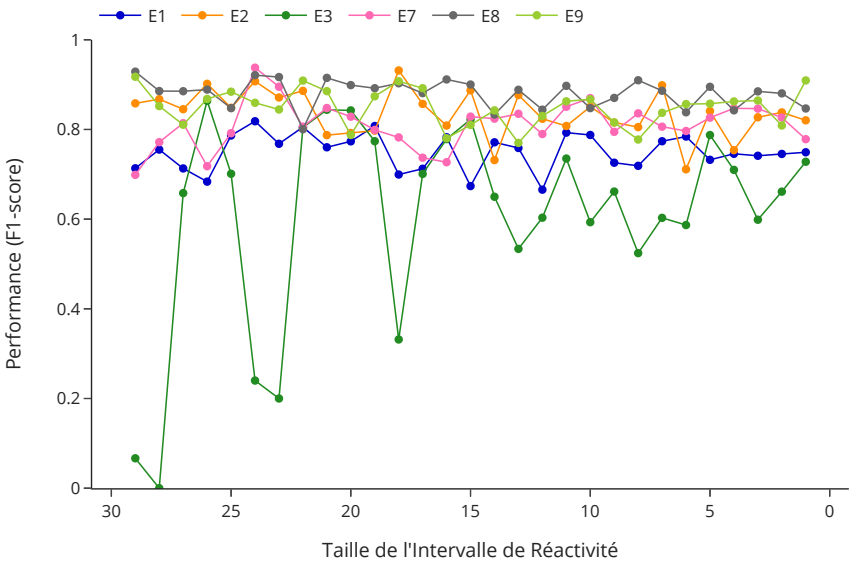

FIGURE 4.9 – Comparaison des performances des modèles du groupe 1, en fonction de la valeur RI lorsque $PI = 10$. Évaluation effectuée sur l'ensemble TEST_{POS}.

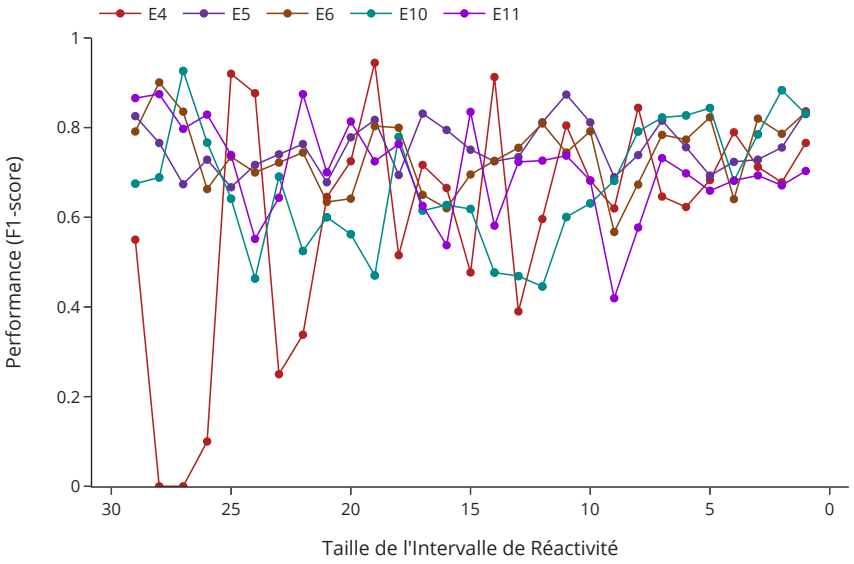

FIGURE 4.10 – Comparaison des performances de chaque modèle associé à l'erreur critique E_i, en fonction de la valeur RI lorsque $PI = 15$. Évaluation effectuée sur l'ensemble TEST_{POS}.

Jusqu'à présent, nous avons cherché à optimiser les paramètres associés à la structuration des données, nous allons à présent discuter des choix de conception du modèle qui ont été effectués pour mettre en œuvre cette solution de MP.

4.3 Discussion

Rappelons que l'architecture et les hyper-paramètres du modèle GRU (présentés dans le tableau 4.1) ont été ajustés en fonction des performances de prédiction de l'erreur $E8$. Premièrement, nous allons discuter de l'effet de chaque élément que nous avons ajusté pour obtenir les configurations finales du modèle.

4.3.1 Architecture du GRU et hyper-paramètres

La diminution du *learning rate*, de 0.5 à $1e-3$ a entraîné une dégradation des performances liées à la classe positive sans impact sur la classe négative. Au contraire, augmenter le *learning rate* dégrade les performances dans les deux classes. Lorsque l'on force le modèle à s'attarder sur l'information (avec un *learning rate* faible, le modèle a tendance à se concentrer sur le signal des *bags* négatifs).

D'un certain point de vue, la taille du *batch* pourrait s'avérer insuffisante. En effet, pour chaque *epoch* seuls 64 exemples sont sélectionnés aléatoirement pour un nombre de *features* assez élevé (plusieurs centaines).
De manière similaire à la diminution du *learning rate*, nous avons augmenté la taille du *batch* à 128, mais cela a conduit à une dégradation de la performance en faveur de la classe négative. Le phénomène restait cependant assez peu marqué : plus de 50 % des *bags* positifs étaient encore détectés par le modèle de prédiction. Du point de vue de l'évolution de l'apprentissage, la courbe d'erreur de classement en fonction du nombre d'*epochs* était profondément marquée par des écarts de valeurs importants. Nous en avons déduit que **la taille du *batch* doit rester faible afin de réduire l'effet provoqué par l'hétérogénéité des données**.

Augmenter le paramètre de stabilisation numérique ($\epsilon = 1e-3$ au lieu de $1e-6$) a provoqué un ralentissement important de l'apprentissage menant à une convergence difficile du modèle. Côté performance, nous avons obtenu une dégradation nette du nombre de VP, ce qui suppose que l'apprentissage s'effectue en faveur de la classe négative, encore une fois. Un paramètre *epsilon* trop grand a tendance à effacer l'information qui pourrait aider le modèle à discriminer les *bags* positifs.

En termes d'optimisation, nous avons choisi d'utiliser l'algorithme `Adadelta` [Zei12] en comparaison avec `Adam` [KB14]. En effet, `Adadelta` est reconnu pour sa capacité à

réadapter le *learning rate* à chaque *epoch* afin de trouver dynamiquement le minimum local. Il convient donc mieux pour des petits jeux de données hétérogènes tels que ceux que nous avons manipulés dans les travaux de cette thèse.

Les diverses expérimentations que nous avons mises en œuvre nous ont permis de stabiliser un modèle capable de considérer les *bags* comme des séquences. Les résultats que nous avons obtenus, ne surpassent pas dans l'absolu ceux obtenus avec l'approche du chapitre 3. Une approche différente qui pourrait être expérimentée dans le futur pour améliorer ces résultats, est de construire une architecture combinant à la fois l'aspect séquence et l'aspect méta-instance des *bags*. Des travaux similaires à cette idée ont été proposés par Wang *et.al* [WCN+21]. L'architecture combine un modèle global et un modèle local permettant de considérer les motifs séquentiels à deux échelles de granularité différentes.

4.3.2 Retrait des séquences dépourvues de l'erreur cible

Nous avons fait le choix, lors de nos analyses du chapitre 3, de retirer les séquences dépourvues de la survenue de l'erreur cible pour réduire le nombre de *bags* négatifs dans le jeu de données. Ce choix a été effectué, car les algorithmes de classification appliqués initialement (`SVM`, Bayésien, `AD` et `RF`) sont sensibles au déséquilibre des classes et notre objectif est avant tout de classer correctement les *bags* positifs. Dans la continuité de ces analyses, nous avons également retiré ces séquences avant l'application du modèle GRU.

Nous souhaiterions à présent évaluer la capacité du GRU à prédire même lorsque ces séquences, qui apportent un grand volume de *bags* négatifs, sont conservées dans le jeu de données. L'évaluation du modèle dans les deux cas a été effectuée sur l'ensemble TEST_{POS}. Les chiffres sont présentés dans le diagramme en bâtons de la figure 4.11. En majorité, nous pouvons voir que lorsque les séquences dépourvues de l'erreur cible sont retirées du jeu d'entraînement, les performances du modèle se dégradent par rapport au cas pour lequel ces séquences sont conservées. Cette observation nous permet de conclure que le modèle RNN semble sensible au contenu des séquences négatives, et nécessite donc que celles-ci soient conservées dans les jeux de données, pour une meilleure capacité de prédiction.

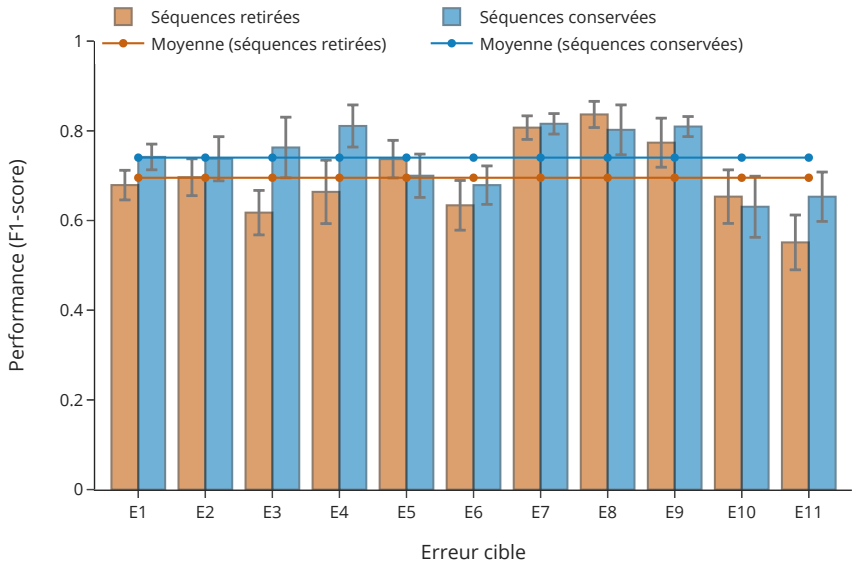

FIGURE 4.11 – Comparaison des performances des modèles lorsque les séquences dépourvues de l'erreur cible sont retirées, ou non, du jeu de données d'entraînement. Évaluation sur le jeu TEST_{POS}.

Dans le chapitre 3, nous avons soulevé la question de l'hétérogénéité, notamment due au fait que notre cas d'usage s'adresse à différentes machines de l'industrie. Pour tenir compte de ce problème, nous avons ajouté l'identifiant de la machine dans les variables explicatives. Le paragraphe suivant nous permet de discuter de choix.

4.3.3 Influence de l'identifiant de la machine

Comme nous venons de l'évoquer, nous pensons que rajouter l'identifiant de la machine dans les *features* du modèle pouvait aider celui-ci à mieux prédire la survenue des erreurs cibles. Pour vérifier notre hypothèse, nous avons comparé les performances des modèles avec et sans l'identifiant de la machine dans les *features*. Notons que le partage des données en un ensemble d'entraînement et un ensemble de test, s'effectue aléatoirement sur les *bags*, en veillant à maintenir dans les deux ensembles une proportion similaire de *bags* positifs. Aussi, nous ne contrôlons pas le fait que certains *bags* de l'ensemble d'entraînement soit issu d'une machine dont on ne retrouve aucun *bag* dans l'ensemble de test. Les résultats, obtenus à partir de l'ensemble de test TEST_{POS} sont présentés dans la figure 4.12. Le score de rappel a également été évalué sur l'ensemble TEST_{NEG} dans les mêmes conditions de comparaison (figure 4.13).

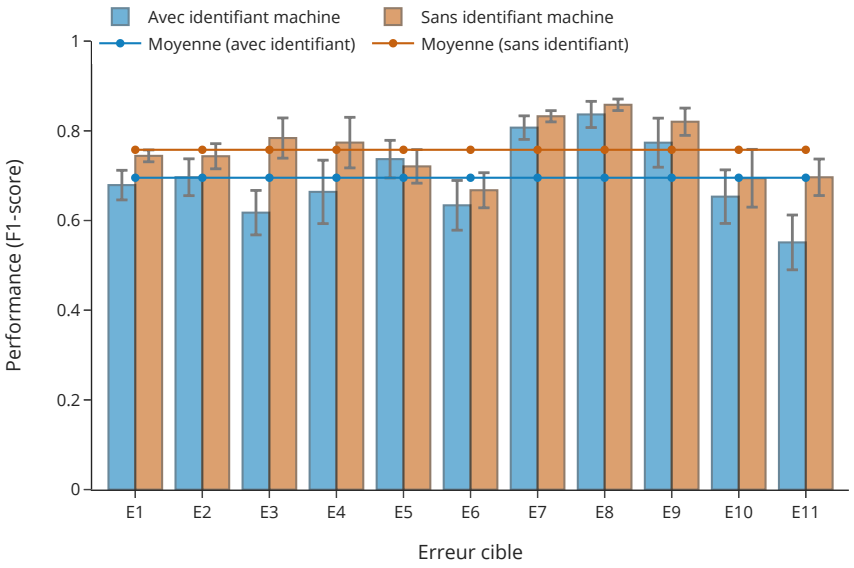

FIGURE 4.12 – Comparaison des performances du modèle sur le jeu TEST_{POS} lorsque l'identifiant de la machine est ajouté ou non en tant que *feature*.

Globalement, nous pouvons voir que notre hypothèse n'est pas vérifiée. Pour la plupart des erreurs cibles, le score de performance semble sensiblement plus élevé lorsque l'identifiant de la machine ne fait pas partie des *features*. Pour le cas de l'erreur $E3$, le modèle perd 0.2 de F1-score entre le cas où il est entraîné avec des attributs contenant l'identifiant de la machine et celui où il est entraîné sans cet identifiant.
Une explication possible à ce phénomène est que le modèle GRU, lors de l'apprentissage, tient compte des fluctuations des *features* au cours du temps. L'identifiant de la machine n'étant pas une variable évolutive, le rajouter en tant que variable prédictive pourrait créer du bruit. Par ailleurs, ces résultats pourraient indiquer également que pour la plupart des erreurs cibles, les motifs prédictifs ne diffèrent pas trop d'une machine à l'autre.

Sur le jeu TEST_{NEG}, les scores en termes de rappel de la classe négative ne démontrent pas de tendance à améliorer ou dégrader les résultats. Globalement, l'ajout ou non de l'identifiant de la machine en tant que *feature* ne semble pas avoir d'effet sur la prédiction de la classe négative.

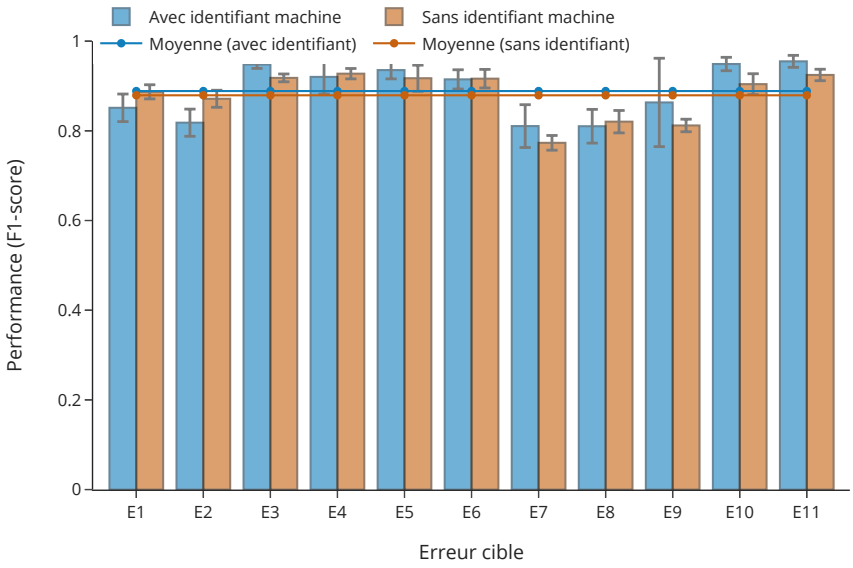

FIGURE 4.13 – Comparaison des performances du modèle sur le jeu TEST_{NEG} lorsque l'identifiant de la machine est ajouté ou non en tant que *feature*.

4.3.4 Étude de corrélation entre les caractéristiques statistiques et les performances du modèle

Les écarts de performances observés en fonction des 11 erreurs cibles à prédire pourraient s'expliquer par les caractéristiques statistiques des jeux de données ayant servi à entraîner le modèle. Parmi ces caractéristiques, ont été sélectionnées :

— la taille du jeu de données
— le nombre de machines concernées par l'erreur cible à prédire
— le ratio de *bags* positifs avant ré-équilibrage des classes par sous-échantillonnage
— le nombre de *features*

Sur la figure 4.14, chaque graphique représente le nuage de points entre deux caractéristiques statistiques des jeux de données ou entre une caractéristique des jeux de données et les F1-score moyens. Une diagonale a été tracée en pointillés sur chacun. Lorsque le nuage de points s'aligne en suivant la diagonale du graphique, cela démontre l'existence d'une corrélation proportionnelle entre les deux variables. Chaque graphique est annoté par le coefficient de corrélation de Pearson, donné par la formule suivante :

$$R_{X,Y} = \frac{COV(X,Y)}{\sigma_X \times \sigma_Y} \tag{4.1}$$

Avec le terme $COV(X,Y)$ défini par l'équation suivante :

$$COV(X,Y) = E(X,Y) - E(X) \times E(Y) \tag{4.2}$$

Lorsque le coefficient de Pearson est proche de 0, il n'y a pas de corrélation, si le coefficient est proche de 1 il existe une corrélation proportionnelle positive et lorsqu'il est proche de -1, il existe une corrélation proportionnelle négative.

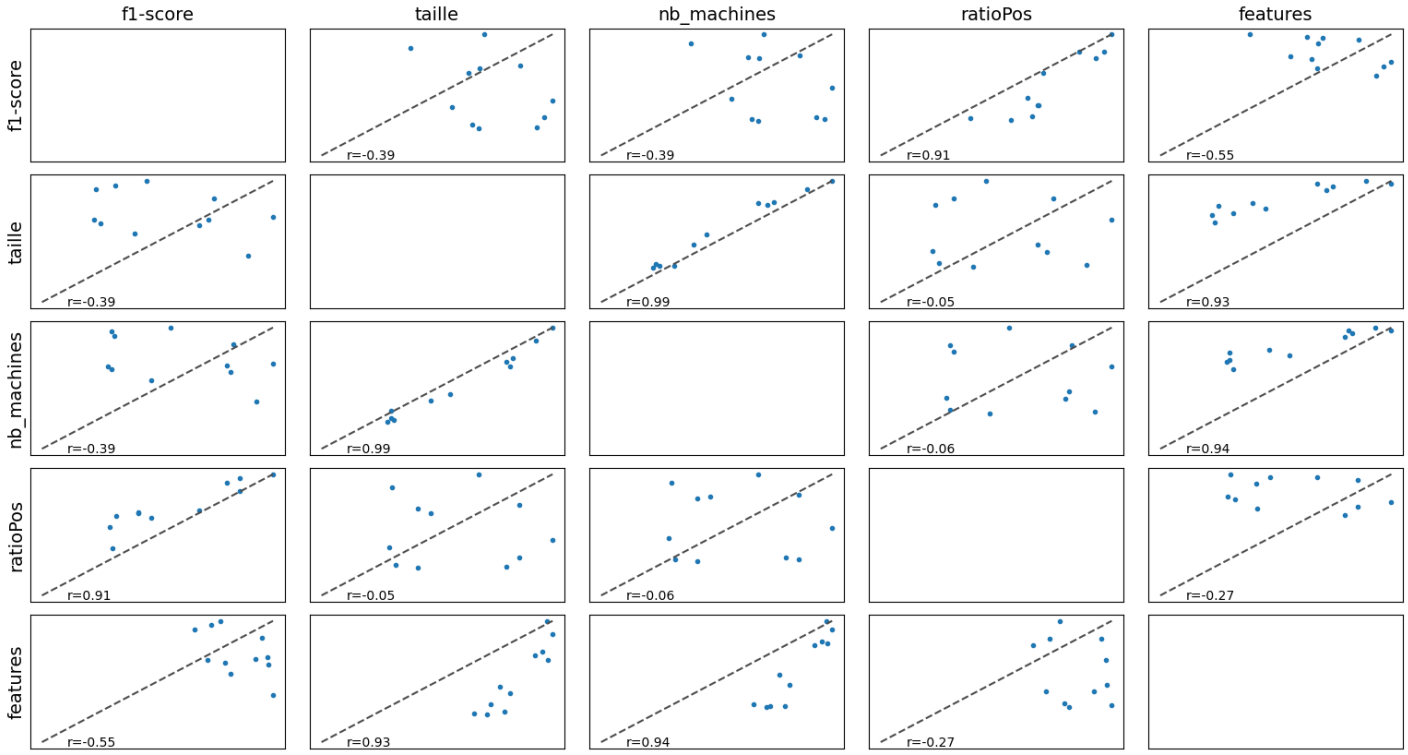

FIGURE 4.14 – Évolution des performances (moyenne des 10 itérations de cross validation) des 11 modèles de prédiction en fonction des caractéristiques statistiques du jeu d'entraînement.

Sans surprise, la figure 4.14 démontre une corrélation positive pour les couples suivants :

— le nombre de machines et la taille du jeu de données,

— le nombre de machines et le nombre de *features*,

— le nombre de *features* et la taille du jeu de données

Concernant le F1-score, par rapport à chacune des caractéristiques, seul le ratio initial de *bags* positifs semble avoir une influence sur les performances du modèle. Cela confirme

les conclusions que nous avons eues dans la section 4.2 : à partir d'un jeu de données, même de taille faible, le modèle parviendra à prédire la survenue des erreurs cibles, pourvu qu'il y ait suffisamment d'informations pour discriminer ces *bags* par rapport aux *bags* négatifs.

4.3.5 Temps d'exécution machine

En moyenne, l'entraînement du modèle pour une erreur cible donnée en validation croisée de 10 itérations prend 1 h 30 sur un processeur non graphique (CPU), mais le temps d'exécution varie selon les jeux de données d'entraînement et leurs caractéristiques. Pour comparaison, l'apprentissage à l'aide du modèle DNN prenait moins de 30 min avec les mêmes configurations du processeur.

Le tableau 4.15 met en exergue les performances du GRU, rapprochées du temps d'exécution.

Erreur cible	# *machines*	# *bags*	# *features*	*ratio de positifs initial*	*F1-score*	*temps d'entraînement*
$E1$	70	3705	189	0.184	0.645 ± 0.05	3h44m
$E2$	22	869	128	0.217	0.769 ± 0.07	0h27m
$E3$	20	836	139	0.113	0.660 ± 0.11	0h14m
$E4$	21	936	154	0.087	0.723 ± 0.15	0h16m
$E5$	57	2517	190	0.128	0.564 ± 0.08	1h46m
$E6$	56	2664	179	0.112	0.668 ± 0.05	1h28m
$E7$	58	2763	185	0.232	0.748 ± 0.04	3h21m
$E8$	39	1685	150	0.271	0.833 ± 0.03	1h44m
$E9$	21	1170	142	0.232	0.822 ± 0.07	0h53m
$E10$	33	1483	158	0.079	0.577 ± 0.10	0h24m
$E11$	65	3298	194	0.082	0.492 ± 0.05	1h31m

FIGURE 4.15 – Statistique des jeux d'entraînement et f1-score associés à chaque modèle

Sans surprise, le temps d'exécution augmente d'autant plus que la taille du jeu de données d'entraînement est grande. Le nombre de *features* ne semble pas impacter le ralentissement de l'apprentissage. En effet, l'erreur $E10$ est associée à un plus grand nombre de *features* que l'erreur $E8$, la taille du jeu d'entraînement étant similaire pour ces deux cas. Toutefois, le modèle associé à l'erreur $E10$ a été entraîné en 3 fois moins de temps.

Conclusion

Les modèles de prédiction basés sur la récurrence et la succession des événements n'ont pas démontré dans l'absolu de meilleures performances qu'une approche méta-instance, ce qui laisse à penser que le modèle DNN, plus simple à mettre en place, pourrait être conservé comme solution à l'objectif de cette thèse. Toutefois, pour certaines erreurs cibles, des progrès ont été observés. En effet, dans le meilleur des cas, le RNN est capable de prédire la survenue d'une erreur cible avec un F1-score de 0.83 pour l'erreur $E8$ avec un modèle GRU, et pour l'erreur $E9$ avec un modèle LSTM.

Pour tenter de corriger le déséquilibre des classes qui pourrait être la cause des résultats dégradés pour les autres erreurs cibles, nous avons appliqué différents ratios de sous-échantillonnage adaptatif. Comme pour le modèle DNN, cela n'a pas permis de corriger ce problème.

Faire varier la valeur du paramètre PI a révélé l'existence de deux profils de résultats. Cette observation est d'autant plus intéressante que l'approche méta-instance développée dans le chapitre 3 n'avait pas permis de révéler ces profils et notre conclusion avait été que la taille minimale requise pour cet intervalle de prédiction était de 5 jours. Pour l'approche basée sur un réseau de neurones récurrents, nous pouvons en conclure qu'en fonction des modèles, la taille minimale requise n'est pas la même.

Une faiblesse du modèle GRU réside dans la difficulté à généraliser la prédiction. En effet, la comparaison des performances par rapport à l'approche méta-instances a révélé, sur plus de la moitié des erreurs cibles, une dégradation des scores initialement obtenus avec le DNN.
Par ailleurs, l'architecture des modèles a été mise au point en se basant sur les performances de prédiction associée à l'erreur $E8$ uniquement. Étant donné que les expériences de ce chapitre nous ont permis de distinguer plusieurs profils potentiels parmi les erreurs critiques, un bon compromis entre un modèle générique et un modèle adapté serait de proposer une solution de MP par profil d'erreur cible.

Enfin, une autre perspective de travaux serait de réadapter l'architecture du réseau de neurones profonds afin de combiner l'approche méta-instances à l'approche RNN.

Conclusion et perspectives

Bilan des contributions

Dans les travaux de cette thèse, nous avons proposé une approche inspirée du *Multiple Instance Learning* permettant de structurer les données issues des fichiers *logs* sous la forme de *bags*.
Les *bags* obtenus sont ensuite transformés en méta-instances, qui peuvent alimenter un algorithme d'apprentissage supervisé. Trois paramètres ont été introduits dans cette approche (PI, RI et EI). Chacun d'eux permet d'ajuster respectivement l'intervalle de temps durant lequel les données sont considérées pour effectuer la prédiction, la période d'anticipation liée à la prédiction, et l'intervalle de temps durant lequel on s'attend à observer la survenue ou non de l'événement à prédire.

Notre méthodologie peut être adoptée pour tout contexte industriel qui présuppose l'émission quotidienne de fichiers *logs*. Dans le cadre de cette thèse CIFRE, nous avons appliqué cette méthodologie sur un ensemble de données issues d'une flotte de machines de découpe.
Nous avons cherché à prédire la survenue de certaines erreurs de haute sévérité. Pour chacune d'entre elles, un classifieur à deux classes est entraîné afin de prédire la survenue ou la non survenue de l'erreur de haute sévérité. Nous avons exploré cinq algorithmes de classification candidats pour construire le modèle : Arbre de Décision, Forêt Aléatoire, Bayésien naïf, SVM et DNN. Parmi ces cinq algorithmes, le DNN s'est avéré être le meilleur candidat avec un F1-score maximal de 0.80, soit 0.3 de plus comparé à une méthode de l'état de l'art [SFMW14] appliquée sur les mêmes données.

Nous avons également cherché à tenir compte du caractère séquentiel des événements au travers d'une seconde approche de prédiction basée sur un réseau de neurones récurrents (RNN). Deux modèles RNN ont été comparés, à savoir un modèle LSTM et un modèle GRU.
Les résultats obtenus ne nous ont pas permis de statuer définitivement sur l'avantage

d'un modèle séquentiel comparé à un modèle méta-instances. Cependant, les scores de performance obtenus pour le modèle GRU se sont révélés suffisamment satisfaisants pour conclure sur l'utilité d'un RNN dans un tel cas d'usage.
De manière générale, dans notre contexte, l'approche basée sur l'apprentissage profond constitue le meilleur candidat pour traiter les données de fichiers *logs*. Parmi les trois modèles d'apprentissage profond que nous avons exploré dans cette thèse (DNN, GRU et LSTM), il n'y pas de gagnant absolu et utiliser une approche méta-instances, basée sur le MIL (DNN) plutôt qu'une approche séquentielle (RNN) constitue pour le moment la solution la plus simple à mettre en place, par rapport à un modèle GRU ou LSTM.

Nous avons cherché à corriger le problème de déséquilibre des classes inhérent à notre cas d'usage soit par l'augmentation de l'effectif de la classe positive, soit par la réduction de celui de la classe négative. Les différentes méthodes que nous avons employées ont conduit à une dégradation des résultats de la performance des trois modèles d'apprentissage profond. Cela suggère que ces modèles sont capables de tenir compte de la présence très marquée des *bags* négatifs, et d'en tirer profit pour la prédiction.
Par ailleurs, nous avons également fait varier les différents paramètres de construction des données (PI, RI et EI) afin d'en déterminer les valeurs optimales. Les expérimentations effectuées avec le modèle DNN ont révélé de meilleures capacités pour le modèle de prédiction lorsque PI est supérieur à 5 jours. En parallèle, le modèle est capable de prédire la survenue de l'erreur critique plus de 20 jours en avance. Le score de performance est d'autant plus grand que la valeur choisie pour EI est importante, mais puisque cela diminue la finesse de prédiction du modèle, notre conclusion est qu'il est préférable de garder une valeur autour de 4 jours, qui constitue un bon compromis entre la finesse et la performance de prédiction du modèle.
Les mêmes expérimentations, effectuées avec le modèle GRU ont révélé deux profils différents concernant la taille idéale de PI. Pour chacun de ces profils, la valeur de PI peut être fixée à 15 et 10 jours. En ce qui concerne la valeur du paramètre RI, elle pourra être fixée respectivement à 15 jours et 20 jours.

Finalement, quel que soit le modèle d'apprentissage profond choisi, notre solution de maintenance prédictive est capable de prédire au minimum 20 jours en avance une erreur de sévérité élevée, avec un F1-score de plus de 0.8 pour le meilleur modèle.

Perspectives

Les analyses du chapitre 4 ont démontré deux groupes d'erreurs critiques partageant des valeurs similaires de paramètres PI, RI et EI. Aussi, il pourrait être intéressant d'explorer une approche multi-classes pour chaque groupe d'erreurs, ce qui nous permettrait de proposer non pas 11 solutions de maintenance prédictive, dédiées aux 11 erreurs cibles, mais plutôt deux solutions de MP (une pour chaque groupe d'erreurs cibles).

Pour le moment, nous n'avons construit les modèles de prédiction que sur la base de la connaissance du nombre d'erreurs de chaque type, émises par les machines.
Ajouter de l'information prédictive dans le modèle à partir de données d'usage, par exemple le niveau d'usure de chaque outil de la machine ou le nombre de tâches effectuées par la machine depuis la dernière maintenance, peut-être une piste favorable à l'amélioration de la performance du modèle prédictif.

Toujours dans l'idée de rendre les variables prédictives plus discriminantes, nous pourrions donner plus de spécificité aux *features* du modèle DNN, en les combinant à des approches d'extraction de motifs. Nous avons vu dans le chapitre 1 que les chroniques [DD99] sont une forme de motif, associé aux dépendances temporelles des événements qui le contiennent. L'utilisation des chroniques pour créer des motifs nous permettrait de conserver les dépendances temporelles intrinsèques aux données, tout en construisant des variables (correspondant aux motifs extraits) nécessaires au modèle d'apprentissage. Cet usage nous permettrait ainsi de considérer les phénomènes d'émission en rafales, caractérisés par l'émission soudaine et simultanée de certains événements.

Enfin, nous pourrions évidemment modifier l'architecture du modèle d'apprentissage profond adopté. Comme évoqué dans le chapitre 4, puisque le modèle DNN et le modèle récurrent ont tous les deux apporté des résultats satisfaisants, nous pourrions explorer la combinaison de ces deux modèles, afin de tenir compte à la fois d'une vue instantanée de l'Intervalle Prédictif et d'une vue séquentielle. Des travaux similaires [WCN+21] à cette idée ont fait leur preuve pour la prédiction de survenue d'événements de type *log* et pourraient constituer une piste possible.
Récemment, une nouvelle architecture de réseau de neurones profond a été proposée, basée sur un mécanisme de l'attention [VSP+17] : le réseau Transformer. Cette méthode, différente du mécanisme de la récurrence proposé par le modèle LSTM et le modèle GRU, est reconnue pour la rapidité avec laquelle le modèle peut être entraîné. Par ailleurs, même si ce réseau est capable de prédire un événement à partir d'observations passées, ces dernières ne doivent pas nécessairement être transmises au réseau sous la forme d'une séquence, ce qui permettrait de paralléliser les calculs.

Bibliographie

[AFGY02] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 429–435, 2002.

[AG18] Nagdev Amruthnath and Tarun Gupta. Fault class prediction in unsupervised learning using model-based clustering approach. In *2018 International Conference on Information and Computer Technologies (ICICT)*, pages 5–12, DeKalb, IL, March 2018. IEEE.

[AJO+18] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications : A survey. *Heliyon*, 4(11) :e00938, November 2018.

[ALRL04] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1) :11–33, 2004.

[AMZ09] Safaa O. Al-Mamory and Hongli Zhang. Intrusion detection alarms reduction using root cause analysis and clustering. *Computer Communications*, 32(2) :419–430, 2009.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499. Morgan Kaufmann, 1994.

[AS95] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 3–14, 1995.

[ASSZ17] A. Abu-Samah, M. K. Shahzad, and E. Zamai. Bayesian based methodology for the extraction and validation of time bound failure signatures for online failure prediction. *Reliability Engineering & System Safety*, 167 :616–628, November 2017.

[Bac08] Francis R. Bach. Bolasso : model consistent Lasso estimation through the bootstrap. In *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 33–40, Helsinki, Finland, 2008. ACM Press.

[Bas14] F. Basile. Overview of fault diagnosis methods based on Petri net models. In *2014 European Control Conference (ECC)*, pages 2636–2642, 2014.

[BNJL03] David M. Blei, Andrew Y. Ng, Michael I. Jordan, and John Lafferty. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3(4/5) :993–1022, May 2003.

[Bre01] Leo Breiman. Random Forests. *Machine Learning*, 45(1) :5–32, October 2001.

[BTR15] Paula Branco, Luis Torgo, and Rita Ribeiro. A Survey of Predictive Modelling under Imbalanced Distributions. *arXiv :1505.01658 [cs]*, May 2015. arXiv : 1505.01658.

[CANZ+20] Zeki Murat Cinar, Abubakar Abdussalam Nuhu, Qasim Zeeshan, Orhan Korhan, Mohammed Asmael, and Babak Safaei. Machine Learning in Predictive Maintenance towards Sustainable Smart Manufacturing in Industry 4.0. *Sustainability*, 12(19) :8211, January 2020. Number : 19 Publisher : Multidisciplinary Digital Publishing Institute.

[CBHK02] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE : Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16 :321–357, June 2002.

[CCC16] Zhicheng Cui, Wenlin Chen, and Yixin Chen. Multi-Scale Convolutional Neural Networks for Time Series Classification. *arXiv :1603.06995 [cs]*, May 2016. arXiv : 1603.06995.

[CGHS15] Maria Paola Cabasino, Alessandro Giua, Christoforos N. Hadjicostis, and Carla Seatzu. Fault model identification and synthesis in Petri nets. *Discrete Event Dynamic Systems*, 25(3) :419–440, 2015.

[CGLRML20] Jair Cervantes, Farid Garcia-Lamont, Lisbeth Rodríguez-Mazahua, and Asdrubal Lopez. A comprehensive survey on support vector machine

classification : Applications, challenges and trends. *Neurocomputing*, 408 :189–215, September 2020.

[CGPS11] M. P. Cabasino, A. Giua, M. Pocci, and C. Seatzu. Discrete event diagnosis using labeled Petri nets. An application to manufacturing systems. *Control Engineering Practice*, 19 :989–1001, 2011.

[Cho20] K. R. Chowdhary. Natural Language Processing. In K.R. Chowdhary, editor, *Fundamentals of Artificial Intelligence*, pages 603–649. Springer India, New Delhi, 2020.

[CL11] Chih-Chung Chang and Chih-Jen Lin. LIBSVM : A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3) :27 :1–27 :27, 2011.

[CMM12] Damien Cram, Benoît Mathern, and Alain Mille. A complete chronicle discovery approach : application to activity analysis. *Expert Systems*, 29 :321–346, 2012.

[CPLA16] Arno Candel, Viraj Parmar, Erin LeDell, and Anisha Arora. Deep learning with h2o. *H2O. ai Inc*, 2016.

[CTL16] Asha Chigurupati, Romain Thibaux, and Noah Lassar. Predicting hardware failure using machine learning. In *2016 Annual Reliability and Maintainability Symposium (RAMS)*, pages 1–6, Tucson, AZ, USA, January 2016. IEEE.

[CvMG+14] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, 2014. Association for Computational Linguistics.

[CWDW20] Jia Chen, Peng Wang, Shiqing Du, and Wei Wang. Log Pattern Mining for Distributed System Maintenance. *Complexity*, 2020 :e6628165, 2020.

[CYLZ21] YunFei Chen, Jianping Yuan, Yin Luo, and Wenqi Zhang. Fault Prediction of Centrifugal Pump Based on Improved KNN. *Shock and Vibration*, 2021 :e7306131, October 2021. Publisher : Hindawi.

[DA10] René M. G. David and Hassane Alla. *Discrete, Continuous, and Hybrid Petri Nets*. Springer, 2010.

[DB21] Alican Dogan and Derya Birant. Machine learning and data mining in manufacturing. *Expert Systems with Applications*, 166 :114060, 2021.

[DD99] Christophe Dousson and Thang Vu Duong. Discovering chronicles with numerical time constraints from alarm logs for monitoring dynamic systems. In *IJCAI*, volume 99, pages 620–626, 1999.

[DGAGH19] Yann Dauxais, David Gross-Amblard, Thomas Guyet, and André Happe. Discriminant Chronicle Mining. In Bruno Pinaud, Fabrice Guillet, Fabien Gandon, and Christine Largeron, editors, *Advances in Knowledge Discovery and Management : Volume 8*, pages 89–118. Springer International Publishing, 2019.

[DH73] Richard O. Duda and Peter E. Hart. *Pattern Classification and scene analysis.* Wiley, a wiley-interscience publication edition, 1973.

[DLLP97] Thomas G. Dietterich, Richard H. Lathrop, and Tomás Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2) :31–71, 1997. Publisher : Elsevier.

[DVK17] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv*, 2017.

[FLCM16] Fabio Fumarola, Pasqua Fabiana Lanotte, Michelangelo Ceci, and Donato Malerba. CloFAST : closed sequential pattern mining using sparse and vertical id-lists. *Knowledge and Information Systems*, 48(2) :429–463, 2016.

[FVGCT14] Philippe Fournier-Viger, Antonio Gomariz, Manuel Campos, and Rincy Thomas. Fast Vertical Mining of Sequential Patterns Using Co-occurrence Information. In *Advances in Knowledge Discovery and Data Mining*, volume 8443, pages 40–52. Springer International Publishing, Cham, 2014.

[FVLK+17] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Rage Uday Kiran, Yun Sing Koh, and Rincy Thomas. A survey of sequential pattern mining. *Data Science and Pattern Recognition*, 1(1) :54–77, 2017.

[GCMG13] Antonio Gomariz, Manuel Campos, Roque Marin, and Bart Goethals. ClaSP : An Efficient Algorithm for Mining Frequent Closed Sequences. In *Advances in Knowledge Discovery and Data Mining*, volume 7818, pages 50–61. Springer Berlin Heidelberg, 2013.

[GE03] Isabelle Guyon and André Elisseeff. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3(Mar) :1157–1182, 2003.

[GFS+19] Clemens Gutschi, Nikolaus Furian, Josef Suschnigg, Dietmar Neubacher, and Siegfried Voessner. Log-based predictive maintenance in discrete parts manufacturing. *Procedia CIRP*, 79 :528–533, January 2019.

[Gol17] Yoav Goldberg. Neural network methods for natural language processing. *Synthesis lectures on human language technologies*, 10(1) :1–309, 2017.

[GRW16] Xinze Guan, Raviv Raich, and Weng-Keen Wong. Efficient multi-instance learning for activity recognition from time series data using an auto-regressive hidden markov model. In *International Conference on Machine Learning*, pages 2330–2339. PMLR, 2016.

[GTDVMFV09] Pedro Garcia-Teodoro, Jesus Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection : Techniques, systems and challenges. *computers & security*, 28(1-2) :18–28, 2009.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-term Memory. *Neural computation*, 9 :1735–80, December 1997.

[HVB+16] Francisco Herrera, Sebastián Ventura, Rafael Bello, Chris Cornelis, Amelia Zafra, Dánel Sánchez-Tarragó, and Sarah Vluymans. *Multiple Instance Learning : Foundations and Algorithms*. Springer International Publishing, 2016.

[HX98] S. L. Ho and M. Xie. The use of ARIMA models for reliability forecasting and analysis. *Computers & Industrial Engineering*, 35(1) :213–216, October 1998.

[HYGS08] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li. ADASYN : Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 1322–1328, Hong Kong, China, June 2008. IEEE.

[IB97] R. Isermann and P. Ballé. Trends in the application of model-based fault detection and diagnosis of technical processes. *Control Engineering Practice*, 5(5) :709–719, 1997.

[Ise11] Rolf Isermann. Fault diagnosis of machine tools. In Rolf Isermann, editor, *Fault-Diagnosis Applications : Model-Based Condition Monitoring : Actuators, Drives, Machinery, Plants, Sensors, and Fault-tolerant Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[ITW18] Maximilian Ilse, Jakub M. Tomczak, and Max Welling. Attention-based Deep Multiple Instance Learning. *arXiv :1802.04712 [cs, stat]*, June 2018. arXiv : 1802.04712.

[JMM96] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. Artificial neural networks : A tutorial. *Computer*, 29(3) :31–44, 1996.

[JQS+15] Shuhui Jiang, Xueming Qian, Jialie Shen, Yun Fu, and Tao Mei. Author Topic Model based Collaborative Filtering for Personalized POI Recommendation. *IEEE Transactions on Multimedia*, pages 1–1, 2015.

[JSS09] Szymon Jaroszewicz, Tobias Scheffer, and Dan A. Simovici. Scalable pattern mining with Bayesian networks as background knowledge. *Data Mining and Knowledge Discovery*, 18(1) :56–100, February 2009.

[KB14] Diederik P Kingma and Jimmy Ba. Adam : A method for stochastic optimization. *arXiv preprint arXiv :1412.6980*, 2014.

[KBV18] P. Korvesis, S. Besseau, and M. Vazirgiannis. Predictive Maintenance in Aviation : Failure Prediction from Post-Flight Reports. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1414–1422, April 2018.

[KFRM13] Bernard Kamsu-Foguem, Fabien Rigal, and Félix Mauget. Mining association rules for the quality improvement of the production process. *Expert systems with applications*, 40(4) :1034–1045, 2013.

[KKČ+18] Jan Kohout, Tomáš Komárek, Přemysl Čech, Jan Bodnár, and Jakub Lokoč. Learning communication patterns for malware discovery in HTTPs data. *Expert Systems with Applications*, 101 :129–142, July 2018.

[KKKK21] Abdulgani Kahraman, Mehmed Kantardzic, M. Mustafa Kahraman, and Muhammed Kotan. Sequential Pattern Mining Method for Predictive Maintenance of Large Mining Trucks. In Jasminka Hasic Telalovic and Mehmed Kantardzic, editors, *Mediterranean Forum – Data Science Conference*, pages 126–136. Springer International Publishing, 2021.

[KSG+21] Mustafa Kuntoğlu, Emin Salur, Munish Kumar Gupta, Murat Sarıkaya, and Danil Yu. Pimenov. A state-of-the-art review on sensors and signal processing systems in mechanical machining processes. *The International Journal of Advanced Manufacturing Technology*, 116(9) :2711–2735, October 2021.

[KVIT18] Nikolaos Kolokas, Thanasis Vafeiadis, Dimosthenis Ioannidis, and Dimitrios Tzovaras. Forecasting faults of industrial equipment using machine learning classifiers. In *2018 Innovations in Intelligent Systems and Applications (INISTA)*, pages 1–6. IEEE, 2018.

[KWTI15] T. Kimura, A. Watanabe, T. Toyono, and K. Ishibashi. Proactive failure detection learning generation patterns of large-scale network logs. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 8–14, November 2015.

[Laf19] Stéphane Lafortune. Discrete Event Systems : Modeling, Observation, and Control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2(1) :141–159, 2019.

[LAH06] Yang Liu, Aijun An, and Xiangji Huang. Boosting Prediction Accuracy on Imbalanced Datasets with SVM Ensembles. In Wee-Keong Ng, Masaru Kitsuregawa, Jianzhong Li, and Kuiyu Chang, editors, *Advances in Knowledge Discovery and Data Mining*, Lecture Notes in Computer Science, pages 107–118, Berlin, Heidelberg, 2006. Springer.

[Lap92] J. C. Laprie. Dependability : Basic Concepts and Terminology. In J. C. Laprie, editor, *Dependability : Basic Concepts and Terminology : In English, French, German, Italian and Japanese*. Springer, 1992.

[LBADM21a] Myriam Lopez, Marie Beurton-Aimar, Gayo Diallo, and Sofian Maabout. Approche de traitement des logs pour la prédiction d'erreurs critiques. *Extraction et Gestion des Connaissances : Actes EGC'2021*, 1(2) :1, 2021.

[LBADM21b] Myriam Lopez, Marie Beurton-Aimar, Gayo Diallo, and Sofian Maabout. Log Data Preparation for Predicting Critical Errors Occurrences. In Álvaro Rocha, Hojjat Adeli, Gintautas Dzemyda, Fernando Moreira, and Ana Maria Ramalho Correia, editors, *Trends and Applications in Information Systems and Technologies*, pages 224–233, Cham, 2021. Springer International Publishing.

[LBADM22] Myriam Lopez, Marie Beurton-Aimar, Gayo Diallo, and Sofian Maabout. A simple yet effective approach for log based critical errors prediction. *Computers in Industry*, 137 :103605, 2022.

[LBM+18] Konstantinos G. Liakos, Patrizia Busato, Dimitrios Moshou, Simon Pearson, and Dionysis Bochtis. Machine Learning in Agriculture : A

Review. *Sensors*, 18(8) :2674, August 2018. Number : 8 Publisher : Multidisciplinary Digital Publishing Institute.

[LC03] Dorron Levy and Ram Chillarege. Early warning of failures through alarm analysis a case study in telecom voice mail systems. In *14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.*, pages 271–280. IEEE, 2003.

[LCDF+15] Anna Leontjeva, Raffaele Conforti, Chiara Di Francescomarino, Marlon Dumas, and Fabrizio Maria Maggi. Complex Symbolic Sequence Encodings for Predictive Monitoring of Business Processes. In Hamid Reza Motahari-Nezhad, Jan Recker, and Matthias Weidlich, editors, *Business Process Management*, Lecture Notes in Computer Science, pages 297–313. Springer International Publishing, 2015.

[LGMT16] Arthur Le Guennec, Simon Malinowski, and Romain Tavenard. Data augmentation for time series classification using convolutional neural networks. In *ECML/PKDD workshop on advanced analytics and learning on temporal data*, 2016.

[LHT12] Wei-Yang Lin, Ya-Han Hu, and Chih-Fong Tsai. Machine Learning in Financial Crisis Prediction : A Survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4) :421–436, July 2012. Conference Name : IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews).

[LKK17] Hwa Kyung Lim, Yongdai Kim, and Min-Kyoon Kim. Failure Prediction Using Sequential Pattern Mining in the Wire Bonding Process. *IEEE Transactions on Semiconductor Manufacturing*, 30 :285–292, 2017.

[LS95] Pat Langley and Herbert A. Simon. Applications of Machine Learning and Rule Induction. *Commun. ACM*, 38, 1995.

[LSW+17] Jing Li, Rebecca J. Stones, Gang Wang, Xiaoguang Liu, Zhongwei Li, and Ming Xu. Hard drive failure prediction using Decision Trees. *Reliability Engineering & System Safety*, 164 :55–65, August 2017.

[LTD13] Renaud Lambiotte, Lionel Tabourier, and Jean-Charles Delvenne. Burstiness and spreading on temporal networks. *The European Physical Journal B*, 86(7) :320, July 2013.

[LW+02] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3) :18–22, 2002.

[LWL+19] B. Luo, H. Wang, H. Liu, B. Li, and F. Peng. Early Fault Detection of Machine Tools Based on Deep Learning and Dynamic Identification. *IEEE Transactions on Industrial Electronics*, 66(1) :509–518, January 2019.

[LYC+18] Guang Li, Xin Yang, Duanbing Chen, Anxing Song, Yuke Fang, and Junlin Zhou. Tool Breakage Detection using Deep Learning. *arXiv :1808.05347 [cs, stat]*, August 2018. arXiv : 1808.05347.

[MHKDS05] Joseph F Murray, Gordon F Hughes, Kenneth Kreutz-Delgado, and Dale Schuurmans. Machine learning methods for predicting failures in hard drives : A multiple-instance application. *Journal of Machine Learning Research*, 6(5), 2005.

[MTB+21] Ioannis Mavroudopoulos, Theodoros Toliopoulos, Christos Bellas, Andreas Kosmatopoulos, and Anastasios Gounaris. Sequence detection in event log files. In *EDBT*, pages 85–96, 2021.

[MWS13] M. M. Mansour, Mohamed A. A. Wahab, and Wael M. Soliman. Petri nets for fault diagnosis of large power generation station. *Ain Shams Engineering Journal*, 4(4) :831–842, 2013.

[NND+20] Srikanth Namuduri, Barath Narayanan Narayanan, Venkata Salini Priyamvada Davuluru, Lamar Burton, and Shekhar Bhansali. Review—Deep Learning Methods for Sensor Based Predictive Maintenance and Future Perspectives for Electrochemical Sensors. *Journal of The Electrochemical Society*, 167(3) :037552, 2020.

[NPF18] Samad Nejatian, Hamid Parvin, and Eshagh Faraji. Using sub-sampling and ensemble clustering techniques to improve performance of imbalanced classification. *Neurocomputing*, 276 :55–66, 2018.

[OKFE17] Kazuki Otomo, Satoru Kobayashi, Kensuke Fukuda, and Hiroshi Esaki. An Analysis of Burstiness and Causality of System Logs. In *Proceedings of the Asian Internet Engineering Conference*, AINTEC '17, pages 16–23, New York, NY, USA, November 2017. Association for Computing Machinery.

[Pet09] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2) :1883, 2009.

[PZHK05] Louchka Popova-Zeugmann, Monika Heiner, and Ina Koch. Time petri nets for modelling and analysis of biochemical networks. *Fundamenta Informaticae*, 67(1-3) :149–162, 2005.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1) :81–106, March 1986.

[Rab89] L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2) :257–286, February 1989. Conference Name : Proceedings of the IEEE.

[RBVdW14] Adria Ruiz, Xavier Binefa, and Joost Van de Weijer. Regularized Multi-Concept MIL for weakly-supervised facial behavior categorization. In *Proceedings of the British Machine Vision Conference 2014*, pages 13.1–13.12, Nottingham, 2014. British Machine Vision Association.

[RDGC95a] David E Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. Backpropagation : The basic theory. *Backpropagation : Theory, architectures and applications*, pages 1–34, 1995.

[RDGC95b] David E Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. Backpropagation : The basic theory. *Backpropagation : Theory, architectures and applications*, pages 1–34, 1995.

[RF11] L. Todd Rose and Kurt W. Fischer. Garbage In, Garbage Out : Having Useful Data Is Everything. *Measurement : Interdisciplinary Research and Perspectives*, 9 :222–226, 2011.

[RFC+20] Alberto Rivas, Jesús M. Fraile, Pablo Chamoso, Alfonso González-Briones, Inés Sittón, and Juan M. Corchado. A Predictive Maintenance Model Using Recurrent Neural Networks. In *14th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2019)*, pages 261–270. Springer International Publishing, 2020.

[RLL09] Matthias Roth, Jean-Jacques Lesage, and Lothar Litz. A residual inspired approach for fault localization in DES. *IFAC Proceedings Volumes*, 42(5) :305–310, 2009.

[Sam00] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 44(1.2) :206–226, January 2000. Conference Name : IBM Journal of Research and Development.

[SB16] Gian Antonio Susto and Alessandro Beghi. Dealing with time-series data in Predictive Maintenance problems. In *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on*, pages 1–4. IEEE, 2016.

[SFMW14] Ruben Sipos, Dmitriy Fradkin, Fabian Moerchen, and Zhuang Wang. Log-based predictive maintenance. In *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1867–1876. ACM Press, 2014.

[SHIH02] Tinghsu Su, S. Hattori, M. Ishida, and T. Hori. Suppression control method for torque vibration of AC motor utilizing repetitive controller with Fourier transform. *IEEE Transactions on Industry Applications*, 38(5) :1316–1325, September 2002. Conference Name : IEEE Transactions on Industry Applications.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout : a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1) :1929–1958, January 2014.

[SM07] Felix Salfner and Miroslaw Malek. Using Hidden Semi-Markov Models for Effective Online Failure Prediction. In *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*, pages 161–174, October 2007. ISSN : 1060-9857.

[SMS+20] Chayma Sellami, Carlos Miranda, Ahmed Samet, Mohamed Anis Bach Tobji, and François de Beuvron. On mining frequent chronicles for machine failure prediction. *Journal of Intelligent Manufacturing*, 31 :1019–1035, 2020.

[SOR+03] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 426–435, New York, NY, USA, 2003. Association for Computing Machinery.

[SOvdAtH13] Suriadi Suriadi, Chun Ouyang, Wil M. P. van der Aalst, and Arthur H. M. ter Hofstede. Root Cause Analysis with Enriched Process Logs. In Marcello La Rosa and Pnina Soffer, editors, *Business Process Management Workshops*, pages 174–186. Springer Berlin Heidelberg, 2013.

[SPG+17] Amit Saxena, Mukesh Prasad, Akshansh Gupta, Neha Bharill, Om Prakash Patel, Aruna Tiwari, Meng Joo Er, Weiping Ding, and Chin-Teng Lin. A review of clustering techniques and developments. *Neurocomputing*, 267 :664–681, December 2017.

[SQS04] Jing Sun, Shi-Yin Qin, and Yong-Hua Song. Fault diagnosis of electric power systems based on fuzzy Petri nets. *IEEE Transactions on Power Systems*, 19 :2053–2059, 2004.

[SR21] Aman Sharma and Rinkle Rani. A Systematic Review of Applications of Machine Learning in Cancer Prediction and Diagnosis. *Archives of Computational Methods in Engineering*, 28(7) :4875–4896, December 2021.

[SWLY18] Jing Shen, Jian Wan, Se-Jung Lim, and Lifeng Yu. Random-forest-based failure prediction for hard disk drives. *International Journal of Distributed Sensor Networks*, 14(11) :1550147718806480, November 2018. Publisher : SAGE Publications.

[Tib96] Robert Tibshirani. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1) :267–288, 1996. Publisher : [Royal Statistical Society, Wiley].

[TK21] Jasminka Hasic Telalovic and Mehmed Kantardzic. *Mediterranean Forum – Data Science Conference*. Springer Nature, 2021.

[TTBS22] Dimpal Tomar, Pradeep Tomar, Arpit Bhardwaj, and GR Sinha. Deep learning neural network prediction system enhanced with best window size in sliding window algorithm for predicting domestic power consumption in a residential building. *Computational Intelligence and Neuroscience*, 2022, 2022.

[TYTS01] P.Y.L. Tu, R. Yam, P. Tse, and A.O. Sun. An Integrated Maintenance Management System for an Advanced Manufacturing Company. *The International Journal of Advanced Manufacturing Technology*, 17(9) :692–703, 2001.

[VAH+02] R. Vilalta, C. V. Apte, J. L. Hellerstein, S. Ma, and S. M. Weiss. Predictive algorithms in the management of computer systems. *IBM Systems Journal*, 41(3) :461–474, 2002.

[Vap00] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Information Science and Statistics. Springer-Verlag, New York, 2 edition, 2000.

[VJA+19] Gregory W. Vogl, N. Jordan Jameson, Andreas Archenti, Károly Szipka, and M. Alkan Donmez. Root-cause analysis of wear-induced error motion changes of machine tool linear axes. *International Journal of Machine Tools and Manufacture*, 143 :38–48, 2019.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[WCN+21] Zhiwei Wang, Zhengzhang Chen, Jingchao Ni, Hui Liu, Haifeng Chen, and Jiliang Tang. Multi-Scale One-Class Recurrent Neural Networks for Discrete Event Sequence Anomaly Detection. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, pages 3726–3734, New York, NY, USA, 2021. Association for Computing Machinery.

[WH04] J. Wang and J. Han. BIDE : efficient mining of frequent closed sequences. In *Proceedings. 20th International Conference on Data Engineering*, pages 79–90, Boston, MA, USA, 2004. IEEE Comput. Soc.

[WJT+17] Dazhong Wu, Connor Jennings, Janis Terpenny, Robert X. Gao, and Soundar Kumara. A Comparative Study on Machine Learning Algorithms for Smart Manufacturing : Tool Wear Prediction Using Random Forests. *Journal of Manufacturing Science and Engineering*, 139(7) :071018–071018–9, April 2017.

[WL05] T. Warren Liao. Clustering of time series data—a survey. *Pattern Recognition*, 38(11) :1857–1874, November 2005.

[WLH+17] J. Wang, C. Li, S. Han, S. Sarkar, and X. Zhou. Predictive maintenance based on event-log analysis : A case study. *IBM Journal of Research and Development*, 61(1) :11 :121–11 :132, January 2017. Conference Name : IBM Journal of Research and Development.

[WLM19] Yun Wang, Juncheng Li, and Florian Metze. A Comparison of Five Multiple Instance Learning Pooling Functions for Sound Event Detection with Weak Labeling. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 31–35, Brighton, United Kingdom, May 2019. IEEE.

[Wu75] Chialin Wu. *The decision tree approach to classification.* Purdue University, 1975.

[WW18] Kesheng Wang and Yi Wang. How AI Affects the Future Predictive Maintenance : A Primer of Deep Learning. In Kesheng Wang, Yi Wang, Jan Ola Strandhagen, and Tao Yu, editors, *Advanced Manufacturing*

and Automation VII, Lecture Notes in Electrical Engineering, pages 1–9, Singapore, 2018. Springer.

[Xia10] Guang-ming Xian. Mechanical failure classification for spherical roller bearing of hydraulic injection molding machine using DWT-SVM. *Expert Syst. Appl.*, 37 :6742–6747, October 2010.

[YB19] O. E. Yurek and D. Birant. Remaining Useful Life Estimation for Predictive Maintenance Using Feature Engineering. In *2019 Innovations in Intelligent Systems and Applications Conference (ASYU)*, pages 1–5, October 2019.

[YHA03] Xifeng Yan, Jiawei Han, and Ramin Afshar. CloSpan : Mining : Closed Sequential Patterns in Large Datasets. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, Proceedings, pages 166–177. Society for Industrial and Applied Mathematics, 2003.

[YZ15] Ke Yan and David Zhang. Feature selection and analysis on correlated gas sensor data with recursive feature elimination. *Sensors and Actuators B : Chemical*, 212 :353–363, June 2015.

[Zak01] Mohammed Javeed Zaki. SPADE : an efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2) :31–60, 2001.

[Zei12] Matthew D Zeiler. Adadelta : an adaptive learning rate method. *arXiv preprint arXiv :1212.5701*, 2012.

[ZL13] J. Zaytoon and S. Lafortune. Overview of fault diagnosis methods for Discrete Event Systems. *Annual Reviews in Control*, 37(2), 2013.

[ZLC+14] Y. Zheng, Q. Liu, Enhong Chen, Y. Ge, and J.L. Zhao. Time series classification using multi-channels deep convolutional neural networks. *WAIM 2014. LNCS*, 8485 :298–310, January 2014.

[ZLC+16] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J. Leon Zhao. Exploiting multi-channels deep convolutional neural networks for multivariate time series classification. *Frontiers of Computer Science*, 10(1) :96–112, February 2016.

[ZLPG09] Z. Zheng, Z. Lan, B. H. Park, and A. Geist. System log pre-processing to improve failure prediction. In *2009 IEEE/IFIP International Conference on Dependable Systems Networks*, pages 572–577, June 2009.

[ZXM+16] Ke Zhang, Jianwu Xu, Martin Renqiang Min, Guofei Jiang, Konstantinos Pelechrinis, and Hui Zhang. Automated IT system failure prediction :

A deep learning approach. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1291–1300, Washington DC,USA, December 2016. IEEE.

Annexe A

Fonctionnement d'un réseau de neurones profonds

Un réseau de neurones correspond à un ensemble de couches de neurones. chaque neurone effectue un calcul à partir du résultat des neurones de la couche précédente et produit lui-même un résultat. Dans le cadre d'une classification, le réseau de neurones prend en entrée un ensemble d'apprentissage constitué de variables explicatives, que l'on appelle également *features* dans le vocabulaire de l'apprentissage profond. Pour chaque *feature*, un poids est déterminé par le réseau de neurones, grâce aux calculs effectués par l'ensemble des neurones du réseau, lors d'un mécanisme de **propagation vers l'avant**. Afin d'ajuster les poids (ou paramètres) en fonction du jeu de données, le réseau devra évaluer, pour chaque instance, l'erreur de classement, en fonction de laquelle il pourra alors corriger les valeurs des paramètres. Ce mécanisme de correction des paramètres en fonction de l'erreur de classement est appelé **rétro-propagation**.

A.1 Mécanismes de propagation

Les réseaux de neurones sont capables d'approcher toute fonction non linéaire représentative de la distribution des données, aussi complexe soit-elle. Cette approximation s'effectue à l'aide de deux mécanismes :

La propagation vers l'avant (également appelée *forward propagation* dans la littérature du domaine) consiste à calculer l'**activation** a pour chaque neurone du

réseau, à partir d'un vecteur x et d'un vecteur w de poids. Ces calculs s'effectuent depuis la couche d'entrée vers la couche de sortie.

La rétro-propagation du gradient (*backward propagation*)[RDGC95b], dont les calculs vont de la couche de sortie vers la couche d'entrée, permet de mettre à jour les vecteurs de poids w du réseau en fonction de l'erreur de classification.

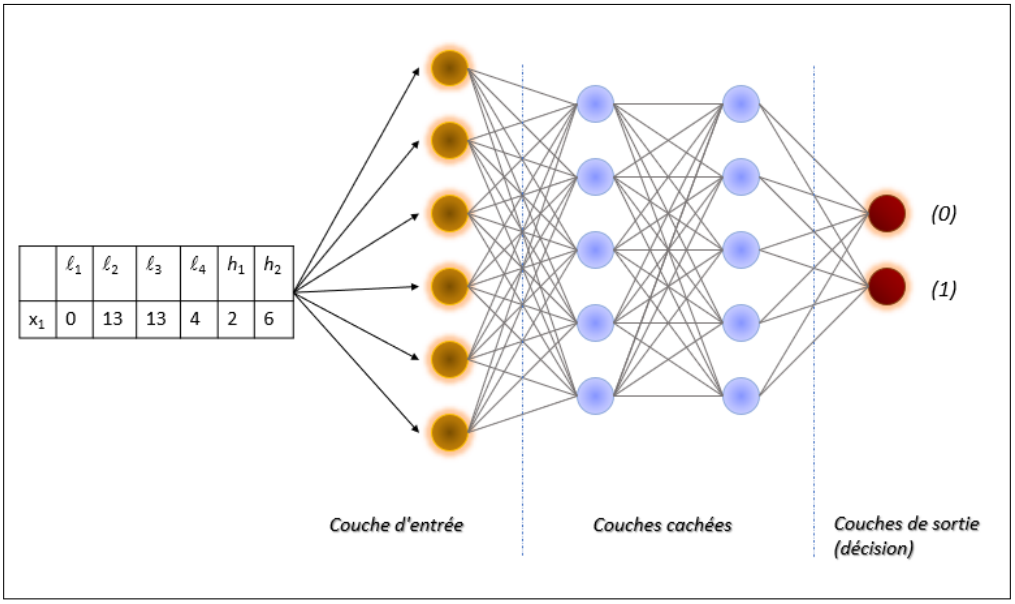

FIGURE A.1 – Réseau de neurones artificiels à deux couches, utilisé en tant que modèle de classification à deux classes. Le premier *bag* de la figure 2.6b illustre le vecteur d'attribut passé en entrée du réseau.

Lors du mécanisme de *forward propagation*, l'activation a_i de chaque neurone i est calculée en prenant en paramètre l'activation des neurones de la couche précédente. Pour la première couche cachée, il s'agira des valeurs du vecteur x de la couche d'entrée. La formule suivante définit le calcul de la fonction d'activation propre à un neurone i :

$$a_i = f(z) = f(\Sigma w_i x_i + \beta) \tag{A.1}$$

Avec x_i le vecteur de sortie des neurones de la couche précédente et w_i le vecteur de poids associé. La fonction f est une **fonction d'activation** qui ramène la valeur de sortie dans un intervalle de valeurs plus réduit. Parmi les fonctions d'activation possible, nous pouvons citer les fonctions suivantes :

Logistique (ou Sigmoïde) définie par : $s = \frac{1}{1+e^{-z}}$
Son intervalle de sortie est $\{0, 1\}$

Unité de rectification linéaire (*Rectified Linear Unit* - ReLU).
Avec un intervalle de sortie de $[0, +\infty]$, prend la valeur maximale parmi z et 0.

Au début de l'entraînement, les vecteurs de poids w sont initialisés de façon aléatoire. Les poids de chaque neurone i sont ensuite mis à jour en recherchant ceux qui permettent de minimiser l'erreur de classement δ_i évaluée par une **fonction de coût**. L'algorithme de rétro-propagation du gradient est appliqué afin de chercher à atteindre le minimum global de la fonction de coût. Concrètement, en ce faisant, les valeurs des poids sont modifiées de manière à approcher la valeur δ_i la plus faible possible.

Le mécanisme de rétro-propagation met d'abord à jour les poids des neurones dans la couche de sortie, puis propage le calcul jusqu'à la première couche cachée.

Lors de l'entraînement d'un réseau de neurones, l'erreur de classification est déterminée à la fin de chaque *epoch* par une fonction de coût.

A.2 Fonction de coût : *Binary Cross-Entropy* (BCE)

Pour les expérimentations réalisées dans le chapitre 4 nous avons choisi d'utiliser la fonction de coût dite en anglais ***binary cross-entropy with logit loss*** pour entropie croisée binaire avec fonction logistique.

Soit N le nombre d'éléments dans le *batch*, et p un poids attribué à la classe positive, le calcul de l'erreur de classement est donné par la formule suivante :

$$\ell = \frac{\Sigma_{i=1}^{N} \left([p \times y_i \times log\ \sigma(x_i) + (1 - y_i) \times log(1 - \sigma(x_i))] \right)}{N} \tag{A.2}$$

Annexe B

Latent Dirichlet Allocation - LDA

En supposant qu'un document d soit constitué d'un ensemble de mots, chacun d'entre eux étant attribué à l'un des thèmes constituant le document, l'objectif de la méthode est de rapprocher les documents en comparant leur similarité par thème. À l'initialisation de l'algorithme, on donne un nombre k de thèmes à déterminer et chaque mot w du document est associé aléatoirement à un thème t en suivant une distribution de Dirichlet.

La mise à jour de l'assignation s'effectue ensuite pour chaque document de la base de données.
Pour chaque document et chaque mot de chaque document, deux probabilités sont calculées :

— la probabilité $p(t|d)$ que le document d soit assigné au thème t (1)
— la probabilité $p(w|t)$ que le thème t, dans le corpus de documents, soit assigné au mot w (2)

Enfin, le nouveau thème attribué au document est choisi en fonction de la probabilité la plus élevée pour chaque thème, calculée avec la probabilité que le thème t génère le mot w dans le document d, donné par la formule suivante :

$$p(t|d) * p(w|t)$$

En répétant les étapes précédentes sur l'ensemble des documents un certain nombre de fois, les assignations se stabilisent.

Annexe C

Algorithmes de classification : description et méthode

Dans le cadre de la maintenance prédictive, plusieurs algorithmes de classification peuvent être appliqués pour prédire la survenue d'une défaillance (ou d'une erreur critique). Cette annexe présente le fonctionnement théorique du classifieur Bayésien Naïf et de l'Arbre de Décision, deux méthodes de classification issues du domaine des statistiques.

C.1 Classifieur Bayésien naïf

Dans le cadre d'une prédiction de défaillance, le modèle Bayésien s'appuie sur la distribution probabilité des différents événements précurseurs de la défaillance.

On note $P(D|A)$ la probabilité ***a posteriori*** d'observer un événement B sachant un événement A. Cette probabilité *a posteriori* est calculée à l'aide de la formule suivante : $P(D|A) = \frac{P(A|D)*P(D)}{P(A)}$

On note également les probabilités suivantes :

- $P(D|A)$ la probabilité *a posteriori* qu'une défaillance D survienne sachant l'occurrence de l'événement A,
- $P(A|D)$ la probabilité *a posteriori* que l'événement A survienne sachant que la défaillance D a été observée,
- $P(A)$ la probabilité *a priori* d'avoir l'événement A,
- $P(D)$ la probabilité *a priori* d'avoir l'événement D.

L'ensemble de ces probabilités est modélisée sur la base du théorème de Bayes, défini par :

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \tag{C.1}$$

Sachant un ensemble d'instances étiquetées positives (classe C_1) ou négatives (classe C_0). Dans le cadre d'une prédiction, le modèle Bayésien a pour but de classer les instances selon la propriété du **maximum *a posteriori***. En effet, pour chaque classe C_k, la probabilité *a posteriori* $P_p(C_k|x)$ que l'instance appartienne à la classe C_k est calculée, puis l'instance est classée en fonction de la probabilité la plus forte.

Voyons à présent comment appliquer concrètement ce théorème dans le cadre d'une classification binaire en tenant compte de plusieurs variables explicatives (ou attributs). Soient deux classes Y_0 et Y_1, on note x_j la valeur d'un attribut j pour l'instance x du jeu de données. La probabilité postérieure de classifier une instance x dans la classe Y_1 est alors définie par la formule suivante :

$$P_p(C_1|x) = \frac{P(C_1)P(x_0|C_1)P(x_1|C_1)P(x_j|C_1)}{P(C_1)P(x_0|C_1)P(x_1|C_1)P(x_j|C_1) + P(C_0)P(x_0|C)P(x_1|C_0)P(x_j|C_0)} \tag{C.2}$$

Pour chaque attribut du jeu d'entraînement, deux paramètres d'espérance μ_j et de variance σ_j^2 sont calculés. Soit $P(x_j|C_k)$ la probabilité *a posteriori* d'observer l'instance x_j sachant qu'elle appartient à la classe C_k donnée par la formule suivante :

$$P(x_j|C) = \frac{1}{\sqrt{2\pi \times \sigma_j^2}} \exp(\frac{-1}{2\sigma_j^2} \times (x - \mu_j)^2 \tag{C.3}$$

La classe C_k est finalement attribuée à l'instance en fonction de la probabilité $P(x_j|C)$ la plus forte.

C.2 Arbres de décision

L'arbre de décision (`AD`) est un modèle qui permet de classer les instances d'un jeu de données en fonction de la répartition des valeurs des variables explicatives et la contribution de chacune de ces variables à répartir les instances dans les classes.

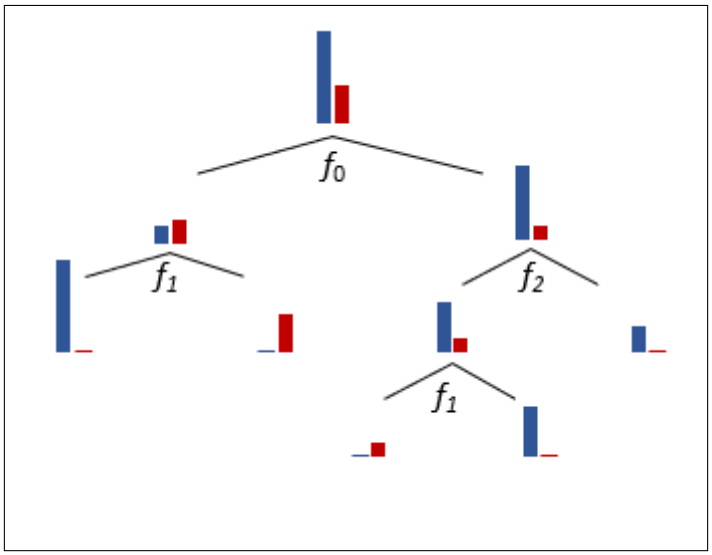

FIGURE C.1 – Arbre de décision dans le cas d'une classification binaire : classe 0 en bleu et classe 1 en rouge.

La figure C.1 illustre un AD de profondeur 3, appliqué sur des données comportant 3 attributs notés f_0, f_1 et f_2. À chaque nœud de l'arbre, un attribut est sélectionné et les instances sont réparties dans les deux branches qui naissent de ce nœud, en fonction de leur valeur pour l'attribut considéré. Dans chaque branche, la répartition des instances par classe est évaluée, comme le montrent les diagrammes en bâton sur la figure C.1. Plus la répartition obtenue dans chaque branche est homogène (dans l'idéal, les instances obtenues appartiennent toutes à la même classe), plus la contribution de l'attribut choisi est importante.

Pour évaluer la capacité d'un attribut à discriminer les classes, il existe plusieurs mesures, basées sur la distribution :

- **Gain d'information** : ce critère est basé sur la mesure de l'entropie des attributs, qui permet d'évaluer l'écart de la distribution de la variable à prédire par rapport à la distribution uniforme. L'attribut ayant l'entropie la plus faible est sélectionné.
- L'**Indice de Gini** : il permet d'évaluer l'inégalité de distribution dans une variable.
- Le **rapport de gain** [Qui86] est une version normalisée du gain d'information, qui permet de réduire le biais lié aux grands effectifs de certaines variables.

La création de nouveaux nœuds dans l'arbre cesse lorsque la répartition des instances est complètement homogène (chaque branche issue du nouveau nœud ne contient que des instances appartenant à la même classe). Cette situation arrive rarement en pratique,

surtout dans le cas de données complexes. Aussi, un choix de paramétrage de l'`AD` consiste à fixer la limite de profondeur de l'arbre en fonction de l'exigence souhaitée quant à la précision de classification et à la rapidité d'exécution.

L'exemple illustratif de la figure C.1, est associé à une situation de classification binaire. Les AD peuvent évidemment être utilisés pour des problèmes multiclasses, dans ce cas chaque nœud donne naissance à autant de branches qu'il existe de classes possibles.

Annexe D

Contributions scientifiques

Contributions internationales

① `Article de journal` - Myriam Lopez, Marie Beurton-Aimar, Gayo Diallo, Sofian Maabout. "*A simple yet effective approach for log based critical errors prediction*". Journal : Computers in Industry (2022).
② `Article de conférence` - Myriam Lopez, Marie Beurton-Aimar, Gayo Diallo, Sofian Maabout. "*Log data preparation for predicting critical errors occurrences*". Conférence : World Conference on Information Systems and Technologies - WorldCIST (2021)

Contributions locales

③ `Article de conférence` - Myriam Lopez, Marie Beurton-Aimar, Gayo Diallo, Sofian Maabout. "*Approche de traitement des logs pour la prédiction d'erreurs critiques*". Conférence : Extraction et Gestion de Données - EGC (2021).
④ `Poster` - Myriam Lopez. "*Prédiction d'erreurs critiques pour l'aide à la maintenance industrielle*". Conférence : Rencontre des Jeunes Chercheurs en Intelligence Artificielle - RJCIA (2021)
⑤ `Contribution orale` - Myriam Lopez. "*Prédiction d'erreur critique pour la maintenance prédictive.* Conférence : Dataquitaine (2021)

Liste des acronymes

PI	*Predictive Interval* (ang.) Intervalle Prédictif
RI	*Reactive Interval* (ang.) Intervalle de Réactivité
EI	*Error Interval* (ang.) Intervalle d'Erreur
VP	Vrai Positif
FP	Faux Positif
VN	Vrai Négatif
FN	Faux Négatif
MP	Maintenance Prédictive
ML	*Machine Learning* (ang.) Apprentissage Automatique
TM	*Topic Modeling* (ang.) Modélisation de thèmes
NPL	*Natural Langage Processing* (ang.) Traitement Automatique de la Langue
LDA	*Latent Dirichlet Allocation* (ang.) Allocation Latente de Dirichlet
SVM	*Support Vector Machine* (ang.) Séparateur à Vaste Marge
DNN	*Deep Neural Network* (ang.) Réseau de neurones artificiels
MIL	*Mulitple Instance Learning* (ang.) Apprentissage Multi-Instances
FPR	*False Positive Rate* (ang.) Taux de Faux Positifs
CPU	*Central Processor Unit* (ang.) microprocesseur de calcul
IoT	*Internet of Things* (ang.) Internet des Objets
RdP	Réseau de Petri
CIFRE	Convention Industrielle de Formation par la REcherche

Table des figures

Liste des tableaux