Optimisation automatique de l'apprentissage en ligne

Émile Royer

Émile Royer
Promotion 2024 — 5e année
Majeure ingénierie et numérique

Mémoire de projet de fin d'études

Optimisation automatique de l'apprentissage en ligne

Lieu du stage : Inria Paris (France)
Dates du stage : 4 mars 2024 – 30 aout 2024
Service où s'est déroulé le stage : équipe-projet Mimove
Nom de la tutrice professionnelle : Maroua Bahri
Nom de la tutrice pédagogique : Zehira Haddad

Remerciements

Je souhaite remercier les membres, actuels et passés, de l'équipe Mimove pour leur accueil et pour m'avoir intégré dans le quotidien de la recherche.

Je remercie aussi ma tutrice de stage, Maroua Bahri, qui m'a poussé pour donner le mieux.

Enfin, je pense aux personnes des équipes administratives, en particulier les assistantes administratives de l'équipe, Diana et Eugénie-Marie.

Résumé

Les méthodes d'optimisation de modèles d'apprentissage automatique en ligne utilisent exclusivement des algorithmes génétiques. Il est raisonnable de penser que des méthodes plus efficaces encore existent en faisant un parallèle avec les autres domaines de l'optimisation. Nous montrons qu'il est possible d'utiliser l'optimisation bayésienne pour l'optimisation en ligne, avec des résultats équivalents à l'état de l'art. Cette nouvelle technique a le potentiel de créer de nouveaux optimiseurs en ligne plus efficaces.

Mots-clés : apprentissage automatique, apprentissage automatique en ligne, automl, optimisation bayésienne

Abstract

Automated machine learning methods for online machine learning exclusively use genetic algorithms. Drawing a parallel with other fields of optimisation studies, it is reasonable to think that more efficient methods exist. We show it is possible to use bayesian optimisation for online autoML, with equivalent results to the state of the art. This new method has the potential to create new, more efficient online optimisers.

Keywords: machine learning, online machine learning, automl, bayesian optimisation

Table des matières

Avant-propos

Les acronymes anglais sont omniprésents et ont rarement un équivalent français universellement compris. C'est pourquoi ce rapport prend le parti d'utiliser le nom français des algorithmes quand il est dit en entier, mais de conserver l'acronyme anglais. On dira ainsi « forêt aléatoire adaptative », mais le terme sera abrégé en « ARF ».

Il a été choisi d'utiliser l'orthographe moderne du français adoptée par l'Académie française dans la réforme de 1990 ; l'écriture de certains mots est différente de l'ancienne orthographe sans pour autant être fautive.

1 Introduction

Ce rapport présente le travail réalisé pendant mon stage de fin d'études d'ingénieur, qui conclut cinq années de formation. Ces cinq ans ont été constitué en la découverte et l'apprentissage du métier d'ingénieur, et ce stage a été l'occasion de mettre en application et transcender ces compétences.

J'ai travaillé pendant six mois dans le centre parisien de l'institut national de recherche en informatique et en automatique (Inria), de mars à aout 2024.

C'est un stage de recherche et non pas d'application : en étant immergé dans la vie scientifique, le but est de produire des nouvelles connaissances et de les partager avec la communauté scientifique.

Ce rapport est structuré comme suit : d'abord, l'établissement accueillant ce stage sera présenté, avec son histoire et ses missions. Ensuite, le sujet de recherche de ce stage sera discuté et détaillant l'état de l'art et la nouvelle méthode proposée. Enfin, la conclusion synthétisera le rapport tant du côté technique que des apprentissages personnels.

2 Présentation de l'Inria

Ce stage s'est déroulé à l'Inria [1], une institution reconnue de la recherche française, et en particulier au centre de recherche de Paris.

2.1 Statut

L'institut national de recherche en informatique et en automatique (Inria) est un organisme public français de recherche en sciences de l'information [2]. Dans sa communication, il se définit comme « l'institut de recherche en sciences du numérique ». C'est un établissement public à caractère scientifique et technologique (EPST), au même titre, par exemple, que le CNRS ou l'Inserm.

FIGURE 2.1 – Le logo actuel de l'Inria

Ses missions sont de conduire des recherches en informatique et en mathématiques appliquées et de participer au transfert technologique en direction des entreprises françaises.

L'Inria a été pensé dès l'origine pour avoir une structure plus horizontale, différente des autres institutions de recherche ; au lieu d'être structuré en gros laboratoires, il est séparé en équipes-projet plus petites et avec un sujet plus précis, qui se recomposent régulièrement.

2.2 Histoire

L'histoire de l'Inria commence à l'adolescence de l'informatique. Pendant les années 1960, la France est dans un programme de réindustrialisation, selon la vision du président de la République Charles de Gaulle [3].

En parallèle, l'industrie informatique est en expansion depuis la fin de la Seconde Guerre Mondiale, portée surtout par les États-Unis d'Amérique. Les innovations sur les infrastructures de calcul, comme l'invention du transistor en 1947, font que l'écosystème est en constante mutation. L'étatsunien IBM domine largement le secteur à partir des années 1950.

1. https://inria.fr
2. Le nom « Inria » s'accorde usuellement au masculin, sur le calque d'« institut ».
3. Cette partie se base essentiellement sur le livre « Histoire d'un pionnier de l'informatique » de BELTRAN et GRISET [6].

En 1964, l'entreprise Bull, seul fabricant français d'ordinateurs, a des difficultés économiques. La seule issue possible est un rachat par l'entreprise étasunienne General Electric. Cette vente est vue comme une humiliation par le gouvernement français, qui perd ses capacités de production nationale alors que ses besoins sont croissants. Le refus des États-Unis de vendre des ordinateurs au CEA aggrave la tension.

Soucieux de développer des capacités de production nationale, le gouvernement français lance en 1966 le plan Calcul pour développer l'industrie informatique française.

Début 1967, l'Iria est créé pour concentrer la recherche en informatique et faciliter les transferts de technologie avec l'industrie[2], initialement sous la tutelle du ministère de l'industrie. L'ambition est un institut de niveau national : ne pas se contenter d'un laboratoire en région parisienne, mais s'installer progressivement en de nombreux endroits du territoire. Jusqu'alors la recherche en informatique s'était effectuée, et continuera, dans des institutions généralistes (comme le CNRS et les universités).

Dès sa création, l'Iria a une organisation différente des instituts existant, plus horizontale. Alors que les institutions de recherche étaient habituellement divisées en laboratoires, avec un grand nombre de personnes et une certaine hiérarchie, l'Iria est organisé en équipes-projet, plus petites et flexibles.

Installé sur l'ancien quartier général de l'Otan à Rocquencourt (Yvelinnes), l'institut s'essaime et crée au fil des années des centres dans d'autres régions (Rennes à partir de 1972, Sophia-Antipolis en 1980, sans compter les petits essaimages antérieurs), conformément à la vision de ses créateurs.

L'Iria mène le projet Cyclades pendant les années 1970, un des premiers réseaux entre ordinateurs. Son interconnexion avec Arpanet (aux États-Unis) a donné naissance à Internet.

Fin 1979, L'Iria gagne en importance et en lettres : il devient l'Inria, institut national[3].

2.3 De nos jours

L'Inria est dirigé par Bruno Sportisse depuis 2018. Il compte à présent une dizaine de centres dans toute la France, 220 équipes et 3800 chercheuses et chercheurs.

L'Inria est reconnu pour ses productions académiques tant en France qu'à l'international. Le Haut Conseil de l'évaluation de la recherche et de l'enseignement supérieur (Hcéres), qui contrôle les établissements publics de recherche scientifiques, souligne l'expertise mondiale de l'Inria dans des domaines de pointe comme l'intelligence artificielle[24].

L'institut a vu naitre dans ses portes des projets scientifiques majeurs : parmi les réalisations actuelles on peut citer le langage de programmation *OCaml*, la bibliothèque pour l'apprentissage automatique *scikit-learn* ou la méthode des Gaussiennes 3D.

Le centre de recherche de Paris, dans lequel s'est déroulé ce stage, est situé dans le 13^{e} arrondissement de Paris, au 48 de la rue Barrault depuis juillet 2024.

Avant son déménagement, il était localisé rue Simone If, dans le 12^{e} arrondissement.

2.4 Équipe-projet

Ce stage a été fait au sein de l'équipe-projet Mimove [4]. Ses thèmes de recherche sont les systèmes distribués et leurs intergiciels (« middleware »).

L'équipe est dirigée par Nikolaos Georgantas et a été formée en 2014, accueillant les membres de l'ancienne équipe « Arles ».

De nos jours, Mimove est une petite équipe : elle compte seulement un chercheur permanent et en tout six personnes au début de ce stage (sans compter le stagiaire).

4. https://www.inria.fr/fr/mimove

3 Projet d'étude

Le sujet de stage est « Automatisation de l'apprentissage automatique dans les contextes par lots et en ligne », ou en anglais « Automated machine learning for batch and stream settings ». C'est un sujet vaste et volontairement vague ; une précision du domaine est nécessaire. Il fut décidé de travailler sur l'optimisation de l'apprentissage automatique (AutoML) en ligne.

Ce domaine de recherche de ce stage n'est pas jeune — on peut trouver des recherches s'y rattachant dès le début des années 2000[5] —, mais il connait une accélération depuis le début des années 2020.

3.1 Apprentissage automatique

L'apprentissage automatique est une branche de l'intelligence artificielle. En tant que tel, son but est de créer des programmes exhibant des comportements qu'on pourrait croire nécessitent une intelligence, a fortiori une intelligence humaine.

La classification comme intelligence artificielle évolue avec le temps : la recherche du plus court chemin dans un graphe a pu être considérée comme de l'intelligence artificielle par le passé, elle est aujourd'hui considérée volontiers comme une catégorie d'algorithmes classiques.

Le principe de l'apprentissage automatique est de dériver le comportement des programmes non pas d'instructions données explicitement par les programmeuses et programmeurs, mais plutôt de le construire à partir d'un jeu de données. Cette capacité à prédire la forme des données est puissante, à tel point que l'essentiel de la recherche en intelligence artificielle depuis vingt ans se fait sur l'apprentissage automatique.

Ces qualifications d'intelligence et d'apprentissage ne doivent cependant pas faire croire que ces algorithmes sont équivalents aux capacités humaines ou animales. Leur apprentissage n'est pas le même que les humains, il n'y a pas de compréhension du sujet, seulement une description statistique. Ces programmes n'ont pas non plus de processus conscient.

Un programme d'apprentissage automatique utilise une description statistique de ses données d'entrainement pour prédire un résultat. Ce résultat reproduira les caractéristiques statistiques apprises, aussi bien en bien qu'en mal : les biais dans les données initiales resurgiront dans les prédictions.

L'apprentissage automatique est utilisé pour trois type de tâches :

— la classification : déterminer à quelle catégorie appartient une observation ;
— la régression : donner la valeur d'une variable continue au point demandé ;
— le regroupement de points : exploiter la proximité entre les points pour faire ressortir des groupes.

Il existe quatre méthodes pour entrainer un modèle d'apprentissage automatique, selon les données disponibles :

— supervisé : les données et les réponses attendues sont données ;
— non-supervisé : seules les données sont données, sans la réponse attendue ;
— semi-supervisé : une partie des réponses attendue est donnée ;
— par renforcement : le modèle est autonome et doit améliorer un score mesurant la tâche à accomplir.

L'apprentissage supervisé est le plus simple conceptuellement, c'est celui qui sera décrit dans la suite de cette section.

Prenons un exemple de classification par apprentissage automatique supervisé ; on imagine un modèle très simple qui répond la classe la plus fréquente pour les données.

On lui montre les données suivantes : (Marianne, 1), (Marianne, 1), (Soleil, 2), (Soleil, 3), (Marianne, 2), (Soleil, 2), (Marianne, 1).

Avec l'entrée « Marianne », le programme d'apprentissage répondra « 1 ». Le modèle pourra ignorer de lui-même les données avec peu d'occurrences, comme « Marriane, 2 ».

3.2 Apprentissage en ligne

L'approche classique de l'apprentissage automatique, décrite ci-dessus, est aussi appelée par lots : toutes les données sont disponibles simultanément et sont traitées par morceau.

Les modèles doivent être ré-entrainés du début s'ils doivent être mis à jour. Le pire des cas se présente avec le travail sur des flux de données changeants où le modèle doit être mis à jour après chaque information reçue : les ressources nécessaires ont une croissance exponentielle. Comment les mettre à jour plus efficacement sur des flux de données ?

Ce concept définit l'apprentissage en ligne.

Les données sont séparées en observations individuelles, des instances. Une instance correspond généralement à une entrée d'un jeu de données.

Ainsi, l'apprentissage en ligne consiste à apprendre sur un flux de données : au lieu de d'entrainer un modèle une fois et de le garder tel quel, l'apprentissage se fait en continu.

On ré-entraine le modèle dès que des nouvelles données arrivent. L'évaluation d'un modèle peut se faire de différentes façons, mais la plus courante est le test-puis-entrainement (*test-then-train*) : quand une instance est reçue, on fait prédire le modèle dessus pour connaitre son erreur, puis le modèle est entrainé avec cette nouvelle donnée. Ceci assure que le modèle est évalué sur des données qu'il n'a pas encore vues tout en pouvant faire une évaluation en continu.

Les objectifs de l'apprentissage en ligne sont[13, chapitre 2] :

1. Ne traiter qu'une instance à la fois, et l'inspecter au plus une fois ;
2. N'utiliser qu'un temps limité pour traiter chaque instance ;
3. N'utiliser qu'une quantité limitée de mémoire ;
4. Être prêt à prédire à tout moment ;
5. S'adapter aux changements au cours du temps.

Les modèles d'apprentissage en ligne sont nécessairement différents de ceux par lots pour pouvoir s'approcher des objectifs ci-dessus, ils doivent être faits sur-mesure. Un équivalent de chaque type de modèle a ainsi été créé : les arbres d'Hoeffding (HT)[4] pour les arbres de décision ; la forêt aléatoire adaptative (ARF)[12, 14] pour la forêt aléatoire ; mais aussi une

adaptation simple des k plus proches voisins (kNN), de Naive Bayes, des perceptrons ou de la régression linéaire.

La classification est la tâche la plus étudiée, mais même là on voit la différence avec l'apprentissage par lots : on est loin d'avoir la même richesse d'algorithmes.

Pour s'adapter aux changements, il est nécessaire de déterminer si la distribution portée par le flux de données a changé : il a été créé des détecteurs de changement, qui permettent de prévenir un une dérive est détectée pour adapter l'entrainement du modèle.

3.3 Optimisation des hyperparamètres

Les modèles complexes ont de nombreux hyperparamètres, qui permettent de changer le comportement du modèle. Trouver les bons hyperparamètres est essentiel : une bonne combinaison permet une bonne adaptation au problème, alors que de mauvais peuvent rendre le modèle inutilisable.

Choisir les hyperparamètres est long et demande des connaissances sur le fonctionnement du modèle. Mais exécuter des programmes et comparer leurs résultats est quelque chose que les ordinateurs savent très bien faire.

C'est le but de l'apprentissage automatique automatisé (AutoML, pour *automated machine learning*).

Plusieurs méthodes de recherche existent, des plus naïves (recherche en grille , recherche aléatoire) aux plus complexes (algorithme génétique, recherche basée sur un modèle).

Parmi les méthodes naïves, la recherche aléatoire fonctionne étonnamment bien : elle explore efficacement le domaine de recherche, sans a priori. Sous réserve de donner du temps, une bonne configuration d'hyperparamètres sera toujours trouvée.

Les méthodes complexes fonctionnent assez évidement mieux : la recherche génétique permet de diriger la recherche aléatoire en direction des meilleurs points connus pour les améliorer. Les meilleurs optimiseurs toutefois sont souvent ceux basés sur un modèle : on peut citer la configuration séquentielle d'algorithme basée sur un modèle (SMAC)[9] et les estimateurs de Parzen en forme d'arbres (TPE)[8].

Le problème à résoudre est formalisé pour l'apprentissage supervisé sous le nom du problème *combined algorithm selection and hyperparameter* (CASH)[10] : il s'agit de trouver la combinaison d'algorithme d'apprentissage et d'hyperparamètres qui minimise l'erreur de prédiction.

3.4 Optimisation bayésienne

L'optimisation bayésienne (OB) est une technique d'optimisation itérative qui utilise un modèle interne pour approximer la fonction cible à optimiser.

L'OB est un domaine complexe et l'objectif de ce document n'est pas d'expliquer son fonctionnement en détail. On redirigera vers des ressources comme le livre « Bayesian optimization »[23] pour en savoir plus. Cette section donne un aperçu avec certaines notions utiles pour comprendre le principe du sujet d'étude.

L'OB est adaptée pour les cas où obtenir des nouveaux points à évaluer est couteux : elle arrive à donner de bonnes estimations de l'optimum global d'une fonction avec peu de cycles et

peu d'évaluation de la fonction cible. Une implémentation typique utilise les processus gaussiens car cette famille de régresseurs a beaucoup de propriétés utiles, mais elle n'y est pas limitée.

On crée un modèle interne [1], et on l'entraine avec les entrées et sorties de la fonction cible — quelle entrée donne quelle sortie. On utilise ensuite une autre fonction, la fonction d'activation, pour décider du point optimal à évaluer ensuite. Le modèle interne est alors entrainé à nouveau avec le résultat du nouveau point.

La fonction d'acquisition est clé ici ; elle décide des points à évaluer en fonction de leur valeur prédite, mais aussi de l'incertitude du modèle interne à propos de cette prédiction. Le prochain point à évaluer sera un compromis entre la plus grande incertitude (pour améliorer la précision du modèle) et la meilleure valeur prédite (pour améliorer l'estimation de l'optimum).

3.4.1 SMAC

Sequential model-based algorithm configuration (SMAC) est un algorithme d'AutoML pour l'apprentissage par lots créé en 2011 par Frank Hutter, Holger Hoos et Kevin Leyton-Brown[9], puis mis à jour avec des variantes en 2017[20]. Il est basé sur l'optimisation bayésienne.

Grâce à son efficacité, il est devenu l'une des références de l'autoML, et est utilisé dans un grand nombre de systèmes comme auto-scikit-learn.

Sa force vient en partie du fait d'être basé sur l'optimisation bayésienne : il est couteux d'entrainer un modèle et l'optimiseur se doit de converger rapidement tout en équilibrant le rapport entre amélioration des points connus et exploration de nouvelles zones. L'optimisation bayésienne remplit bien ces critères.

SMAC utilise une forêt aléatoire d'arbres de décision : les arbres sont plus flexibles que les processus gaussiens, au prix d'une perte de précision. Les processus gaussiens ne permettent de traiter que des données numériques continues, les arbres peuvent traiter d'autres types de données, comme les données catégorielles.

En revanche, les forêts aléatoires n'ont pas de mécanisme d'estimation de l'incertitude. Pour en émuler un, SMAC utilise la variance entre les prédictions des arbres individuels : quand on prédit la valeur d'un point, on prédit son incertitude aussi.

Pour la même raison de son choix de modèle interne, la méthode d'optimisation de la fonction d'acquisition (pour chercher des nouvelles configurations à évaluer) doit être modifiée. En effet, un arbre de décision produit une surface discrète, composée de segments constants reliées par des discontinuités. Il n'est donc pas possible d'utiliser de méthode d'optimisation classique qui suppose la continuité de la surface, comme la descente de gradient.

À la place, SMAC utilise une combinaison de recherche par voisinage et de recherche aléatoire. Premièrement, on part des 10 meilleurs points connus empiriquement, et on trouve récursivement leur meilleur voisin prédit. Deuxièmement, on tire 10 000 points aléatoirement. Finalement, on trie les 10 010 point obtenus (par voisinage et aléatoirement) selon leur score donné par la fonction d'acquisition : les meilleurs candidats sont premiers. Pour assurer la diversité des points même si la fonction d'acquisition est biaisée, on intercale des points aléatoires.

Cette liste de 20 000 points sera évaluée un par un ; si l'une des ces configurations a une meilleure performance que la titulaire elle la remplace et un nouveau cycle commence avec une nouvelle liste.

1. en anglais « surrogate model »

3.5 État de l'art

Il existe quelques algorithmes d'optimisation des hyperparamètres en ligne qui proposent une solution pour le problème CASH.

ConfStream[16] utilise une distribution normale tronquée pour créer des mutations à partir de la meilleure configuration trouvée.

Champion-Challengers (ChaCha)[18] utilise un oracle pour choisir des compétiteurs à partir d'un champion puis programme des affrontements. Il utilise un optimiseur hors-ligne.

EvoAutoML[19] applique des mutations à sa meilleure configuration avec une distribution uniforme, puis utilise une méthode de renforcement par ensemble pour entrainer les modèles.

AutoClass[21] est un mélange de confStream et EvoAutoML. Sa particularité est d'ajouter un filtre avant d'accepter un modèle à l'évaluation : il faut qu'un modèle interne estime que la nouvelle configuration sera assez performante.

OAML[22] est une méthode utilisant un optimiseur hors-ligne pour s'adapter aux dérives de distribution.

ASML[25], le plus récent, remplace toutes les configurations au début d'un nouvelle fenêtre et ne conserve que la meilleure. De nouvelles configuration sont générées à partir de cette dernière et d'autres sont tirées aléatoirement.

Tous ci-dessus sont basés sur l'exploration aléatoire ou génétique.

À noter *Single-pass Self Parameter Tuning*[17] qui n'utilise pas de recherche aléatoire mais l'algorithme de Nelder-Mead[1]. À son désavantage, cette méthode ne traite que le problème de l'optimisation des hyperparamètres, elle n'est ainsi pas comptée dans la même catégorie.

ChaCha et OAML ne sont pas totalement en ligne : il utilisent des optimiseurs par lots et utilisent des algorithmes en ligne entre deux optimisations. Leur proposition tient plus de la conception de pipeline que d'optimiseur.

3.6 Méthode proposée

Ayant vu les bonnes performances de SMAC dans le contexte hors-ligne, il a été décidé pour ce travail de créer un dérivé de SMAC qui fonctionne en ligne. L'algorithme est ainsi baptisé « OnlineSMAC », abrégé en OSMAC.

Il a fallu adapter SMAC pour répondre aux contraintes de l'apprentissage en ligne. En particulier, utiliser une seule fois chaque instance ; un des éléments puissants de SMAC est son utilisation de plusieurs passes pour choisir la meilleure configuration.

L'architecture d'OSMAC est similaire à celle d' ASML : à chaque cycle, toutes les configuration évaluées sont mises de côté et de nouvelles sont tirées. La meilleure passée sert de référence. On utilise une grande partie du principe de SMAC : la méthode de sélection des nouvelles configurations, et la fonction d'acquisition sont les mêmes.

Une adaptation a dû être faite sur le modèle interne : il faut que ce soit aussi un modèle en ligne. Il a été choisi d'utiliser l'équivalent de la forêt aléatoire, l'ARF en mode régression. Parce que nous avons constaté que l'estimation de l'incertitude avec l'écart-type empirique des prédictions donne de mauvais résultats, il a été décidé de changer cet aspect pour utiliser la méthode du Jackknife[11] en ligne.

```
1 : procedure OnlineSMAC(S, B, p)
2 :     ms ← InitializeModels(S, B)
3 :     i ← randomConfiguration(S)
4 :     M ← CreateSurrogateModel()                    ▷ Modèle interne
5 :     t ← 0
6 :     repeat                                        ▷ Phase d'entraînement
7 :         x, y ← GetNextInstance()
8 :         Train(i, x, y)                            ▷ Entraînement du titulaire
9 :         for all m ∈ ms do
10 :            Train(m, x, y)                        ▷ Entraînement des modèles évalués
11 :        t ← t + 1
12 :        if t ≡ 0 (mod p) then                     ▷ Début d'un nouveau cycle
13 :            Train(M, ms)                          ▷ Entraînement du modèle interne
14 :            i ← max ms
15 :            ms ← SelectNew(S, B, M, i)            ▷ Sélection de nouvelles configurations
16 :    until stream ends
```

FIGURE 3.1 – Entraînement de OnlineSMAC

La plus grosse différence tient dans la comparaison des modèles évalués : pour traiter les données en une seule passe, le meilleur modèle parmi ceux évalués devient le nouveau modèle titulaire. Le meilleur modèle ne changeant pas à chaque cycle, la configuration du modèle titulaire est ajoutée aux configurations à évaluer : ceci permet une évaluation équitable entre tous qui reçoivent le même entrainement.

Enfin, pour ne pas perdre d'historique d'entrainement, si la meilleure configuration après l'entrainement est la même que celle du modèle titulaire, on conserve l'ancien modèle sans le remplacer.

3.6.1 Algorithme

Cette implémentation de OSMAC (figure 3.1) a trois hyperparamètres : l'espace de recherche (S), le budget (combien de modèles à évaluer en même temps, B) et la longueur d'une fenêtre d'évaluation (p, pour période).

On commence par initialiser les différents éléments (tirage initial des modèles évalués et création du modèle interne), aux lignes 2 à 5.

Le cycle d'optimisation commence ensuite, et se répète tant que des éléments arrivent dans le flux de données.

D'abord, on entraine le modèle titulaire (noté i pour *incumbent*) et des modèles ms évalués sur la nouvelle instance de donnée (lignes 8 à 10).

Ensuite, on vérifie si la fin de la période d'évaluation est atteinte : dans ce cas, on entraine le modèle interne avec les résultats de l'évaluation, on sélectionne un nouveau modèle titulaire et choisit de nouvelles configurations à évaluer (lignes 12 à 15). Et le cycle recommence.

TABLE 3.1 – hyperparameters search space

Modèle	Paramètre	Type	Domaine
kNN	neighbors	entier	$[3, 8]$
	weighted	booléen	True, False
	window size	entier	[100, 2000]
	p	entier	[1, 5]
SoftMax	l2	réel	$[0, 10^{-1}]$
StdScaler	with std. dev.	booléen	True, False
HT, HAT	max. depth	entier	[10, 100]
	grace period	entier	[50, 500]
	delta	réel	$[10^{-9}, 10^{-2}]$
	tau	réel	$[2 \cdot 10^{-2}, 8 \cdot 10^{-2}]$
	threshold	entier	[0, 50]
	split criterion	nominal	info. gain, Gini, Hellinger
	prediction type	nominal	MC, NB, NBA
ARF	nb. models	entier	[3, 10]
	max. depth	entier	[10, 100]
	grace period	entier	[50, 500]
	delta	réel	$[10^{-9}, 10^{-2}]$
	tau	réel	$[2 \cdot 10^{-2}, 8 \cdot 10^{-2}]$
	threshold	entier	[0, 50]
	split criterion	nominal	info. gain, Gini, Hellinger
	prediction type	nominal	MC, NB, NBA

3.6.2 Implémentation

L'algorithme a été implémenté en Python 3, avec la bibliothèque River 0.21.

Python est loin d'être le langage le plus rapide à l'exécution, mais il a quelques avantages de son côté dans ce cas. C'est un langage qui offre une grande facilité de développement, des concepts de haut niveau d'abstraction sont facilement utilisables. Ensuite, la présence de River, une bibliothèque reconnue pour l'apprentissage en ligne, offre toutes les briques nécessaires pour créer ce type de modèle. Enfin, beaucoup des optimiseurs de l'état de l'art sont aussi écrits en Python, il plus facile de comparer la consommation de ressources quand le même langage est utilisé.

Le tableau 3.1 montre les hyperparamètres qui pourront être sélectionnés pendant l'optimisation.

3.6.3 Expérimentations

Pour comparer OSMAC aux autres optimiseurs, nous les avons exécutés sur des jeux de données communément utilisés. Le tableau 3.2 liste les jeux de données utilisés en donnant le type de données (réel ou synthétique), le nombre d'instances comportées, le nombre de

caractéristiques par instance, le nombre de classes total ainsi que la proportion d'instances représentant la classe majoritaire et la classe minoritaire dans le jeu de données.

Tous ces jeux de données ont pour objectif la classification ; c'est la tâche la plus couramment représentée en apprentissage automatique en ligne et celles avec le plus de modèles dédiés.

Avoir plus de jeux de données à tester permet de réduire des effets de chacun d'entre eux et de donner une meilleure puissance statistique au résultat.

Des variantes des jeux de données synthétiques sont créées pour observer l'effet de la dérive conceptuelle. Un qualificatif est ajouté au nom pour indiquer le type de dérive injecté.

On ajoute aux test les modèles ARF, *streaming random patches* (SRP)[15] et *leveraging bagging* (LB)[7] : il sont aussi composites (ils utilisent un ensemble) et ont de très bonnes performances.

Protocole

Pour prévenir des fluctuations aléatoires et réduire les biais tout en assurant la reproductibilité des expérimentations, les algorithmes avec une composante aléatoire sont exécutés cinq fois sur chaque jeu de données, avec à chaque fois une graine différente pour leur générateur de nombres pseudo-aléatoires. Les graines utilisées ont été tirées par un générateur de nombre aléatoires dans l'intervalle $[0; 1000]$, et sont les suivantes : 760, 10, 490, 158 et 36.

Quand une classe peut être configurée avec des arguments, tous les champs possibles prennent leur valeur par défaut. La seule exception étant la graine du générateur d'aléatoire, qui a été spécifiée comme décrit précédemment.

Les calculs ont été effectués sur un serveur en temps partagé fonctionnant sous AlmaLinux 9 équipé d'un processeur Intel Xeon 5218 (Cascade Lake). Chaque modèle s'est vu attribuer deux cœurs et 10 Go de RAM ; aucun des modèles n'a atteint cette limite de mémoire.

Résultats

Pour la performance de classification (tableau 3.3), OSMAC arrive juste derrière ASML selon la moyenne, mais à la quatrième place selon la médiane et le classement moyen [2].

ASML arrive bon premier, avec quelques points d'avance, mais sa différence performance par rapport à OSMAC est non-significative.

Pour la mémoire (tableau 3.4), OSMAC arrive premier en moyenne grâce à sa régularité : sur aucun jeu de données sa mémoire utilisée ne dépasse les 20 Mo. Cependant, ASML arrive à nouveau premier en médiane et en rang moyen grâce à de nombreux très bons scores.

OSMAC ne peut rien contre la rapidité de ARF sur l'analyse du temps d'exécution (tableau 3.5). S'il se défend sur la moyenne, OSMAC arrive dernier sur les autres métriques.

3.7 Gestion de projet

L'objectif du stage était de créer un algorithme d'optimisation pour l'apprentissage automatique en ligne. Secondairement, il était souhaité de publier ces résultats dans une revue.

L'algorithme a bien été conçu ; il a été présenté plus haut. Malheureusement, la rédaction de l'article n'a pas pu être finie et la publication n'a pas eu lieu avant la fin du stage. L'écriture

2. moyenne du classement sur chacun des jeux de données

Table 3.2 – Jeux de données d'évaluation

Jeu de données	Type	Instances	Caractéristiques	Classes	Maj.	Min.
Bananas	real	5300	2	2	55.2%	44.8%
Covertype	real	581012	54	7	36.5%	0.5%
Electricity	real	45312	8	2	57.5%	42.5%
Image Segments	real	2310	18	7	14.3%	14.3%
Keystroke	real	20400	31	51	2.0%	2.0%
SensItVehicle	real	98528	100	3	50.0%	23.2%
Sensors	real	2219803	5	55	3.0%	0.1%
Agrawal	synthetic	1000000	40	2	67.2%	32.8%
Agrawal Abrupt	synthetic	1000000	40	2	52.8%	47.2%
Agrawal Gradual	synthetic	1000000	40	2	52.8%	47.2%
Hyperplane	synthetic	1000000	10	2	50.1%	49.9%
Hyperplane Fast	synthetic	1000000	10	2	50.0%	50.0%
Hyperplane Slow	synthetic	1000000	10	2	50.0%	50.0%
Led	synthetic	1000000	24	10	10.0%	10.0%
Led Abrupt	synthetic	1000000	24	10	10.1%	9.9%
Led Gradual	synthetic	1000000	24	10	10.1%	9.9%
Mixed	synthetic	1000000	4	2	50.0%	50.0%
Mixed Abrupt	synthetic	1000000	4	2	50.0%	50.0%
Mixed Gradual	synthetic	1000000	4	2	50.0%	50.0%
Random Tree	synthetic	1000000	30	2	57.8%	42.2%
Rbf	synthetic	1000000	10	5	30.0%	9.3%
Rbf Fast	synthetic	1000000	10	5	30.0%	9.3%
Rbf Slow	synthetic	1000000	10	5	30.0%	9.3%
Sea	synthetic	1000000	3	2	64.3%	35.7%
Sea Abrupt	synthetic	1000000	3	2	59.9%	40.1%
Sea Gradual	synthetic	1000000	3	2	59.9%	40.1%
Sine	synthetic	1000000	4	2	50.0%	50.0%
Sine Abrupt	synthetic	1000000	4	2	50.0%	50.0%
Sine Gradual	synthetic	1000000	4	2	50.0%	50.0%
Waveform	synthetic	1000000	21	3	33.4%	33.3%
Waveform Abrupt	synthetic	1000000	21	3	33.4%	33.2%
Waveform Gradual	synthetic	1000000	21	3	33.4%	33.2%

TABLE 3.3 – Performance de prédiction, en pourcent

	OSMAC	OSMACEns	ASML	AC	ARF	SRP	LB
Bananas	84,17	86,47	82,77	86,11	**88,40**	87,24	83,03
Covertype	95,58	95,83	**95,88**	95,71	86,71	93,06	87,71
Electricity	86,32	89,00	89,07	**90,18**	88,29	87,91	89,44
Image Segments	78,55	75,18	80,29	**86,69**	80,87	76,44	77,90
Keystroke	92,56	95,97	98,11	**98,54**	97,67	95,60	94,99
SensItVehicle	73,05	72,74	75,20	74,40	77,94	**79,51**	78,23
Sensors	95,35	**95,76**	94,98	76,34	79,41	73,94	68,68
Agrawal	88,40	87,37	94,37	94,08	91,11	92,53	**94,79**
AgrawalAbrupt	78,27	78,22	89,06	83,23	83,90	81,29	**92,30**
AgrawalGradual	74,43	75,52	86,03	82,39	81,14	78,72	**87,90**
Hyperplane	92,33	91,96	93,88	**94,13**	85,50	81,46	90,63
HyperplaneFast	92,09	91,78	**92,50**	89,18	84,02	81,91	87,60
HyperplaneSlow	92,12	91,78	**92,51**	89,21	84,02	81,91	87,60
Led	73,38	73,20	73,91	**73,96**	73,16	71,91	73,89
LedAbrupt	73,46	73,34	**73,99**	70,72	72,98	72,48	73,73
LedGradual	72,48	72,33	**73,12**	69,67	72,07	71,52	72,83
Mixed	98,21	98,21	98,94	97,62	98,08	88,58	**99,64**
MixedAbrupt	98,13	98,14	**98,82**	96,98	98,13	88,12	98,81
MixedGradual	96,27	96,29	97,01	95,34	96,26	84,70	**97,72**
RandomTree	77,01	77,17	82,19	84,85	81,78	85,93	**95,22**
Rbf	91,90	**91,94**	91,41	87,61	72,71	85,42	91,04
RbfFast	86,82	88,27	**88,28**	87,00	65,97	69,00	57,35
RbfSlow	92,05	**92,14**	91,48	88,08	80,79	82,00	83,58
Sea	87,33	87,17	87,57	88,56	89,76	89,61	**89,84**
SeaAbrupt	87,21	87,13	87,76	88,04	89,38	86,27	**89,40**
SeaGradual	86,94	86,82	87,47	87,76	**89,04**	84,34	88,80
Sine	97,79	97,60	98,21	98,44	99,07	89,52	**99,78**
SineAbrupt	96,46	96,35	97,39	96,14	98,46	87,53	**99,03**
SineGradual	91,69	91,59	93,24	86,83	93,36	77,65	**94,65**
Waveform	85,18	85,13	**85,60**	80,58	83,62	83,99	85,41
WaveformAbrupt	85,13	85,05	**85,58**	80,54	83,41	83,89	85,18
WaveformGradual	84,91	84,83	**85,34**	80,39	83,35	83,83	85,09
Average	87,05	87,20	**88,81**	86,85	85,32	83,06	86,93
Median	87,27	87,82	**89,07**	87,69	84,02	83,94	88,35
Avg. rank	4,28	4,28	**2,47**	4,16	4,34	5,56	2,91

Table 3.4 – Mémoire utilisée, en mégaoctets

	OSMAC	OSMACEns	ASML	AC	ARF	SRP	LB
Bananas	**2,03**	3,08	2,36	5,23	13,01	11,43	3,64
Covertype	5,54	7,24	13,68	4,72	**2,79**	41,81	22,40
Electricity	**4,24**	10,03	5,95	8,89	8,05	84,78	5,51
Image Segments	5,54	6,62	5,28	5,92	**4,06**	9,47	4,46
Keystroke	3,62	4,25	3,99	2,84	**1,23**	49,20	10,61
SensItVehicle	14,56	15,14	**6,98**	123,00	265,47	2 074,96	980,70
Sensors	12,77	12,14	4,55	3,59	**1,14**	4,04	5,67
Agrawal	**15,04**	18,36	214,36	372,69	463,85	1 763,62	1 045,80
AgrawalAbrupt	**5,74**	6,24	249,11	374,79	354,73	2 882,93	1 116,82
AgrawalGradual	6,18	**5,54**	144,97	705,25	255,03	2 381,08	1 059,09
Hyperplane	11,32	13,35	**0,18**	1,99	209,26	993,84	945,42
HyperplaneFast	10,37	14,69	**0,17**	4,15	378,86	371,69	111,74
HyperplaneSlow	10,61	14,69	**0,17**	4,03	378,86	371,69	111,74
Led	11,46	17,31	**1,34**	127,73	112,28	1 055,73	1 019,74
LedAbrupt	13,30	16,61	**1,03**	256,40	82,70	1 165,77	586,68
LedGradual	9,49	14,42	**0,88**	295,88	72,67	1 177,76	521,49
Mixed	11,99	13,01	**3,88**	15,65	35,36	382,28	84,83
MixedAbrupt	11,55	13,65	**4,17**	15,37	29,44	409,86	86,79
MixedGradual	6,94	8,95	**3,97**	18,81	27,37	664,91	117,30
RandomTree	**13,63**	15,52	56,89	181,97	163,67	1 721,46	1 047,05
Rbf	12,74	11,91	**4,04**	6,29	5,34	143,31	749,73
RbfFast	13,25	11,99	**1,51**	6,34	34,95	34,83	3,37
RbfSlow	11,13	10,20	**3,59**	6,02	33,42	78,24	132,39
Sea	14,31	15,70	**0,27**	44,37	997,36	986,25	553,56
SeaAbrupt	10,42	13,27	**0,25**	3,68	339,53	491,72	182,28
SeaGradual	10,76	12,17	**0,23**	3,87	255,16	512,17	242,82
Sine	11,07	12,79	**0,17**	14,21	15,63	674,18	54,58
SineAbrupt	5,25	**5,23**	14,47	59,41	18,45	549,24	79,07
SineGradual	5,19	5,17	12,09	**2,51**	15,43	502,56	97,32
Waveform	11,84	13,57	**0,19**	6,08	724,54	954,28	941,19
WaveformAbrupt	11,03	13,06	**0,18**	6,31	533,13	1 014,42	603,33
WaveformGradual	11,12	13,22	**0,18**	6,15	555,42	1 019,62	677,86
Average	**9,81**	11,53	23,78	84,19	199,63	768,10	412,66
Median	11,05	12,90	**3,73**	6,32	77,68	530,71	157,33
Avg. rank	2,66	3,44	**1,88**	3,41	4,53	6,66	5,44

Table 3.5 – Temps nécessaire, en secondes

	OSMAC	OSMACEns	ASML	AC	ARF	SRP	LB
Bananas	25,63	28,26	44,01	97,91	**4,75**	10,12	7,40
Covertype	8 457,95	14 540,08	85 758,90	121 864,63	**808,73**	4 673,79	6 020,92
Electricity	262,51	383,97	865,69	642,01	**53,38**	156,14	112,00
Image Segments	23,32	20,81	33,90	87,70	**3,72**	16,27	25,16
Keystroke	145,75	238,19	271,12	1 016,73	**19,36**	140,90	170,02
SensItVehicle	1 714,07	1 790,72	819,33	1 585,16	**253,57**	4 966,52	2 410,70
Sensors	28 026,57	39 069,76	64 771,18	115 363,41	**7 449,29**	28 929,35	35 496,93
Agrawal	10 119,49	10 310,27	6 133,53	6 298,18	**2 966,59**	7 911,72	5 980,34
AgrawalAbrupt	21 897,61	48 051,03	6 609,83	6 993,23	**2 673,06**	8 643,25	6 942,14
AgrawalGradual	32 963,17	32 491,99	19 401,41	5 056,13	**2 755,89**	8 389,48	5 613,68
Hyperplane	6 170,15	3 252,95	**1 393,02**	2 124,98	2 013,74	4 613,76	7 210,35
HyperplaneFast	5 073,69	3 398,07	**1 396,77**	2 245,80	2 099,46	4 374,14	5 443,27
HyperplaneSlow	4 241,81	3 393,76	**1 391,53**	2 223,92	2 103,90	4 362,08	5 435,74
Led	9 644,24	8 304,76	5 449,23	8 418,69	**2 359,10**	9 592,00	26 637,77
LedAbrupt	9 085,40	9 304,69	5 528,59	8 975,44	**2 327,25**	11 612,82	16 506,46
LedGradual	8 321,59	10 066,37	6 291,62	9 035,39	**2 331,71**	12 274,69	15 577,46
Mixed	8 856,38	16 479,26	57 315,67	57 777,58	**940,77**	10 288,42	1 638,95
MixedAbrupt	8 857,61	14 676,94	51 580,30	54 231,56	**868,76**	12 276,48	1 626,35
MixedGradual	9 199,62	16 959,83	45 379,37	51 240,33	**959,51**	12 971,54	1 803,38
RandomTree	25 153,12	46 631,16	3 483,44	3 057,53	**2 055,01**	6 729,96	4 135,99
Rbf	14 521,38	27 481,62	61 055,46	89 628,44	**2 146,24**	5 635,64	5 408,06
RbfFast	15 784,83	20 332,82	25 611,59	82 253,12	**2 484,23**	5 875,28	4 787,87
RbfSlow	16 994,59	23 051,80	50 694,17	82 109,19	**2 266,80**	5 214,05	5 306,40
Sea	4 521,21	3 541,77	1 521,88	**1 046,92**	4 496,89	4 834,24	2 206,41
SeaAbrupt	4 262,55	3 467,67	1 516,07	**1 019,95**	2 452,49	4 009,36	1 982,36
SeaGradual	4 300,45	3 411,69	1 497,54	**1 040,16**	2 350,68	4 111,89	2 069,96
Sine	3 769,34	2 979,92	**1 014,18**	1 663,92	1 308,12	4 868,06	1 787,36
SineAbrupt	5 356,67	5 549,42	9 252,10	1 747,61	**1 289,24**	3 374,81	2 000,19
SineGradual	4 830,55	5 208,82	3 948,03	1 391,29	**1 378,57**	4 094,60	2 191,65
Waveform	7 487,84	5 108,00	**2 103,77**	2 370,80	4 688,63	5 953,26	6 049,45
WaveformAbrupt	6 968,94	5 028,79	**2 072,63**	2 281,01	3 885,58	5 954,22	6 134,56
WaveformGradual	5 490,53	5 159,16	**2 087,27**	2 250,95	4 314,67	5 964,22	5 782,75
Average	9 141,52	12 178,57	16 446,66	22 723,11	**2 128,43**	6 650,72	6 078,19
Median	7 228,39	5 379,12	3 715,74	2 325,91	**2 125,07**	5 424,84	5 047,14
Avg. rank	4,91	4,97	3,69	4,19	**1,62**	4,56	4,06

est cependant en bonne voie et presque finie, il est envisagé de soumettre à l'article à une date prochaine. L'objectif principal a ainsi été atteint, mais pas l'objectif secondaire.

Les figures 3.2 et 3.3 présentent respectivement le déroulé prévu du projet et le déroulé réel. Elles montrent que la partie d'obtention des résultats a été plus longue que prévue. De plus, mais les figures ne le montrent pas, l'écriture a aussi été longue, et c'est un point à améliorer.

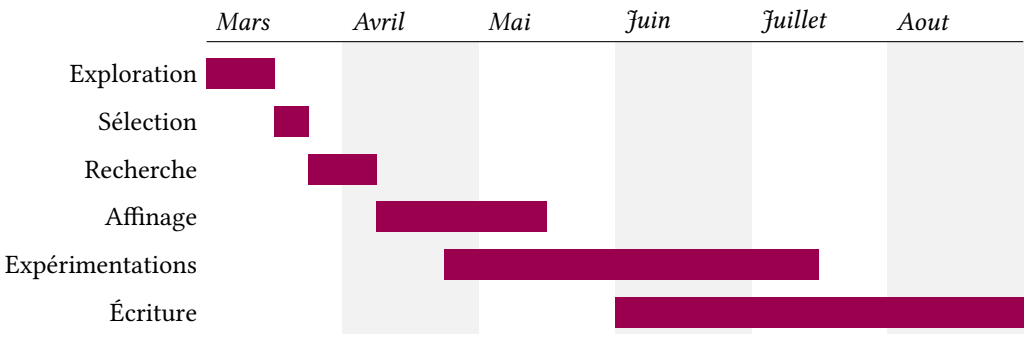

FIGURE 3.2 – Diagramme de Gantt du déroulé prévisionnel

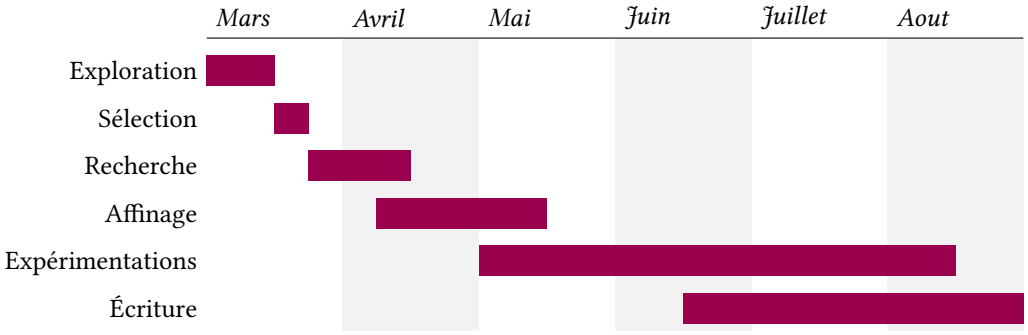

FIGURE 3.3 – Diagramme de Gantt du déroulé réel

4 Conclusion

4.1 Conclusion technique

Nous avons présenté l'algorithme OnlineSMAC, qui propose une solution au problème CASH pour l'AutoML en ligne en utilisant l'optimisation bayésienne. C'est le premier algorithme à proposer cette approche pour l'apprentissage automatique en ligne.

Notre algorithme se classe à un niveau similaire en performance prédictive à l'état de l'art.

C'est une piste prometteuse et à approfondir pour réduire l'empreinte en ressources nécessaire.

4.2 Conclusion personnelle

J'ai beaucoup aimé ce stage. Il m'a permis de découvrir de nouvelles perspectives sur l'ingénierie et la conception de systèmes. J'ai réalisé pendant cette période un travail plus proche de l'idéal de l'ingénierie que ce que beaucoup de perspective alternatives m'auraient offert.

À mon opinion, tous les élèves ingénieurs devraient être exposés activement à l'activité de recherche, sans se limiter à faire un résumé de l'état de l'art.

J'ai confirmé un peu plus mon rapport à l'écriture. Écrire un article est un processus difficile. J'ai avec évidence beaucoup de progrès à faire de ce côté.

Cette expérience m'a donné envie de continuer sur cette voie. Ma candidature a été acceptée pour faire un doctorat en continuité de mon stage dans cette même équipe.

Bibliographie

[1] J. A. Nelder et R. Mead. « A Simplex Method for Function Minimization ». In : *The Computer Journal* 7.4 (1er jan. 1965), p. 308-313. ISSN : 0010-4620. DOI : 10.1093/comjnl/7.4.308. URL : https://doi.org/10.1093/comjnl/7.4.308 (visité le 15/03/2024).

[2] *Loi n°67-7 du 3 janvier 1967 portant création d'organismes de recherhce (CNEXO, ANVAR, IRIA)*. 3 jan. 1967. URL : https://www.legifrance.gouv.fr/jorf/id/JORFTEXT000000692719 (visité le 26/05/2024).

[3] Ministère de l'industrie. *Décret n° 79-1158 du 27 décembre 1979 portant création d'un institut national de recherche en informatique et en automatique*. 27 déc. 1979. URL : https://www.legifrance.gouv.fr/jorf/id/JORFTEXT000000519325 (visité le 26/05/2024).

[4] Pedro Domingos et Geoff Hulten. « Mining High-Speed Data Streams ». In : *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '00. New York, NY, USA : Association for Computing Machinery, 1er août 2000, p. 71-80. ISBN : 978-1-58113-233-5. DOI : 10.1145/347090.347107. URL : https://dl.acm.org/doi/10.1145/347090.347107 (visité le 06/03/2024).

[5] Ralf Klinkenberg. « Meta-Learning, Model Selection, and Example Selection in Machine Learning Domains with Concept Drift ». In : *Proceedings of the Annual Workshop of the Special Interest Group on Machine Learning, Knowledge Discovery, and Data Mining (FGML-2005) of the German Computer Science Society*. Sous la dir. de J. Furnkranz et G. Grieser. Jan. 2005, p. 164-171.

[6] Alain Beltran et Pascal Griset. *Histoire d'un pionnier de l'informatique : 40 ans de recherche à l'INRIA*. 1re éd. Sciences & histoires. Les Ulis : EDP sciences, 2007. ISBN : 978-2-86883-806-3.

[7] Albert Bifet, Geoff Holmes et Bernhard Pfahringer. « Leveraging Bagging for Evolving Data Streams ». In : *Machine Learning and Knowledge Discovery in Databases*. Sous la dir. de José Luis Balcázar, Francesco Bonchi, Aristides Gionis et Michèle Sebag. Berlin, Heidelberg : Springer, 2010, p. 135-150. ISBN : 978-3-642-15880-3. DOI : 10.1007/978-3-642-15880-3_15. URL : https://link.springer.com/content/pdf/10.1007/978-3-642-15880-3_15.pdf.

[8] James Bergstra, Rémi Bardenet, Yoshua Bengio et Balázs Kégl. « Algorithms for Hyper-Parameter Optimization ». In : *Proceedings of the 24th International Conference on Neural Information Processing Systems*. NIPS'11. Red Hook, NY, USA : Curran Associates Inc., 12 déc. 2011, p. 2546-2554. ISBN : 978-1-61839-599-3. URL : http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization (visité le 20/03/2024).

[9] Frank Hutter, Holger H. Hoos et Kevin Leyton-Brown. « Sequential Model-Based Optimization for General Algorithm Configuration ». In : *Learning and Intelligent Optimization*. LION2011. Sous la dir. de Carlos A. Coello Coello. T. 6683. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 2011, p. 507-523. ISBN : 978-3-642-25566-3. DOI : 10.1007/978-3-642-25566-3_40.

[10] Chris Thornton, Frank Hutter, Holger H. Hoos et Kevin Leyton-Brown. « Auto-WEKA : Combined Selection and Hyperparameter Optimization of Classification Algorithms ». In : *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '13. New York, NY, USA : Association for Computing Machinery, 11 août 2013, p. 847-855. ISBN : 978-1-4503-2174-7. DOI : 10.1145/2487575.2487629. URL : https://dl.acm.org/doi/10.1145/2487575.2487629 (visité le 05/03/2024).

[11] Stefan Wager, Trevor Hastie et Bradley Efron. « Confidence Intervals for Random Forests : The Jackknife and the Infinitesimal Jackknife ». In : *The Journal of Machine Learning Research* 15.1 (1er jan. 2014), p. 1625-1651. ISSN : 1532-4435.

[12] Heitor M. Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes et Talel Abdessalem. « Adaptive Random Forests for Evolving Data Stream Classification ». In : *Machine Learning* 106.9 (1er oct. 2017), p. 1469-1495. ISSN : 1573-0565. DOI : 10.1007/s10994-017-5642-8. URL : https://doi.org/10.1007/s10994-017-5642-8 (visité le 10/04/2024).

[13] Albert Bifet, Ricard Gavaldà, Geoffrey Holmes et Bernhard Pfahringer. *Machine Learning for Data Streams : With Practical Examples in MOA*. The MIT Press, 2 mars 2018. ISBN : 978-0-262-34604-7. DOI : 10.7551/mitpress/10654.001.0001. URL : https://direct.mit.edu/books/oa-monograph/4475/Machine-Learning-for-Data-Streamswith-Practical (visité le 05/03/2024).

[14] Heitor Murilo Gomes, Jean Paul Barddal, Luis Eduardo Boiko Ferreira et Albert Bifet. « Adaptive Random Forests for Data Stream Regression ». In : *Proceedings of the 26th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. 26th European Symposium on Artificial Neural Networks, ESANN 2018. Bruges, Belgium, 25-27 avr. 2018. ISBN : 978-287587047. URL : https://www.esann.org/sites/default/files/proceedings/legacy/es2018-183.pdf.

[15] Heitor Murilo Gomes, Jesse Read et Albert Bifet. « Streaming Random Patches for Evolving Data Stream Classification ». In : *2019 IEEE International Conference on Data Mining (ICDM)*. 2019 IEEE International Conference on Data Mining (ICDM). Beijing, China : IEEE, nov. 2019, p. 240-249. DOI : 10.1109/ICDM.2019.00034. URL : https://ieeexplore.ieee.org/abstract/document/8970784 (visité le 02/05/2024).

[16] Matthias Carnein, Heike Trautmann, Albert Bifet et Bernhard Pfahringer. « confStream : Automated Algorithm Selection and Configuration of Stream Clustering Algorithms ». In : *Learning and Intelligent Optimization : 14th International Conference, LION 14, Athens, Greece, May 24–28, 2020, Revised Selected Papers*. Berlin, Heidelberg : Springer-Verlag, 24 mai 2020, p. 80-95. ISBN : 978-3-030-53551-3. DOI : 10.1007/978-

3-030-53552-0_10. URL : https://doi.org/10.1007/978-3-030-53552-0_10 (visité le 11/03/2024).

[17] Bruno Veloso, João Gama, Benedita Malheiro et João Vinagre. « Hyperparameter Self-Tuning for Data Streams ». In : *Information Fusion* 76 (1er déc. 2021), p. 75-86. ISSN : 1566-2535. DOI : 10.1016/j.inffus.2021.04.011. URL : https://www.sciencedirect.com/science/article/pii/S1566253521000841 (visité le 15/03/2024).

[18] Qingyun Wu, Chi Wang, John Langford, Paul Mineiro et Marco Rossi. « ChaCha for Online AutoML ». In : *Proceedings of the 38th International Conference on Machine Learning*. International Conference on Machine Learning. T. 139. Proceedings of Machine Learning Research. Online : PMLR, 1er juill. 2021, p. 11263-11273. URL : https://proceedings.mlr.press/v139/wu21d.html (visité le 18/03/2024).

[19] Cedric Kulbach, Jacob Montiel, Maroua Bahri, Marco Heyden et Albert Bifet. « Evolution-Based Online Automated Machine Learning ». In : *Advances in Knowledge Discovery and Data Mining*. Sous la dir. de João Gama, Tianrui Li, Yang Yu, Enhong Chen, Yu Zheng et Fei Teng. Lecture Notes in Computer Science. Cham : Springer International Publishing, 10 mai 2022, p. 472-484. ISBN : 978-3-031-05933-9. DOI : 10.1007/978-3-031-05933-9_37. URL : https://inria.hal.science/hal-03667231.

[20] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass et Frank Hutter. « SMAC3 : A Versatile Bayesian Optimization Package for Hyperparameter Optimization ». In : *Journal of Machine Learning Research* 23.54 (2022), p. 1-9. ISSN : 1533-7928. DOI : 10.48550/arXiv.2109.09831. URL : http://jmlr.org/papers/v23/21-0888.html (visité le 25/03/2024).

[21] Maroua Bahri et Nikolaos Georgantas. « AutoClass : AutoML for Data Stream Classification ». In : 8th Workshop on Real-time Stream Analytics, Stream Mining, CER/CEP & Stream Data Management in Big Data in Conjunction with the IEEE International Conference on Big Data 2023. Sorrento, Italy : IEEE, 15 déc. 2023, p. 2658-2666. DOI : 10.1109/BigData59044.2023.10386362. URL : https://inria.hal.science/hal-04338637 (visité le 08/03/2024).

[22] Bilge Celik, Prabhant Singh et Joaquin Vanschoren. « Online AutoML : An Adaptive AutoML Framework for Online Learning ». In : *Machine Learning* 112.6 (1er juin 2023), p. 1897-1921. ISSN : 1573-0565. DOI : 10.1007/s10994-022-06262-0. URL : https://link.springer.com/content/pdf/10.1007/s10994-022-06262-0.pdf (visité le 22/03/2024).

[23] Roman Garnett. *Bayesian Optimization*. Cambridge : Cambridge University Press, 2023. ISBN : 978-1-108-42578-0. DOI : 10.1017/9781108348973. URL : https://bayesoptbook.com/ (visité le 25/03/2024).

[24] *Rapport d'évaluation d'Inria*. Paris : Hcéres, 7 mars 2024, p. 48. URL : https://www.hceres.fr/sites/default/files/media/publications/rapports_evaluations/pdf/C2023-EV-0780491K-DEO-ORG230023625-RD.pdf (visité le 19/05/2024).

[25] Nilesh Verma, Albert Bifet, Bernhard Pfahringer et Maroua Bahri. « ASML : A Scalable and Efficient AutoML Solution for Data Streams ». In : AutoML 2024 - International Conference on Automated Machine Learning. Paris, France, 9 sept. 2024. URL : https://inria.hal.science/hal-04581479 (visité le 16/09/2024).

Colophon

Ce rapport a été mis en page avec le logiciel LaTeX. Le style utilise la classe `scrreport` de KOMA-Script comme base. Le texte a été rédigé dans GNU Emacs.

Les polices d'écriture sont Libertinus Serif et Libertinus Sans pour le texte et Latin Modern Mono pour le code.