Apprentissage incrémental anytime d'un classifieur Bayésien naïf pondéré

Carine Hue, Marc Boullé, Vincent Lemaire

Orange Labs Lannion, 2 avenue Pierre Marzin, 22300 Lannion

Résumé. Nous considérons le problème de classification supervisée pour des flux de données présentant éventuellement un très grand nombre de variables explicatives. Le classifieur Bayésien naïf se révèle alors simple à calculer et relativement performant tant que l'hypothèse restrictive d'indépendance des variables conditionnellement à la classe est respectée. La sélection de variables et le moyennage de modèles sont deux voies connues d'amélioration qui reviennent à déployer un prédicteur Bayésien naïf intégrant une pondération des variables explicatives. Dans cet article, nous nous intéressons à l'estimation directe d'un tel modèle Bayésien naïf pondéré. Nous proposons une régularisation parcimonieuse de la log-vraisemblance du modèle prenant en compte l'informativité de chaque variable. La log-vraisemblance régularisée obtenue étant non convexe, nous proposons un algorithme de gradient en ligne qui post-optimise la solution obtenue afin de déjouer les minima locaux. Les expérimentations menées s'intéressent d'une part à la qualité de l'optimisation obtenue et d'autre part aux performances du classifieur en fonction du paramétrage de la régularisation.

1 Introduction

Du fait de l'accroissement continu des capacités de stockage, la capture et le traitement des données ont profondément évolué durant ces dernières décennies. Il est désormais courant de traiter des données comprenant un très grand nombre de variables et les volumes considérés sont tels qu'il n'est plus forcément envisageable de pouvoir les charger intégralement : on se tourne alors vers leur traitement en ligne durant lequel on ne voit les données qu'une seule fois. Dans ce contexte, on considère le problème de classification supervisée où Y est une variable catégorielle à prédire prenant J modalités $C_1, \ldots, C_J$ et $X = (X_1, \ldots, X_K)$ l'ensemble des K variables explicatives, numériques ou catégorielles. On s'intéresse à la famille des prédicteurs de type Bayésien naïf. L'hypothèse d'indépendance des variables explicatives conditionnellement à la variable cible rend les modèles directement calculables à partir des estimations conditionnelles univariées de chaque variable explicative. Pour une instance n, la probabilité de prédire la classe cible C conditionnellement aux valeurs des variables explicatives se calcule alors selon la formule :

$$P_w(Y = C|X = x^n) = \frac{P(Y = C)\prod_{k=1}^{K} p(x_k^n|C)^{w_k}}{\sum_{j=1}^{J} P(C_j)\prod_{k=1}^{K} p(x_k^n|C_j)^{w_k}} \quad (1)$$

On considère ici qu'une estimation des probabilités a priori $P(Y = C_j)$ et des probabilités conditionnelles $p(x_k|C_j)$ sont disponibles. Dans notre cas, ces probabilités seront estimées dans les expérimentations par discrétisation ou groupage univarié MODL (cf. (Boullé, 2007)). Ces probabilités univariées étant connues, un prédicteur Bayésien naïf pondéré est décrit entièrement par son vecteur de poids des variables $W = (w_1, w_2, \ldots, w_K)$. Au sein de cette famille de prédicteurs, on peut distinguer :

- les prédicteurs avec des poids à valeur booléenne. En parcourant l'ensemble des combinaisons possibles de valeurs pour le vecteur de poids, on peut calculer le prédicteur MAP à savoir le prédicteur qui maximise la vraisemblance des données d'apprentissage. Cependant, quand le nombre de variables est élevé un tel parcours exhaustif devient impossible et l'on doit se résoudre à un parcours sous-optimal de l'espace $\{0, 1\}^K$.
- les prédicteurs avec des poids continus dans $[0, 1]^K$. De tels prédicteurs peuvent être obtenus par moyennage de modèles du type précédent avec une pondération proportionnelle à la probabilité a posteriori du modèle (Hoeting et al., 1999) ou à leur taux de compression (Boullé, 2007). Cependant, pour des bases comprenant un très grand nombre de variables, on observe que les modèles obtenus par moyennage conservent un très grand nombre de variables ce qui rend les modèles à la fois plus coûteux à calculer et à déployer et moins interprétables.

Dans cet article on s'intéresse à l'estimation directe du vecteur des poids par optimisation de la log vraisemblance régularisée dans $[0, 1]^K$. L'attente principale est d'obtenir via cette approche des modèles robustes comprenant moins de variables, à performances équivalentes. Des travaux préliminaires (Guigourès et Boullé, 2011) ont montré l'intérêt d'une telle estimation directe des poids. La suite de l'article est organisée de la façon suivante : la régularisation parcimonieuse proposée est présentée en section 2 et la mise en place d'un algorithme de gradient en ligne, anytime et à budget limité pour l'optimisation du critère régularisé en section 3. Plusieurs expérimentations sont présentées en section 4 avant un bilan et la présentation de nos perspectives pour ces travaux.

2 Construction d'un critère régularisé

Étant donné un jeu de données $D_N = (x_n, y_n)_{n=1}^N$, on cherche à minimiser sa log-vraisemblance négative qui s'écrit :

$$ll(w, D_N) = -\sum_{n=1}^{N} \left(\log P(Y = y^n) + \sum_{k=1}^{K} \log p(x_k^n|y^n)^{w_k} - \log \left(\sum_{j=1}^{J} P(C_j) \prod_{k=1}^{K} p(x_k^n|C_j)^{w_k} \right) \right) \tag{2}$$

Vu comme un problème classique d'optimisation, la régularisation de la log vraisemblance est opérée par l'ajout d'un terme de régularisation (ou terme d'a priori) qui exprime les contraintes que nous souhaiterions imposer au vecteur de poids W. Le critère régularisé est donc un critère de la forme :

$$CR^{D_N}(w) = -\sum_{n=1}^{N} ll(w, z_n = (x_n, y_n)) + \lambda f(w, X_1, \ldots, X_K) \tag{3}$$

où ll désigne la log vraisemblance, f la fonction de régularisation et λ le poids de la régularisation.
Plusieurs objectifs ont guidé notre choix pour la fonction de régularisation :

1. Sa parcimonie, c'est à dire qu'elle favorise les vecteurs de poids comprenant le plus possible de composantes nulles. Les fonctions de norme L^p sont classiquement utilisées en régularisation par l'ajout d'un terme de régularisation de la forme $\sum_{k=1}^{k} |w_k|^p$. Toutes ces fonctions sont croissantes et favorisent donc les vecteurs de poids dont les composantes sont peu élevées. Pour $p > 1$, la fonction de norme L^p est convexe ce qui facilite l'optimisation et la rend donc attractive. Cependant, du fait de cette convexité, la minimisation de ce terme de régularisation pour $p > 1$ ne conduit pas forcément à l'élimination de variables et rend le choix $p \leq 1$ plus favorable à l'obtention d'un vecteur de poids parcimonieux.
2. Sa capacité à prendre en compte un coefficient C_k associé à chaque variable explicative afin qu'à vraisemblance équivalente, les variables "simples" soient préférées aux variables "complexes". En pondérant la contrainte en norme L^p par la prise en compte d'un tel coefficient, on obtient un terme de pénalisation de la forme : $\sum_{k=1}^{K} C_k * |w_k|^p$. Ce coefficient est supposé connu en amont de l'optimisation. Si aucune connaissance n'est disponible, ce coefficient est fixé à 1. Il peut être utilisé pour intégrer des préférences "métier" entre les variables. Dans notre cas, on y décrit le coût de préparation de la variable, i.e. le coût de discrétisation pour une variable numérique et le coût de groupage pour une variable catégorielle décrits respectivement équations (2.4), resp. (2.7) de (Boullé, 2007).
3. Sa cohérence avec le critère régularisé du prédicteur MODL Bayésien naïf avec sélection binaire des variables (Boullé, 2007). Pour que les deux critères coïncident pour $\lambda = 1$ et w_k à valeurs binaires, on utilise finalement le terme de régularisation :
$f(w, X_1, \ldots, X_K) = \sum_{k=1}^{K} (\log K - 1 + C_k) * w_k^p$

3 Algorithme d'optimisation : descente de gradient par mini-lots avec perturbation à voisinage variable

Notons $p_n = P(Y = y^n)$, $p_j = P(C_j)$, $a_{k,n} = p(x_k^n | y^n)$, $a_{k,j} = p(x_k^n | C_j)$ qui sont toutes des quantités constantes dans ce problème d'optimisation.
Le critère régularisé à minimiser s'écrit alors :

$$CR^{D_N}(w) = -\sum_{n=1}^{N} \left\{ \log p_n + \sum_{k=1}^{K} (w_k * \log a_{k,n}) - \log \left(\sum_{j=1}^{J} p_j \prod_{k=1}^{K} (a_{k,j})^{w_k} \right) \right\} + \lambda \sum_{k=1}^{K} (\log K - 1 + C_k) * {w_k}^p \quad (4)$$

On se donne la contrainte que w soit à valeurs dans $[0, 1]^K$ afin d'obtenir des modèles interprétables. Ce critère est non convexe mais différentiable en tout vecteur de poids avec pour

Entrees : D : flux de données ;
N : profondeur d'historique conservé pour évaluer la valeur du critère ;
L : taille des lots utilisés pour la mise à jour des poids ;
w_0 : vecteur initial de poids ;
η_0 : vecteur initial de pas ;
Max : nombre maximal d'itérations ;
Tol : nombre toléré de dégradations successives;
Sorties : $w^* = argminCR^D(w)$;
t_{total} = nombre d'itérations effectuées ;
while *(Amélioration du critère* **ou** *moins de Tol dégradations successives)* **et** *Nombre d'itérations* $<$ *Max* **do**
    t : index de l'itération courante;
    $D_{t,L}$=t-ème lot de données de taille L;
    $D_{t,N}$=historique de données de taille N terminant à la fin du t-ème lot de données;
    $w^{t+1} = P_{[0,1]^K}\left(w^t - \eta_t \frac{1}{L} \nabla CR^{D_{t,L}}(w^t)\right)$;
    Calcul de η_{t+1};
    Calcul de la valeur du critère sur l'historique de taille N : $CR^{D_{t,N}}(w^{t+1})$;
    if *amélioration du critère* **then**
        mémorisation de la meilleure valeur : $w^* = w_{t+1}$;
    else
        incrémentation du compteur des dégradations successives ;
    end
end

Algorithme 1 : Algorithme de descente de gradient projeté par mini-lots (DGML)

dérivée partielle :

$$\frac{\partial CR^{D_N}(w)}{\partial w_\gamma} = -\sum_{n=1}^{N}\left\{\log a_{\gamma,n} - \frac{\sum_{j=1}^{J} p_j \log a_{\gamma,j} \prod_{k=1}^{K}(a_{k,j})^{w_k}}{\sum_{j=1}^{J} p_j \prod_{k=1}^{K}(a_{k,j})^{w_k}}\right\} + \lambda(\log K - 1 + C_k) * p * {w_\gamma}^{p-1} \quad (5)$$

Le gradient $\nabla CR^{D_N}(w^t)$ est le vecteur de ces dérivées partielles pour $\gamma = 1, \ldots, K$.
On s'est intéressé à sa minimisation par un algorithme de type descente de gradient projeté (Bertsekas, 1976) c'est à dire un algorithme de type descente de gradient pour lequel, à chaque itération, le vecteur w obtenu est projeté sur $[0,1]^K$.
Plusieurs objectifs ont guidé notre choix d'algorithme :

1. algorithme en ligne : que la structure de l'algorithme soit adaptée au traitement d'un flux de données et qu'il ne nécessite donc pas le traitement de la base dans son intégralité ;
2. algorithme anytime : que l'algorithme soit interruptible et qu'il soit en mesure de retourner la meilleure optimisation étant donné un temps de calcul budgété au préalable.

Dans l'algorithme de gradient projeté de type batch, on procède itérativement en mettant à jour le vecteur de poids à chaque itération t selon le gradient calculé sur toutes les instances pondéré par un pas. Si on note w^t le vecteur de poids obtenu à l'itération t, la mise à jour à l'itération $t+1$ s'effectue selon l'équation : $w^{t+1} = P_{[0,1]^K}[w^t - \eta_t \nabla CR^{D_N}(w^t)]$ où le pas

Entrees : T : nombre maximal d'itérations au total;
TailleVoi : taille initiale du voisinage;
Entrees : (DGML) : D : flux de données ;
N : profondeur d'historique conservé pour évaluer la valeur du critère ;
L : taille des lots utilisés pour la mise à jour des poids ;
w_0 : vecteur initial de poids ;
η_0 : vecteur initial de pas ;
Max : nombre maximal d'itérations pour une optimisation DGML;
Tol : nombre toléré de dégradations successives;
Sorties : $w^* = argminCR^D(w)$
Initialisation de w_1^0 à 0.5^K ;
Initialisation $w^* = w_1^0$;
Initialisation $SommeT = 0$;
while $SommeT < T$ **do**
  Calcul de $(w_m^*, t_{total}^m) = DGML(D, N, L, w_m^0, \eta_0, \text{Max}, \text{Tol})$;
  $SommeT = SommeT + t_{total}^m$;
  if *amélioration par rapport à* w^* **then**
    Mémorisation de $w^* = w_m^*$
  else
    TailleVoi $= min(2 * \text{TailleVoi}, 1)$
  end
  $w_{m+1}^0 = P_{[0,1]^K}(w_m^* + Random([-\text{TailleVoi}, \text{TailleVoi}]))$;
end

Algorithme 2 : Algorithme de descente de gradient projeté avec perturbation à voisinage variable (DGML-VNS)

η peut, selon les variantes, être une constante scalaire ou varier selon les itérations et/ou varier selon les composantes du vecteur de poids. La projection sur $[0, 1]^K$ consiste simplement à borner les valeurs obtenues pour les poids sur l'intervalle $[0, 1]$. Cette approche batch suppose que l'on dispose de l'intégralité de la base pour être en mesure de débuter l'optimisation. Dans sa variante stochastique, la mise à jour se fait en intégrant le gradient calculé sur une seule instance. La descente de gradient peut alors se révéler chaotique si la variance du gradient d'une instance à l'autre est élevée. Souhaitant adopter une approche en ligne nous avons retenu une variante à la croisée de ces deux approches (batch et stochastique) à savoir l'approche par mini-lots (Dekel et al., 2012) qui consiste à orienter la descente dans le sens des gradients calculés sur des paquets successifs de données de taille que nous noterons L. Afin que les chemins de descente soient comparables lorsque la taille des lots varie, le gradient utilisé est rapporté à la taille L des mini-lots. L'algorithme de descente de gradient par mini-lots adopté est résumé dans l'algorithme 1.

La valeur optimale du pas η_t a fait l'objet de nombreuses recherches conduisant à des algorithmes plus ou moins coûteux en temps de calcul. Nous avons opté pour la méthode Rprop (Riedmiller et Braun, 1993) : le calcul du pas est spécifique à chaque composante du vecteur c'est à dire que η est un vecteur de pas de dimension K et que chaque composante de ce vecteur est multiplié par un facteur plus grand, resp. plus petit que 1, si la dérivée partielle change, resp.

ne change pas, de signe d'une itération à l'autre. En terme de complexité algorithmique, chaque itération nécessite l'évaluation du critère sur un échantillon de taille N soit une complexité en $O(K * N)$. Pour $L = N$, on retrouve l'algorithme batch classique et pour $L = 1$, le gradient stochastique.

Etant donné la non-convexité du critère à optimiser, il présente en général de nombreux minima locaux vers lesquels une telle descente de gradient peut converger. Il est alors courant de lancer plusieurs descentes de gradient avec des initialisations aléatoires distinctes (approche multi-start) en espérant que l'un de ces chemins de descente conduise au minimum global du critère. Afin de rendre l'optimisation efficace et de ne pas perdre de temps de calcul au début de chacune de ces descentes, il est également possible de perturber la solution obtenue au bout d'un certain nombre d'itérations afin de "sortir" de l'éventuelle cuvette contenant un minimum local. En rendant variable la taille du voisinage dans lequel on perturbe la solution, on se donne les moyens de sortir des minima locaux (approche Variable Neighborhood Search (Hansen et Mladenovic, 2001)). Cette approche notée DGML-VNS est décrite dans l'algorithme 2. On peut remarquer que, pour un voisinage recouvrant intégralement $[0, 1]^K$, l'algorithme DGML-VNS revient à une structure multi-start avec initialisation aléatoire. D'autre part, précisons que le tirage aléatoire peut conduire à une valeur non nulle pour un poids mis à zéro à l'issue du start précédent. Une variable peut donc ré-apparaître en cours de lecture du flux.

L'algorithme DGML-VNS est anytime dans le sens où une estimation du minimum du critère est disponible à la fin de la première descente de gradient et qu'elle est ensuite améliorée en fonction du budget disponible en temps de calcul mais interruptible à tout moment. Sa complexité totale est en $O(T * K * N)$ où T est le nombre total d'itérations autorisées. Ce paramétrage par T permet de limiter le budget maximal utilisé par l'algorithme.

4 Expérimentations

Les premières expérimentations ont pour objectif d'évaluer la qualité de l'optimisation obtenue avec l'algorithme DGML-VNS en fonction de la taille L des mini-lots présentés et du nombre total d'itérations autorisées T. Pour étudier la qualité intrinsèque de l'optimisation indépendamment des performances statistiques du prédicteur associé, nous avons pour cela fixé le poids λ de la régularisation à 0 ce qui revient à optimiser directement la vraisemblance non régularisé. La seconde partie des expérimentations traite des performances statistiques du classifieur obtenu par optimisation du critère régularisé ($\lambda \neq 0$).
Pour l'ensemble des expérimentations les paramètres suivants de l'algorithme DGML sont fixés aux valeurs suivantes :

- $w_0 = \{0.5\}^K$
- $\eta_0 = \{10^{-2}\}^K$ avec une multiplication par 0.5, resp. 1.2, en cas de changement, resp. non changement, de signe entre deux gradient successifs
- Max $= 100$ le nombre maximal d'itérations (i.e. le nombre de mini-lots présentés). On a vérifié que ce nombre n'avait jamais été atteint pour les 36 bases testées.
- Tol $= 5$ le nombre de dégradations successives autorisées.

On considère qu'il y a amélioration du critère pour une amélioration d'au moins $\epsilon = 10^{-4}$ de la valeur précédente du critère. Les poids inférieurs à un seuil égal à 10^{-3} sont mis à 0.

Base	Ni	Nv	Nc	Base	Ni	Nv	Nc
Abalone	4177	8	28	Mushroom	8416	22	2
Adult	48842	15	2	PenDigits	10992	16	10
Australian	690	14	2	Phoneme	2254	256	5
Breast	699	10	2	Pima	768	8	2
Bupa	345	6	2	Satimage	768	8	6
Crx	690	15	2	Segmentation	2310	19	7
Flag	194	29	8	Shuttle	58000	9	7
German	1000	24	2	SickEuthyroid	3163	25	2
Glass	214	10	6	Sonar	208	60	2
Heart	270	13	2	Soybean	376	35	19
Hepatitis	155	19	2	Spam	4307	57	2
Horsecolic	368	27	2	Thyroid	7200	21	3
Hypothyroid	3163	25	2	Tictactoe	958	9	2
Ionosphre	351	34	2	Vehicle	846	18	4
Iris	150	4	3	Waveform	5000	21	3
LED	1000	7	10	WaveformNoise	5000	40	3
LED17	10000	24	10	Wine	178	13	3
Letter	20000	16	26	Yeast	1484	9	10

TAB. 1 – *Caractéristiques des 36 bases de l'UCI : Ni=nombre d'instances, Nv=Nombre initial de variables , Nc=Nombre de classes.*

Toutes les expérimentations ont été menées en 10-fold-cross-validation sur les 36 bases de l'UCI décrites dans le tableau 1. Dans la présentation des résultats 'SNB' désigne la performance d'un classifieur de Bayes moyenné à l'aide du taux de compression (Boullé, 2007).

4.1 Expérimentations sur la qualité de l'optimisation

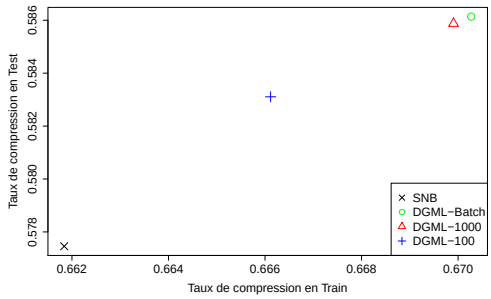

FIG. 1 – *Taux de compression moyen en Train et en Test pour* 36 *bases de l'UCI*

Tout d'abord, nous avons étudié les performances de l'algorithme DGML, c'est à dire de l'algorithme de descente de gradient projeté sans post-optimisation, en fonction de la taille des mini-lots L. Nous avons choisi comme indicateur de la qualité de l'optimisation le taux

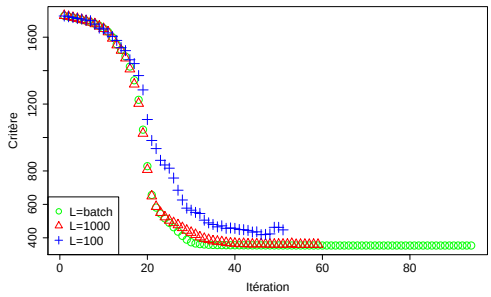

FIG. 2 – *Chemins de convergence du critère selon la taille des mini-lots (DGML) au cours des itérations pour la base Phoneme*

de compression qui mesure le logarithme négatif de la vraisemblance du modèle, normalisé par l'entropie de Shannon. Plus ce taux est proche de 1, plus la vraisemblance du modèle est élevée. Pour des modèles moins performants que le modèle aléatoire, le taux de compression est négatif. La valeur du taux sur les données d'apprentissage est donc un bon indicateur de la qualité de l'optimisation obtenue étant donné que le critère non régularisé est réduit à la log vraisemblance négative.
La figure 1 présente le taux de compression obtenu en Train et en Test et moyenné sur les 36 bases de l'UCI pour différentes tailles de mini-lots $L = 100, 1000, N$. Dans le dernier cas, le choix $L = N$ revient à un algorithme de type batch. Les taux de compression obtenus en Train et en Test pour le SNB MODL (Boullé, 2007) servent ici de référence. Les résultats obtenus indiquent que, plus la taille des mini-lots est petite, plus la qualité de l'optimisation se dégrade. D'autre part, les résultats obtenus pour $L = 1000$ et $L = N$ sont très proches. Le taux de compression en Train est significativement meilleur en batch que pour $L = 1000$ pour 8 des 36 bases.
La figure 2 présente à titre illustratif la série des valeurs prises par le critère au cours de l'optimisation selon la valeur de $L = 100, 1000, N$ pour la base Phoneme. Sur l'ensemble des 36 bases la convergence est plus rapide mais plus chaotique lorsque la taille des mini-lots diminue.

Nous avons ensuite comparé la qualité de l'optimisation pour l'algorithme DGML sans post-optimisation d'une part et pour un algorithme DGML post-optimisé d'autre part. Plusieurs types de post-optimisations ont été testées : par multi-start (DGML-MS) ou par perturbation aléatoire à voisinage variable (DGML-VNS).
Pour avoir une complexité du même ordre de grandeur que celle de l'algorithme de prétraitement univarié MODL, à savoir $O(K * N * \log(K * N))$, on a fixé le nombre total d'itérations autorisées T proportionnel à $\log(K * N)$. Plus précisément, on a choisi $T = \log(K * N) * 2^{\text{PostOptiLevel}}$ où PostOptiLevel est un entier qui permet de régler le niveau de post-optimisation souhaité.
Pour chacun de ces deux types de post-optimisation, on a étudié l'influence de la valeur du niveau d'optimisation OptiLevel $= 3, 4, 5$. Dans la mesure où l'algorithme post-optimisé mémorise au fur et à mesure la meilleure solution rencontrée, la post-optimisation ne peut qu'améliorer le taux de compression en Train. On a donc dans un premier temps mesuré si

cette amélioration était significative ou pas. Pour une post-optimisation de type MS, le taux de compression en Train est amélioré de manière significative pour $7, 16, 18$ des 36 bases avec un niveau d'optimisation respectivement de $3, 4, 5$. Pour une post-optimisation de type VNS, le taux de compression en Train est amélioré de manière significative pour $18, 19, 23$ des 36 bases avec un niveau d'optimisation respectivement de $3, 4, 5$. La post-optimisation de type VNS semble donc préférable à la post-optimisation MS : l'exploration guidée par un voisinage de taille variable à partir du meilleur minima rencontré permet une recherche plus fructueuse des autres minimas qu'une exploration purement aléatoire.
La figure 3 permet d'illustrer ce phénomène de "gaspillage" d'itérations en mode MS au début

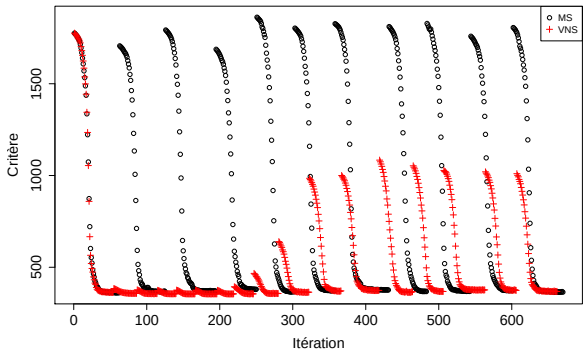

FIG. 3 – *Chemins de convergence du critère selon la post-optimisation pour la base Phonème et pour un niveau d'optimisation égal à* 5

de chaque nouveau lancement pour la base.

Les expérimentations présentées dans cette section ont mis en évidence : l'effet de la taille des mini-lots sur la qualité de l'optimisation. Plus cette taille est élevée meilleure est la qualité de l'optimisation ; l'intérêt de la post-optimisation VNS comparé notamment à une post-optimisation MS. Nous retenons donc pour la suite des expérimentations un algorithme de type DGML-VNS avec une taille de mini-lots fixée à $L = 1000$ et un niveau d'optimisation PostOptiLevel $= 5$.

4.2 Performances du classifieur régularisé

Nous présentons les performances du classifieur en fonction du paramétrage pour le poids de la régularisation λ et l'exposant p de la fonction $|w_k|^p$. Trois valeurs ont été testées pour $\lambda = 0.01, 0.1, 0.5$ et pour $p = 0.5, 1, 2$. Les performances en terme d'AUC des neuf prédicteurs régularisés associés à ces valeurs sont présentées figure 4 avec pour référence les performances du prédicteur non régularisé obtenu pour $\lambda = 0$ et du prédicteur SNB.

Pour la valeur la plus élevée du poids de régularisation, à savoir $\lambda = 0.5$ (en violet sur la figure), les performances en terme d'AUC sont dégradées par rapport aux performances obtenues sans régularisation (en rond rouge sur la figure) quelle que soit la valeur de p. En re-

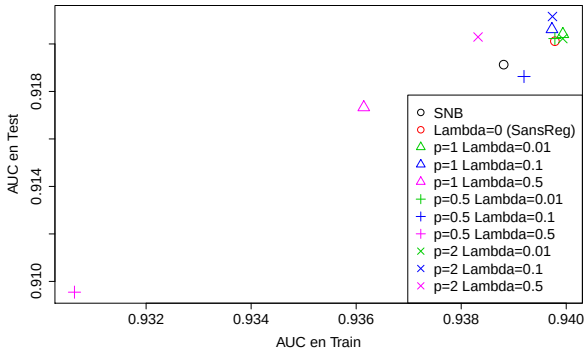

FIG. 4 – *AUC moyenne en Train et en Test pour* 36 *bases de l'UCI en fonction du poids de la régularisation et de son type*

vanche, pour les autres valeurs de poids $\lambda = 0.01$ et $\lambda = 0.1$, les performances sont similaires pour toutes les valeurs de p et légèrement supérieures ou identiques en moyenne à celle du prédicteur sans régularisation. Ces deux valeurs du poids de la régularisation conduisent donc à des performances statistiques équivalentes à celle du prédicteur non régularisé.

D'autre part, pour étudier la parcimonie des prédicteurs obtenus, la figure 5 présente le nombre de variables retenues et la somme de leur poids. On voit tout d'abord que plus p est faible plus le nombre de poids non nuls est faible également. La régularisation quadratique est celle qui conduit aux prédicteurs les moins parcimonieux. Parmi la régularisation en valeur absolue ($p = 1$) et celle en racine carrée ($p = 0.5$), c'est la seconde qui permet une réduction la plus importante du nombre de variables retenues. Concernant la somme des poids des variables retenues, tous les types de régularisation permettent de réduire en moyenne la somme des poids. D'autre part, à poids λ donné, la régularisation quadratique a un impact moins important sur la réduction de la somme des poids que les deux autres régularisations dont les performances sont très proches pour cet indicateur.
En prenant en compte ces deux aspects de performance statistique et de parcimonie, le compromis $p = 1$ et $\lambda = 0.1$ semble le plus favorable. Sans dégrader les performances du prédicteur non régularisé, il permet de réduire de façon importante le nombre de variables sélectionnées ce qui rend le prédicteur plus interprétable et moins complexe à déployer.

5 Conclusion

Nous avons proposé une régularisation parcimonieuse de la log vraisemblance d'un classifieur Bayésien naïf pondéré. Nous avons décrit et expérimenté un algorithme de descente de gradient intégrant en ligne des mini-lots de données et optimisant les poids de ce classifieur par exploration plus ou moins poussée de la solution optimale courante en fonction d'un budget donné. Les expérimentations menées ont permis de montrer l'intérêt de l'introduction de ces

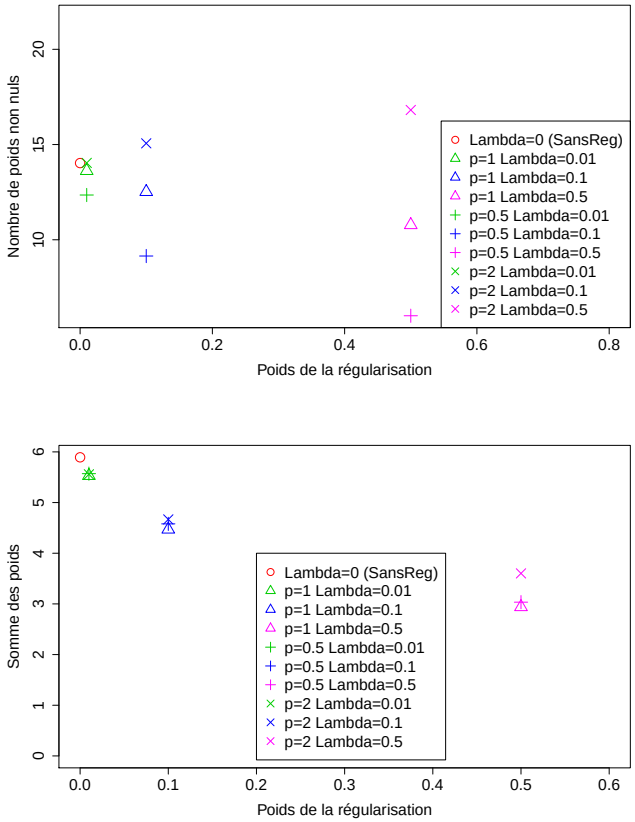

FIG. 5 – *Nombre de variables retenues et somme de leur poids moyennés pour* 36 *bases de l'UCI en fonction du poids de la régularisation et de son type.*

mini-lots et de la post-optimisation. D'autre part, une étude de paramétrage de la régularisation a fait ressortir que le choix optimal correspondait à un terme de régularisation en norme $L1$ avec un poids $\lambda = 0.1$ qui permet d'obtenir un prédicteur parcimonieux et performant. Des expérimentations sur des jeux de données beaucoup plus volumineux sont nécessaires pour tester les performances de l'approche sur de réels flux de données et feront le cadre de travaux futurs.

Références

Bertsekas (1976). On the Goldstein-Levitin-Polyak gradient projection method. *IEEE Transactions on Automatic Control*.

Boullé (2007). Compression-based averaging of selective naive bayes classifiers. *Journal of Machine Learning Research 8*, 1659–1685.

Boullé (2007). *Recherche d'une représentation des données efficace pour la fouille des grandes bases de données*. Ph. D. thesis, Ecole Nationale Supérieure des Télécommunications.

Dekel, Gilad-Bachrach, Shamir, et Xiao (2012). Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research 13*, 165–202.

Guigourès et Boullé (2011). Optimisation directe des poids de modèles dans un prédicteur bayésien naif moyenné. In *13èmes Journées Francophones "Extraction et Gestion de Connaissances" (EGC 2011)*, pp. 77–82.

Hansen et Mladenovic (2001). Variable neighborhood search : Principles and applications. *European Journal of Operational Research 130*.

Hoeting, Madigan, Raftery, et Volinsky (1999). Bayesian model averaging : A tutorial. *Statistical Science 14*(4), 382–401.

Riedmiller et Braun (1993). A direct adaptive method for faster backpropagation learning : The Rprop algorithm. In *IEEE International Conference on Neural Networks*, pp. 586–591.

Summary

We consider supervised classification for data streams with a high number of input variables. The basic naïve Bayes classifier is attractive for its simplicity and performance when the strong assumption of conditional independence is valid. Variables selection and models averaging are two common ways to improve this model. This process amounts to manipulate weighted naïve Bayes classifiers. In this article, we focus on direct estimation of weighted naïve Bayes classifiers. We propose a sparse regularization of the model log-likelihood which takes into account the information contained in each input variable. The sparse regularized likelihood being non convex, we propose an online gradient algorithm using mini-batches and a post-optimization to avoid local minima. Experiments study first the optimization quality and then the classifier performance according to its parameterization and show the effectiveness of our approach.