Sélection efficace de variables par descente par coordonnée avec garanties théoriques

David Saltiel, Eric Benhamou

Sélection efficace de variables par descente par coordonnée avec garanties théoriques

David Saltiel[1] et Eric Benhamou[2]

[1]AI Square Connect, LISIC
[2]AI Square Connect, LAMSADE

10 avril 2019

Résumé

Malgré l'avènement de l'apprentissage par représentation, principalement par le biais d'apprentissage profond, la sélection des variables (*features*) reste un élément clé de nombreux scénarios d'apprentissage automatique. Cet article présente une nouvelle méthode théoriquement motivée pour la sélection des features. Cette approche traite le problème de la sélection des features par le biais de méthodes d'optimisation par coordonnées en tenant compte des dépendances des variables, matérialisant ces dernières par blocs. Le faible nombre d'itérations (jusqu'à la convergence de la méthode) atteste de l'efficacité des méthodes de gradient boosting (par exemple l'algorithme XGBoost) pour ces problèmes d'apprentissage supervisé. Dans le cas de fonctions convexes et lisses, nous pouvons prouver que le taux de convergence est polynomial en terme de dimension de l'ensemble complet des features. Nous comparons les résultats obtenues avec des méthodes faisant l'état de l'art de la sélection des features : Recursive Features Elimination (RFE) et Binary Coordinate Ascent (BCA), afin de montrer que cette nouvelle méthode est compétitive.

Mots-clef : CAP, sélection des features, descente par coordonnée, méthode de gradient boosting.

1 Introduction

La sélection des features est également appelée sélection de variables ou d'attributs. Cette méthode concerne la sélection d'un sous-ensemble d'attributs pertinents dans nos données qui sont les plus pertinents pour notre problème de modélisation prédictive. C'est un domaine de recherche et de développement actif et fructueux depuis des décennies dans l'apprentissage statistique qui s'est avéré efficace et utile à la fois en théorie et en pratique pour plusieurs raisons : l'amélioration de l'efficacité de l'apprentissage et de la précision prédictive (voir [MMP02]), la simplification du modèle pour en faciliter l'interprétation et l'amélioration des performances (voir [AD94], [KS96] et [BL97]), le temps de la phase d'entraînement plus courte (voir [MMP02]), la manière de limiter la malédiction de la dimensionnalité, l'amélioration de la généralisation en réduisant l'overfiting et la variance. [HTF09] et [GE03] sont de bonnes références pour obtenir un aperçu des différentes méthodes permettant de traiter les sélections des features. Les méthodes peuvent être classées en trois catégories principales : méthode de filtrage (Filter methods), d'emballage (Wrapper methods) et intégrées (Embedded methods).

2 Cadre de l'étude

2.1 Structure par blocs

[DO18] traite du choix du nombre de blocs à prendre en compte dans un problème de sélection de fonctionnalités. Nous proposons une réponse au problème ouvert discuté dans [BT13] à propos de la résolution d'une minimisation sans contrainte en utilisant une méthode de projection dans laquelle chaque itération consiste à effectuer une étape de projection du gradient par rapport à un bloc donné pris dans un ordre cyclique ou

en utilisant une méthode de minimisation alternative utilisant seulement deux blocs. En effet, nous décidons de scinder les données en $n \geq 2$ blocs $\left(\left[B^1\right] \dots \left[B^n\right] \right)$ et au lieu d'effectuer simplement une méthode de montée de gradient sur un seul bloc à chaque itération, nous initialisons d'abord notre méthode en trouvant le meilleur jeu de données de sous-blocs (en terme de score de prédiction) pour notre problème d'apprentissage supervisé, puis appliquons une méthode de montée de gradient sur chaque bloc à chaque itération (pour plus de détails, voir la section 3).

2.2 Résultats de convergence

Afin de motiver notre méthode de montée par coordonnées, nous rappelons quelques résultats théoriques sur la convergence des méthodes d'optimisation par montée par coordonnées. La théorie est bien comprise pour le cas convexe (voir [Wri15]). Le cas non convexe sans gradient, qui est notre exemple, est cependant beaucoup plus difficile, car nous avons un problème de minimum local et des hypothèses mathématiques trop faibles pour pouvoir prouver la convergence. Cependant, les résultats de la convergence dans le cas d'une fonction convexe forte donnent une idée de l'efficacité de cette méthode et de son taux de convergence linéaire. Notre preuve, fournie en annexe, est inspirée de [Nes12] avec une légère modification puisque nous commençons par la condition de point critique. Nous fournissons également un lemme en annexe afin d'obtenir rapidement la preuve sur les résultats de convergence. Pour obtenir un résultat significatif, nous devons émettre certaines hypothèses nécessaires à la minimisation de notre fonction f. Nous supposons que nous avons une fonction n-dimensionnelle à valeur réelle $f : \mathbb{R}^n \to \mathbb{R}$. Nous imposons certaines conditions de régularité données dans 2.1 afin d'être en mesure de tirer des conclusions significatives. Évidemment, même si notre problème final est une maximisation, il est trivial de transformer le programme de minimisation en un programme de maximisation en prenant l'opposé de la fonction objectif. Dans cette section, nous nous en tenons à la présentation traditionnelle et examinons la minimisation pour faciliter la lecture. Nous examinons donc le programme d'optimisation suivant :

$$\min_{x \in \mathbb{R}^n} f(x) \tag{1}$$

Hypothèses 2.1. *Nous supposons que notre fonction f est deux fois différentiable et fortement convexe par rapport à la norme euclidienne :*

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{\sigma}{2} \|y - x\|_2^2 \tag{2}$$

avec $\sigma > 0$ pour tout couple $(x, y) \in \mathbb{R}^n \times \mathbb{R}^n$. Nous supposons également que la coordonnée de chaque gradient est uniformément L_i Lipschitz, c'est-à-dire qu'il existe une constante L_i telle que pour tout $x \in \mathbb{R}^n, t \in \mathbb{R}$

$$\left|[\nabla f(x + te_i)]_i - [\nabla f(x)]_i\right| \leq L_i |t| \tag{3}$$

Avec e_i le vecteur de la base canonique.
On note $L_{\max}$ le maximum de ces constantes de Lipschitz :

$$L_{max} = \max_{i=1\dots n} L_i \tag{4}$$

Nous supposons que le minimum de f, noté $f^\star$, est atteignable et que la valeur de gauche de l'épigraphe par rapport à notre point de départ initial x_0 est bornée, c'est-à-dire :

$$\max_x \{\|x - x^\star\| : f(x) \leq f(x_0)\} \leq R_0 \tag{5}$$

Remarque 2.1. *La convexité forte signifie que la fonction se situe entre deux paraboles. La condition* (3) *implique que la croissance du gradient est au plus linéaire. L'inégalité* (5) *indique que la fonction augmente à l'infini.*

Proposition 2.1. *Sous l'hypothèse 2.1, l'optimisation par montée par coordonnée (voir algorithme 2) converge vers le minimum global $f^\star$ à un taux linéaire proportionnel à $2nL_{\max}R_0^2$, donnée par l'inégalité suivante :*

$$\mathbb{E}[f(x_k)] - f^\star \leq \frac{2nL_{\max}R_0^2}{k} \tag{6}$$

Démonstration. Preuve donnée en annexe. □

Remarque 2.2. *La proposition 2.1 nous donne un contrôle théorique sur le nombre maximal d'itérations nécessaires pour obtenir une erreur estimée entre l'optimum réel $f^\star$ et notre optimum estimé calculé par $\mathbb{E}[f(x_k)]$. Pour obtenir une précision de ε, nous devrions avoir :*

$$\frac{2nL_{\max}R_0^2}{k} \leq \varepsilon$$

ce qui conduit à un nombre d'itérations minimum pratique $k_{\max,1}$ donné par :

$$k_{\max,1} = \left\lceil \frac{2nL_{\max}R_0^2}{\varepsilon} \right\rceil \tag{7}$$

où $\lceil x \rceil$ est la fonction partie entière supérieure, c'est-à-dire le plus petit entier supérieur ou égal à x.

Nous avons un contrôle supplémentaire de notre taux de convergence qui est donné par la proposition suivante :

Proposition 2.2. *Sous les hypothèses 2.1, et pour $\sigma > 0$, l'erreur d'optimisation par montée par coordonnées (cf. algorithme 2) est contrôlée par l'inégalité suivante :*

$$\mathbb{E}[f(x_k)] - f^\star \leq \left(1 - \frac{\sigma}{nL_{\max}}\right)^k (f(x_0) - f^\star) \quad (8)$$

Démonstration. Preuve donnée en annexe. □

Remarque 2.3. *Cette dernière proposition est intéressante car elle fournit une deuxième valeur théorique du nombre d'itérations. Si nous voulons obtenir une précision de ε pour notre estimation du minimum, nous devrions avoir :*

$$\left(1 - \frac{\sigma}{nL_{\max}}\right)^k (f(x_0) - f^\star) \leq \varepsilon$$

ce qui conduit à un nombre maximum pratique d'itérations $k_{\max,2}$ donné par :

$$k_{\max,2} = \left\lceil \log\left(\frac{\varepsilon}{f(x_0) - f^\star}\right) / \log\left(1 - \frac{\sigma}{nL_{\max}}\right) \right\rceil \quad (9)$$

Remarque 2.4. *Notre fonction à maximiser n'est évidemment pas convexe. Cependant, un taux linéaire dans le cas convexe est plutôt une bonne performance pour la méthode d'optimisation de montée par coordonnée. Pourvu que la méthode se généralise, ce qui est encore à l'étude, ce taux de convergence est une bonne intuition sur l'efficacité de cette dernière.*

Remarque 2.5. *En combinant les résultats dans (7) et (9), le nombre optimal d'itérations doit être égal au minimum de nos deux bornes précédentes :* $\min(k_{\ max,1}, k_{\ max,2})$*, c'est-à-dire :*

$$\min\left(\left\lceil \frac{2nL_{\max}R_0^2}{\varepsilon} \right\rceil, \left\lceil \frac{\log\left(\frac{\varepsilon}{f(x_0)-f^\star}\right)}{\log\left(1 - \frac{\sigma}{nL_{\max}}\right)} \right\rceil\right)$$

La comparaison entre les deux nombres possibles d'itérations est présentée dans la figure 1. Fait intéressant, les deux vitesses de convergence jouent un rôle différent. Pour les petites précisions (grand ε), le nombre d'itération est ceui de l'équation (7), alors que pour les valeurs de haute précision (faible ε), nous retenons celui issu de l'équation (9) .

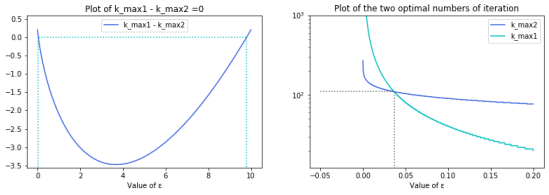


FIGURE 1 – Comparaison entre les deux nombres possibles d'itérations : nous avons pris comme exemple une fonction à uni-dimensionnelle à minimiser avec un point de départ à 10 et nous supposons la convention traditionnelle selon laquelle le minimum est à l'origine. Nous supposons ensuite les constantes $L_{\max}$ et R_0 égales à un et la constante σ égales à 0.1. Nous remarquons d'abord que nous avons deux cas différents selon si $k_{\max,1}$ est supérieur ou inférieur à ,$k_{\max,2}$, mais il est naturel dans notre cas d'étudier uniquement le cas lorsque la précision n'est supérieure à 1. Ce raisonnement conduit donc au choix d'une précision inférieure à $3,36\%$ et au choix de $k_{\max,2}$.

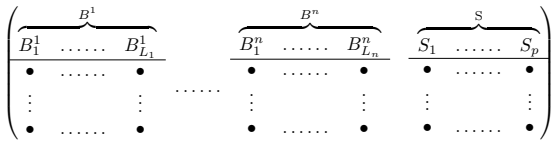
3 Méthode développée

Dans de nombreuses applications, nous pouvons regrouper des fonctionnalités parmi des familles que nous appelons blocs de variables (ou features block variables). Un exemple typique consiste à regrouper des variables qui sont des observations d'une quantité physique mais à un moment différent (comme la vitesse de la mesure du vent à différentes heures pour un problème de prédiction d'énergie, le prix d'un titre dans un algorithmique de stratégie de trading sur les marchés financiers, la température ou battement de cœur d'un patient à un moment différent, ...). On note désormais $\mathcal{M}_{(n,p)}$ l'espace des matrices à n lignes et p colonnes. En supposant que nous ayons J lignes de données, nous pouvons regrouper formellement nos variables en deux ensembles :

- le premier ensemble contient les variables $\left[B^1\right] \dots [B^n]$. Ce sont des variables par bloc de longueur différente L_i. Mathématiquement, les variables de sous-bloc sont désignées par B_j^i avec $(B_j^i)_{\substack{i\in 1\dots n \\ j\in 1\dots L_i}}$ valeur de prise dans $\mathbb{R}^J$,

- le second ensemble est noté $[S]$ et est un bloc de p variables uniques.

Graphiquement, nos variables peuvent être représentées comme suit :



Ainsi, nous avons$N$variables réparties entre variables de bloc et variables simples, donc $N = N_B + p$ avec $N_B = \sum_{i=1}^{n} L_i$.

Notre algorithme fonctionne comme suit. Nous ajustons d'abord notre modèle de classification pour trouver un classement de l'importance de nos features sur la prédiction. Cette importance (notée feature importance dans la littérature) est calculée à l'aide l'indice de Gini pour chaque variable. Nous conservons ensuite les k premières features les mieux classées pour chaque bloc $B^1 \dots B^n$ afin de trouver la meilleure estimation initiale pour notre algorithme de montée par coordonnée. Notez que l'ensemble de variables uniques n'est pas modifié au cours de la première étape de la procédure. La fonction objectif est le nombre d'échantillons correctement classés à chaque itération. Il peut être exprimé comme suit :

$$Accuracy = \frac{TP + TN}{\text{number of examples}} \tag{10}$$

avec True Positives (TP) indiquant le nombre d'exemples positifs étiquetés comme positifs et True Negatives (TN), le nombre d'exemples négatifs étiquetés comme négatifs.

Remarque 3.1. *Notez que le nombre d'exemples (indiqué par n dans cette remarque) respecte la relation suivante : $n = TP + TN + FP + FN$ avec False Positives (FP) indiquant le nombre d'exemples négatifs qualifiés de positifs et de Faux négatifs (FN) le nombre d'exemples positifs étiquetés comme négatifs.*

Nous entrons ensuite dans la boucle principale de l'algorithme. En partant du vecteur $\left(k, \dots, k,\ \mathbb{1}_p^T\right)$ comme estimation initiale de notre algorithme, nous effectuons notre optimisation de montée par coordonnée afin de trouver l'ensemble de features avec le score optimal et le cardinal le plus faible. La procédure de montée par coordonnée s'arrête lorsque nous atteignons le nombre maximal d'itérations ou que la solution optimale actuelle n'a pas été améliorée d'au moins un certain seuil entre deux étapes.

En prenant les notations précédentes, nous commençons avec le jeu de données initial par blocs $\left(\left[B_1^1 \dots B_{L_1}^1\right] \dots \left[B_1^n \dots B_{L_n}^n\right], [S]\right)$ afin de le réduire au nouvel ensemble de données $\left(\left[B^{1,\diamond}\right] \dots \left[B^{n,\diamond}\right], [S]\right)$, avec $B^{i,\diamond} \in \mathcal{M}_{(J,k)}\ \ \forall i \in 1 \dots n$. Nous pouvons désormais appliquer dans un premier temps une optimisation de montée par coordonnées sur chaque bloc afin d'obtenir une nouvelle structure de jeu de données sans tenir compte des blocs. Ensuite, nous appliquons une optimisation de montée par coordonnée binaires sur le reste du jeu de données afin d'obtenir le jeu de données final $\left(\left[B^{1,\star}\right] \quad \dots \quad \left[B^{n,\star}\right], \quad [S^\star]\right)$. La procédure est détaillée dans la figure 2. Nous résumons l'algorithme dans le pseudo-code 1. Pour contrôler l'arrêt anticipé, nous utilisons des variables de précision notées ε_1 et ε_2, ainsi que deux itérations maximales Itération $\max_1$ et Itération $\max_2$ qui sont initialisées avant de démarrer la procédure de l'algotithme. Nous notons également Score$(k_1,\ \dots,\ k_n,\ \mathbb{1}_p)$ le score de précision (ou accuracy score) de notre classifieur avec chaque bloc de variables B_i conservant les k_i meilleures variables et avec les variables uniques toutes retenues.

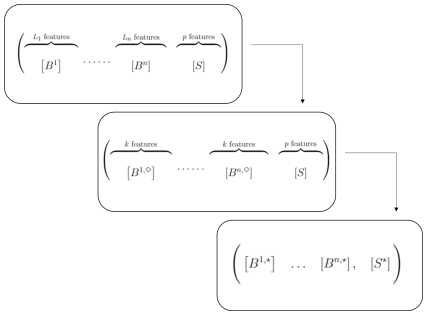


FIGURE 2 – Résumé de notre méthode : nous calculons d'abord la meilleure valeur pour k afin d'avoir tous les blocs (à l'exception du dernier bloc de variables uniques) de k variables optimisant la fonction score. Nous appliquons ensuite une méthode de montée par coordonnée sur chaque bloc afin de maximiser la fonction score. Nous appliquons enfin une montée par coordonnée binaires sur les variables restantes. Nous noterons Optimal Coordinate Ascent (OCA) méthode développée dans la suite du papier.

Algorithm 1 Algorithme OCA :

J Best optimization
We retrieve features importance from a fitted model
We find the index $k^\star$ that gives the best score for variables block of same size k :
$k^\star \in \underset{k \in \mathbb{R}^{L_{\min}}}{\text{argmax}} \text{Score}\,(k,\ \ldots,\ k,\ \mathbb{1}_p)$
Initial guess : $x^0 = (k^\star, \ldots, k^\star, \mathbb{1}_p)$
while $|\text{Score}(x^i) - \text{Score}(x^{i-1})| \geq \varepsilon_1$ and $i \leq$ Iteration $\max_1$ **do**
$x_1^i \in \underset{j \in \mathbb{R}^{L_1}}{\text{argmax}} \text{Score}\left(j,\ x_2^{i-1},\ x_3^{i-1},\ \ldots\ ,\ x_n^{i-1}, \mathbb{1}_p\right)$
...
$x_n^i \in \underset{j \in \mathbb{R}^{L_n}}{\text{argmax}} \text{Score}\left(x_1^i,\ x_2^i,\ x_3^i,\ \ldots\ , j, \mathbb{1}_p\right)$
i += 1
end while

Full coordinate ascent optimization
Use previous solutions : $X^* = (x_1^i, \ldots, x_n^i, \mathbb{1}_p)$
$Y^* = \text{Score}\,(X^*)$
while $|Y - Y^*| \geq \varepsilon_2$ and iteration $\leq$ Iteration $\max_2$ **do**
for i=1 ... N **do**
$X = X^*$
$X_i = \text{not}(X_i^*)$
if $\text{Score}\,(X) \geq \text{Score}\,(X^*)$ **then**
$X^* = X$
end if
end for
$Y = \text{Score}\,(X^*)$
iteration += 1
end while
Return X^*, Y^*

Remarque 3.2. *L'originalité de cette optimisation de montée par coordonnée consiste à regrouper les variables par blocs, ce qui permet de réduire le nombre d'itérations par rapport à la montée par coordonnée binaires (BCA), comme présenté dans [ZS16]. La condition d'arrêt peut être modifiée pour s'adapter à d'autres conditions d'arrêt.*

Remarque 3.3. *Il existe de nombreuses variantes à cet algorithme. Il peut être modifié en utilisant une montée par coordonnée aléatoire. Dans ce cas, nous choisissons l'indice de manière aléatoire à chaque étape au lieu d'utiliser l'ordre classique. Le pseudo code 2 est présenté ci-dessous :*

Algorithm 2 Montée par coordonnée aléatoire :

Initialization
Start with $x_0 \in \mathbb{R}^n$
Set $k = 0$
while stop criteria not satisfied **do**
Choose index i_k uniformly distributed in $\{1, \ldots, n\}$ independently from prior iteration
Set $x_{k+1} = x_k - \alpha_k \left[\nabla f\,(x_k)\right]_{i_k} e_{i_k}$ for some $\alpha_k > 0$
Set $k = k + 1$
end while

Remarque 3.4. *La spécificité de notre méthode consiste à conserver les j variables (ou features) les plus représentatives pour chaque bloc, par opposition aux autres méthodes qui ne sélectionnent qu'une variable représentative de chaque bloc, en ignorant ainsi les fortes similitudes entre les variables. Cette procédure revient à considérer en particulier l'opinion opposée de la sélection de variables par variables artificielles élimination de la redondance, telle que développée dans [TBRT09].*

3.1 Critères d'arrêt

Pour concevoir un algorithme d'optimisation efficace, les conditions d'arrêt sont essentielles car elles éviteront les itérations inutiles. Nous allons utiliser la précision et le nombre d'itérations discutés dans la section 3.

4 Résultats numériques

4.1 Jeu de données

Nous réalisons notre expérience sur un ensemble de données financières réelles. Sur les marchés financiers, le trading algorithmique est devenu de plus en plus standard au cours des dernières années. L'essor de la machine a été particulièrement important sur les marchés des liquides et de l'électronique, tels que les marchés des changes et des contrats à terme, représentant entre 60 et 80% du volume total des transactions (voir par exemple [Cha13], [GVWZ14] ou [CHV15] pour plus de détails sur les différents marchés). Ces stratégies sont encore plus primordiales tant le marché évolue très rapidement, comme indiqué dans [KKST17]. Une stratégie de trading est généralement définie avec un signal qui

génère une entrée du trade. Mais une fois que nous sommes en position, le point principal suivant est la stratégie de sortie du trade. Il existe plusieurs méthodes pour gérer les sorties efficaces, allant de l'objectif de gain et la limite de perte fixes à l'objectif de gain et la limite de perte dynamiques. En effet, pour imposer le succès et cristalliser le gain ou limiter la perte, une pratique courante consiste à associer un objectif de profit à la stratégie et à arrêter la perte, comme décrit dans divers articles ([LL10], [GDG14], [Fun17] , ou [VKSL18]). Nous utilisons notre algorithme pour effectuer une classification supervisée selon certaines variables a priori. Nous avons effectué 1 500 transactions sur un historique de 10 ans avec 135 variables pouvant être classées en cinq blocs de vingt variables, un bloc de trente variables et cinq variables uniques. Nous savons pour chaque trade s'il s'agit d'un 'bon' ou d'un 'mauvais' trade (grâce aux gains et pertes, dénotés par PnL, engendrés par le trade). L'idée est d'utiliser le nombre minimal de variables pour classer a priori cet ensemble de données. Nous utilisons la méthode de validation croisée avec 70 % pour l'ensemble d'apprentissage et 30 % pour les ensembles de test. Pour une reproductibilité totale, le jeu de données complet et le code python correspondant à cet algorithme sont disponibles publiquement sur github. Nous considérons un cadre de gradient boosting en utilisant la librairie XGBoost, initialement développée dans [Che16], pour la mise en oeuvre de notre algorithme.

Il est intéressant d'examiner l'histogramme des gains et des pertes de nos trades au cours de nos 10 années d'historique. Nous pouvons observer deux pics correspondant à l'objectif de profit et au niveau de limite de perte, comme indiqué dans la figure 3. Il vaut mieux utiliser la courbe des gains et des pertes dans la devise d'origine de l'instrument sous-jacent plutôt que d'examiner la devise consolidée de nos stratégies de trades afin d'éviter le bruit généré par le taux de change. Nous examinerons deux ensembles de données d'une stratégie dite 'mauvaise', appelée stratégie 1, et d'une 'bonne' stratégie, appelé stratégie 2 ; chaque stratégie générant des transactions sur une période donnée.

Le graphique de l'évolution du gain et de la perte des deux stratégies est donné à la figure 4.

Remarque 4.1. *L'ensemble de données de la stratégie 1 est équilibré en termes de label ou étiquettes (PnL engendré par la stratégie) alors que ce n'est pas le cas pour le second, nous ne pouvons donc utiliser la formulation de la précision donnée dans* (10) *car celle-ci mènerait à de mauvais résultats de prédiction. Il est recommandé d'utiliser la définition de la précision balancée, c'est-à-dire (en prenant la notation précédente introduite dans* (10) *et remarque 3.1 :*

$$Balance\ accuracy = \frac{TPR + TNR}{2}$$

avec le vrai taux positif (TPR) et le vrai taux négatif (TNR) définis comme :

$$TPR = \frac{TP}{TP + FN}, \quad TNR = \frac{TN}{TN + FP}$$

Le True Positive Rate (TPR) met l'accent sur l'obtention des bonnes prédictions pour les résultats avec les étiquettes positives, sans prêter attention aux labels négatifs À l'opposé, le taux de TNR

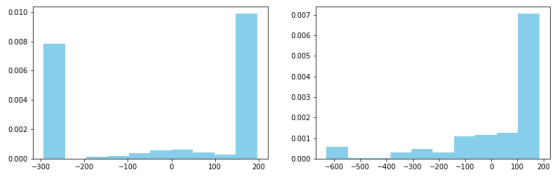


FIGURE 3 – Histogramme du PnL des deux stratégies en devise d'origine. L'histogramme gauche correspond à la stratégie 1 et le droit a la seconde. Comme l'algorithme est exécuté sur un actif sous-jacent côté sur un marché financier américain, la stratégie résultante est libellée en USD. La stratégie utilise un niveau fixe de limite de perte et d'objectif de profit. Cela conduit à deux pics principaux correspondant pour le pic de gauche aux transactions qui se terminent par une perte lorsqu'elles atteignent le niveau de limite de perte et pour le pic de droite aux transactions qui se terminent avec un profit lorsqu'elles atteignent l'objectif de profit.

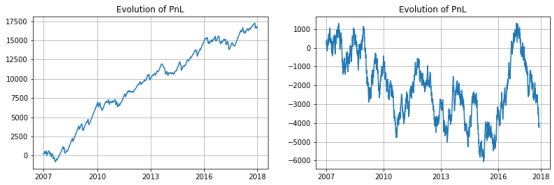


FIGURE 4 – Evolution des gains et pertes des deux stratégies.

(True Negative Rate) rend compte des résultats de prédiction pour les étiquettes négatives, ne prêtant attention qu'aux résultats de prédiction des données étiquetées négativement. Ce raisonnement est similaire aux erreurs de type I et de type II dans les tests statistiques.

4.2 Comparaison

Nous analysons d'abord les résultats donnés par la stratégie 1. Nous comparons notre méthode à deux autres supposées être l'état de l'art de la sélection des variables : RFE et BCA. Notre nouvelle méthode atteint un score de 62,80 % avec 16 % des variables utilisées, tandis que RFE atteint 62,80 % avec 19 % des variables utilisées. La méthode BCA fonctionne moins bien avec un score de 62,19 % pour 27,08 % des variables utilisées. Si nous prenons en termes de critère d'efficacité, le score le plus élevé avec le moins de fonctionnalités, notre méthode est la plus efficace. A titre d'information, avec le même nombre de variables, 16,6 %, RFE obtient un score de 62,39 %. Le tableau 1 résume tous ces chiffres.

TABLE 1 – Comparaison de méthodes pour la stratégie 1 : pour chaque ligne, nous fournissons en rouge la ou les meilleures méthodes (les couleurs les plus chaudes) et en bleu la méthode la plus mauvaise (les couleurs les plus froides), tandis que les méthodes intermédiaires sont en orange. Nous pouvons remarquer que OCA atteint le score le plus élevé avec le minimum de variables. Pour le même nombre de variables, les performances RFE sont pires ou égales, si nous voulons que les performances soient identiques, nous devons disposer d'un ensemble de variables plus large. BCA est la moins bonne méthode en termes de score et de taille de l'ensemble des variables.

Méthode	% de variables	Score (en %)
OCA avec 24 variables	16.6	62.8
RFE avec 24 variables	16.6	62.39
BCA avec 39 variables	27.08	62.19
RFE avec 28 variables	19.4	62.8

5 Discussion

Par rapport à la méthode BCA, notre méthode réduit le nombre d'itérations car elle utilise le fait que les variables peuvent être regroupées en catégories ou en classes. Le nombre d'itérations pour les méthodes OCA et BCA est indiqué ci-dessous dans la figure 5. Notre méthode nécessite seulement 350 étapes d'itérations pour converger, contrairement à BCA qui nécessite jusqu'à 700 étapes d'itérations, car elle considère à l'aveugle les variables ignorant les similarités entre elles.

TABLE 2 – Comparaison des méthodes pour la stratégie 2 :

Méthode	% de variables	Score (en %)
OCA avec 10 variables	6.75	74.28
RFE avec 10 variables	6.75	50.47
BCA avec 52 variables	35.13	77.14

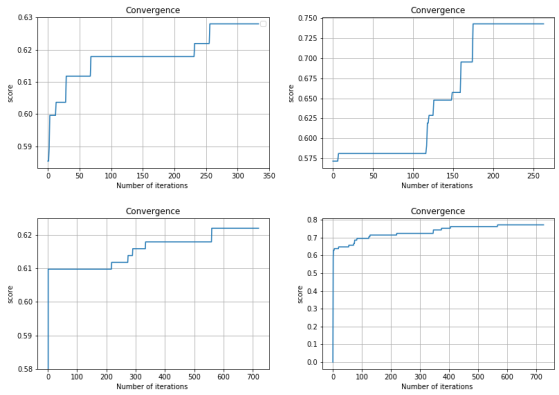


FIGURE 5 – Itérations jusqu'à la convergence pour les méthodes OCA et BCA. La méthode OCA est au dessus et celle de BCA en-dessous. Les graphique de gauche représentent la stratégie 1 tandis que la stratégie 2 est représentée à droite. Nous constatons que la méthode OCA nécessite environ 250 ou 350 étapes d'itération pour converger, alors que la méthode BCA nécessite deux fois plus d'étapes (environ 700 itérations).

Graphiquement, nous pouvons déterminer les meilleurs candidats pour les trois méthodes énumérées dans le tableau 1 sur les figures 6, 7 et 8 pour les deux stratégies. Nous avons pris le code de couleur suivant. La méthode la plus chaude (ou la plus performante) est tracée en rouge tandis que la moins bonne est en bleu. Les méthodes aux performances moyennes sont tracées en orange. Afin de comparer finement les méthodes OCA et RFE, nous avons tracé dans la figure 7 les résultats pour la méthode RFE en utilisant de 10 à 30 % des variables. Nous pouvons remarquer que, pour le

même ensemble de fonctionnalités que OCA, RFE a un score inférieur et que, pour obtenir le même score que OCA, RFE a besoin d'un ensemble de variables supérieur.

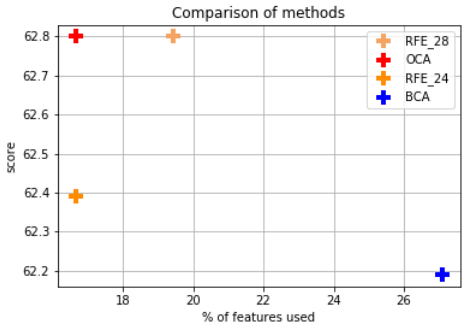


FIGURE 6 – Comparaison entre les 3 méthodes de la stratégie 1. La meilleure méthode est celle qui est dans le coin supérieur gauche. La caractéristique souhaitable est d'avoir le moins de variables possible avec un score le plus élevé. La méthode qui remplit ces critères est la méthode OCA (croix rouge). Le code de couleur est le suivant : rouge pour la meilleure méthode, orange pour celles avec une performance légèrement inférieure et bleu est pour la pire.

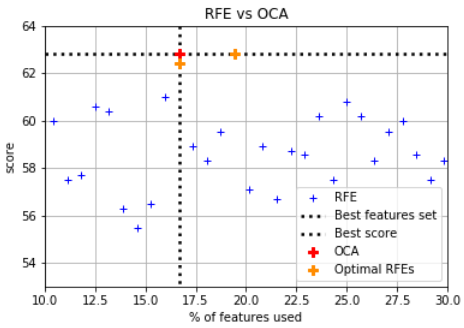


FIGURE 7 – Comparaison entre OCA et RFE pour la stratégie 1 : Pour RFE, nous fournissons le score pour diverses variables définies en bleu. Les deux meilleurs performances de la méthode RFE sont les points en orange qui sont précisément ceux énumérés dans le tableau 1. Le point en rouge représente la méthode OCA. Il atteint la meilleure efficacité car il a le score le plus élevé et le plus petit ensemble de variables pour ce score.

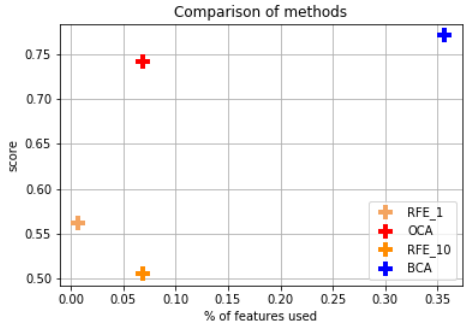


FIGURE 8 – Comparaison entre les 3 méthodes de la stratégie 2. Nous conservons le même code couleur que précédemment. Le point le plus performant pour la méthode RFE a un pourcentage de variables utilisées inférieur à celui de la méthode OCA mais son score est moins élevé. En revanche, la méthode BCA atteint un score plus élevé que les deux autres méthodes mais avec un ensemble de variables plus élevé.

5.1 Réduction du sur-apprentissage

Nous examinons l'objectif final qui consiste à comparer la stratégie de négociation avec et sans apprentissage automatique. Une méthode standard en apprentissage automatique consiste à diviser notre ensemble de données en un ensemble d'entraînement et de test aléatoire. Nous conservons un tiers de nos données à des fins de test afin de détecter tout sur-apprentissage potentiel. Si nous utilisons la méthode standard, voire naïve, de prendre au hasard un tiers des données de notre ensemble de tests, nous mettons un terme à la dépendance temporelle de nos données. Cela a deux conséquences. Nous utilisons dans notre ensemble d'entraînement des données postérieures à nos ensembles d'essais, ce qui n'est pas réaliste par rapport à la réalité. Nous négligeons également tout changement de régime dans nos données en mélangeant des données qui ne datent pas de la même période. Cependant, nous pouvons faire le test sur cette approche traditionnelle et comparer la stratégie de négociation avec et sans filtrage de l'apprentissage automatique. Ceci est fourni dans la figure 9. La courbe orange (respectivement bleue) représente notre stratégie de négociation algorithmique sans filtrage d'apprentissage automatique (avec un filtrage effectué par la méthode xgboost formée avec OCA). Comme la courbe bleue est au-dessus de la courbe orange, nous avons va-

lidé de manière expérimentale le fait que l'apprentissage par machine améliore la rentabilité globale de notre stratégie de négociation en évitant les mauvaises transactions.

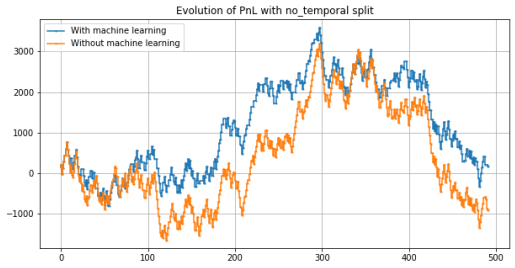


FIGURE 9 – Evolution du PnL pour la stratégie 1 avec un ensemble de tests aléatoires. La courbe orange représente notre stratégie de trading algorithmique sans filtrage par apprentissage automatique, tandis que la ligne bleue combine stratégie de trading algorithmique et filtre oca par xgboost.

Si nous divisons notre ensemble en deux ensembles continus dans le temps, ce qui signifie que nous utilisons comme test d'apprentissage les deux premiers tiers des données selon l'ordre chronologique et comme ensemble de test le dernier tiers, nous obtenons une meilleure performance. Dans ce cas, l'écart entre courbes bleue et orange est plus grand. On peut interpréter cela comme la preuve que la non randomisation de l'ensemble d'entraînement facilite l'apprentissage de notre modèle. Cette méthode de séparation des deux ensembles est présentée dans les graphiques 10 et 11. Comme la courbe bleue est supérieure à celle orange pour les deux stratégies, nous validons expérimentalement que l'approche par machine learning améliore la rentabilité globale de notre stratégie de trading en évitant les mauvaises transactions.

6 Conclusion

Dans cet article, nous avons présenté une nouvelle méthode, appelée Optimal Coordinate Ascent (OCA), qui nous permet de sélectionner des variables parmi des variables individuelles et par blocs. OCA s'appuie sur la montée par coordonnées pour trouver une solution optimale pour le score des méthodes de gradient boosting (nombre d'échantillons correctement classés

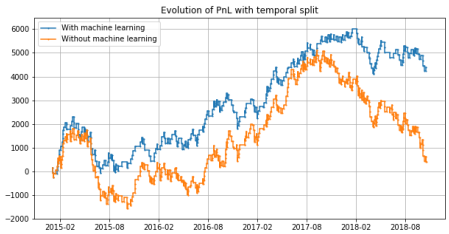


FIGURE 10 – Evolution du PnL pour la stratégie 1 avec un ensemble de test donné par le dernier tiers des données pour tenir compte de la temporalité dans nos données. La courbe orange représente notre stratégie de trading algorithmique sans filtrage de l'apprentissage automatique, tandis que la ligne bleue combine stratégie de trading algorithmique et méthode OCA par xgboost.

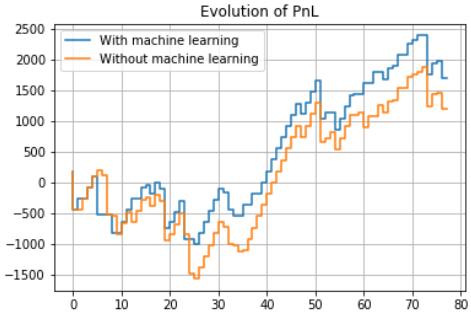


FIGURE 11 – Evolution du PnL pour la stratégie 2 à division temporelle. Nous conservons le même code couleur que précédemment.

pour nos données). OCA prend en compte la notion de dépendances entre les variables formant des blocs dans notre optimisation. L'optimisation de montée par coordonnées résout le problème non polynomial où le nombre de combinaisons explose rapidement, rendant impossible la recherche sur une grille. Il transforme le problème difficile du NP consistant à trouver les meilleures fonctionnalités en un problème de recherche polynomiale. En comparant le résultat avec deux autres méthodes, l'ascension de coordonnées binaires (BCA) et l'élimination récursive des variables (RFE), nous trouvons que l'OCA conduit à l'ensemble de variables minimum avec le score le plus élevé. OCA fournit donc empiriquement le jeu de données le plus compact avec des performances optimales.

Références

[AD94] Hussein Almuallim and Thomas G. Dietterich. Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, 69 :279–305, 1994.

[BL97] Avrim L. Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artif. Intell.*, 97(1-2) :245–271, December 1997.

[BT13] Amir Beck and Luba Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM Journal on Optimization*, 23(4) :2037–2060, 2013.

[Cha13] Ernie Chan. *Algorithmic Trading : Winning Strategies and Their Rationale*. Wiley Publishing, 1st edition, 2013.

[Che16] Chen. Xgboost documentation, 2016.

[CHV15] Benjamin Chaboud, Alain p.and Chiquoine, Erik Hjalmarsson, and Clara Vega. Rise of the machines : Algorithmic trading in the foreign exchange market. *The Journal of Finance*, 69(5) :2045–2084, 2015.

[DO18] Jelena Diakonikolas and Lorenzo Orecchia. Alternating randomized block coordinate descent. In *Proceedings of the 35th ICML*, Proceedings of Machine Learning Research, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[Fun17] Stanley P. Y. Fung. Optimal online two-way trading with bounded number of transactions. *CoRR*, 2017.

[GDG14] Deutsche Bank AG Giuseppe Di Graziano. Optimal trading stops and algorithmic trading. *SSRN*, 2014.

[GE03] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3 :1157–1182, March 2003.

[GVWZ14] Michael Goldstein, Tina Viljoen, P. Joakim Westerholm, and Hui Zheng. Algorithmic trading, liquidity, and price discovery : An intraday analysis of the spi 200 futures. *The Financial Review*, 49(2) :245–270, 2014.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The elements of statistical learning : data mining, inference, and prediction, 2nd Edition*. Springer series in statistics. Springer, 2009.

[KKST17] Andrei Kirilenko, Albert S. Kyle, Mehrdad Samadi, and Tugkan Tuzun. The flash crash : High-frequency trading in an electronic market. *Journal of Finance*, 72(3) :967–998, 2017.

[KS96] Daphne Koller and Mehran Sahami. Toward optimal feature selection. In *ICML'96*, pages 284–292, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[LL10] Mauricio Labadie and Charles-Albert Lehalle. Optimal algorithmic trading and market microstructure. Working papers, HAL, 2010.

[MMP02] Pabitra Mitra, C. A. Murthy, and Sankar K. Pal. Unsupervised feature selection using feature similarity. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(3) :301–312, March 2002.

[Nes12] Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2) :341–362, 2012.

[TBRT09] Eugene Tuv, Alexander Borisov, George Runger, and Kari Torkkola. Feature selection with ensembles, artificial variables, and redundancy elimination. *J. Mach. Learn. Res.*, 10 :1341–1366, December 2009.

[VKSL18] D. Vezeris, T. Kyrgos, C. Take Profit Schinas, and Stop Loss. Trading strategies comparison in combination with an macd trading system. *J. Risk Financial Manag*, 11 :56, 2018.

[Wri15] Wright. Coordinate descent algorithms. *Math. Program.*, 151(1) :3–34, June 2015.

[ZS16] Amin Zarshenas and Kenji Suzuki. Binary coordinate ascent : An efficient optimization technique for feature

subset selection for machine learning. *Knowledge-Based Systems*, 110 :191–201, 2016.