L'apprentissage statistique à grande échelle

Léon Bottou[1], Olivier Bousquet[2]

[1] NEC Labs America, Princeton, USA
[2] Google, Zurich, Suisse

Résumé Depuis une dizaine d'années, la taille des données croit plus vite que la puissance des processeurs. Lorsque les données disponibles sont pratiquement infinies, c'est le temps de calcul qui limite les possibilités de l'apprentissage statistique. Ce document montre que ce changement d'échelle nous conduit vers un compromis qualitativement différent dont les conséquences ne sont pas évidentes. En particulier, bien que la descente de gradient stochastique soit un algorithme d'optimisation médiocre, on montrera, en théorie et en pratique, que sa performance est excellente pour l'apprentissage statistique à grande échelle.

1 Introduction

La théorie de l'apprentissage statistique prend rarement en compte le coût des algorithmes d'apprentissage. Vapnik [1] ne s'y intéresse pas. Valiant [2] exclue les algorithmes d'apprentissage dont le coût croit exponentiellement. Cependant, malgré de nombreux progrès sur les aspects statistiques [3, 4], peu de résultats concerne la complexité des algorithmes d'apprentissage (e.g., [5].)

Ce document développe une idée simple : une optimisation approximative est souvent suffisante pour les besoins de l'apprentissage. La première partie reprend la décomposition de l'erreur de prévision proposée dans [6] dans laquelle un terme supplémentaire qui décrit les conséquences de l'optimisation approximative. Dans le cas de l'apprentissage à petite échelle, cette décomposition décrit le compromis habituel entre approximation et estimation. Dans le cas de l'apprentissage à grande échelle, elle décrit une situation plus complexe qui dépend en particulier du coût de calcul associé à l'algorithme d'apprentissage. La seconde partie explore les propriétés asymptotiques de l'apprentissage à grande échelle lorsque l'on utilise diverses méthodes d'optimisation. Ces résultats montrent clairement que le meilleur algorithme d'optimisation n'est pas nécessairement le meilleur algorithme d'apprentissage. Finalement, cette analyse est confirmée par quelques comparaisons expérimentales.

2 Optimisation approximative

Comme [7, 1], considérons un espace de paires entrées-sorties $(x, y) \in \mathcal{X} \times \mathcal{Y}$ équippé d'une loi jointe de probabilité $P(x, y)$. La loi conditionelle $P(y|x)$ représente la relation inconnue qui lie entrées et sorties. Une fonction de perte $\ell(\hat{y}, y)$ mesure l'écart entre la sortie prédite $\hat{y}$ et la sortie observée y. Notre objectif est la fonction f^* qui minimise le *risque moyen*

$$E(f) = \int \ell(f(x), y)\, dP(x, y) = \mathbb{E}\left[\ell(f(x), y)\right],$$

c'est à dire,

$$f^*(x) = \arg\min_{\hat{y}} \mathbb{E}\left[\ell(\hat{y}, y) | \, x\right].$$

Bien que la distribution $P(x, y)$ soit inconnue, on nous donne un échantillon $\mathcal{S}$ composé de n exemples d'apprentissage (x_i, y_i), $i = 1 \ldots n$ tirés indépendemment de cette loi inconnue. Nous pouvons alors définir le *risque empirique*

$$E_n(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i) = \mathbb{E}_n[\ell(f(x), y)].$$

Notre principe d'apprentissage consiste ensuite à choisir une famille $\mathcal{F}$ de fonctions de prévision, et à rechercher celle qui minimise le risque empirique : $f_n = \arg\min_{f \in \mathcal{F}} E_n(f)$. Lorsque la famille $\mathcal{F}$ est suffisamment restrictive cette approche est justifiée par des résultats combinatoires bien connus [1]. Comme il n'est pas vraisemblable que la famille de fonction $\mathcal{F}$ contienne la fonction optimale f^*, nous appelons $f^*_{\mathcal{F}} = \arg\min_{f \in \mathcal{F}} E(f)$ la meilleure fonction au sein de la famille $\mathcal{F}$. Afin de simplifier l'analyse, nous supposerons que f^*, $f^*_{\mathcal{F}}$ et f_n sont bien définies et uniques.

Nous pouvons alors décomposer l'excédent d'erreur

$$\begin{aligned} \mathbb{E}\left[E(f_n) - E(f^*)\right] &= \underbrace{\mathbb{E}\left[E(f^*_{\mathcal{F}}) - E(f^*)\right]}_{\mathcal{E}_{\text{app}}} + \underbrace{\mathbb{E}\left[E(f_n) - E(f^*_{\mathcal{F}})\right]}_{\mathcal{E}_{\text{est}}} , \\ &= \mathcal{E}_{\text{app}} + \mathcal{E}_{\text{est}} \end{aligned} \tag{1}$$

où les espérances s'entendent par rapport au tirage aléatoire des exemples d'apprentissage. L'*erreur d'approximation* $\mathcal{E}_{\text{app}}$ mesure comment la solution optimale f^* peut être approchée par la famille de fonction $\mathcal{F}$. L'*erreur d'estimation* $\mathcal{E}_{\text{est}}$ mesure ce que l'on perd en minimisant le risque empirique $E_n(f)$ à la place du risque moyen $E(f)$. L'erreur d'estimation est déterminée par le nombre n d'exemples d'apprentissage et par la *capacité* de la famille de fonctions [1]. Choisir une famille plus grande[3] réduit l'erreur d'approximation mais augmente l'erreur d'estimation. Ce compromis a été étudié en détail [1, 3, 8, 9].

2.1 Erreur d'optimization

Il est souvent coûteux de calculer f_n en minimisant le risque empirique $E_n(f)$. Comme le risque empirique $E_n(f)$ est déjà une approximation du risque moyen $E(f)$, il ne devrait pas être nécessaire d'accomplir cette minimisation avec une très grande précision. Il est souvent tentant de stopper un algorithme itératif d'optimisation avant qu'il ne converge.

Supposons qu'un algorithme approximatif d'optimisation calcule une solution approchée $\tilde{f}_n$ qui minimise la fonction de coût avec une tolérance $\rho > 0$ prédéfinie.

$$E_n(\tilde{f}_n) \leq E_n(f_n) + \rho$$

On peut alors décomposer l'excédent d'erreur $\mathcal{E} = \mathbb{E}\big[E(\tilde{f}_n) - E(f^*)\big]$ en trois termes :

$$\begin{aligned} \mathcal{E} &= \underbrace{\mathbb{E}\left[E(f^*_{\mathcal{F}}) - E(f^*)\right]}_{\mathcal{E}_{\text{app}}} + \underbrace{\mathbb{E}\left[E(f_n) - E(f^*_{\mathcal{F}})\right]}_{\mathcal{E}_{\text{est}}} + \underbrace{\mathbb{E}\big[E(\tilde{f}_n) - E(f_n)\big]}_{\mathcal{E}_{\text{opt}}} . \\ &= \mathcal{E}_{\text{app}} + \mathcal{E}_{\text{est}} + \mathcal{E}_{\text{opt}} \end{aligned} \tag{2}$$

[3] On considère souvent une série de familles de fonctions de la forme $F_B = \{f \in \mathcal{H},\ \Omega(f) \leq B\}$. Pour chaque valeur de l'hyperparamètre B, la fonction f_n est obtenue en minimisant le *risque empirique régularisé* $E_n(f) + \lambda\Omega(f)$ avec une valeur appropriée du coefficient de Lagrange λ. On peut alors ajuster le compromis estimation–approximation en choisissant λ au lieu de B.

Nous appelons *erreur d'optimisation* le terme supplémentaire $\mathcal{E}_{\text{opt}}$ qui mesure l'impact de l'optimization approximative sur l'erreur de prévision. L'ordre de grandeur de ce terme est bien sûr comparable à celui de la tolérance ρ (see section 3.1.)

2.2 Le compromis approximation–estimation–optimisation

Cette décomposition décrit un compromis plus complexe car il concerne trois variables et deux contraintes. Les contraintes sont le nombre maximal d'exemples et le temps calcul maximal disponibles pour l'apprentissage. Les variables sont la taille de la famille de fonction $\mathcal{F}$, la tolérance de l'optimisation ρ, et le nombre n d'exemples effectivement utilisés pour l'apprentissage. Cela conduit au programme suivant :

$$\min_{\mathcal{F},\rho,n} \quad \mathcal{E} = \mathcal{E}_{\text{app}} + \mathcal{E}_{\text{est}} + \mathcal{E}_{\text{opt}} \quad \text{sous les contraintes} \quad \left\{ \begin{array}{rcl} n & \leq & n_{\max} \\ T(\mathcal{F},\rho,n) & \leq & T_{\max} \end{array} \right. \tag{3}$$

Le nombre effectif d'exemples d'apprentissage est une variable parce qu'il peut être impossible de mener à bien l'optimisation sur tous les exemples dans le temps imparti. Cela se produit souvent en pratique. La table 1 résume comment les termes du programme (3) varient habituellement lorsque les variables $\mathcal{F}$, n, ou ρ augmentent.

TAB. 1 – Variations des termes de (3) lorsque $\mathcal{F}$, n, ou ρ augmentent.

		$\mathcal{F}$	n	ρ
$\mathcal{E}_{\text{app}}$	(approximation error)	$\searrow$		
$\mathcal{E}_{\text{est}}$	(estimation error)	$\nearrow$	$\searrow$	
$\mathcal{E}_{\text{opt}}$	(optimization error)	$\cdots$	$\cdots$	$\nearrow$
T	(computation time)	$\nearrow$	$\nearrow$	$\searrow$

La solution du programme (3) change considérablement selon que la contrainte active est celle concernant le nombre d'exemples $n < n_{\max}$ ou celle concernant le temps d'apprentissage $T < T_{\max}$.

- Nous parlerons d'*apprentissage à petite échelle* lorsque le programme (3) est contraint par le nombre maximal d'exemples $n_{\max}$. Comme le temps de calcul n'est pas limité, nous pouvons choisir ρ tres petit et réduire l'erreur d'optimisation $\mathcal{E}_{\text{opt}}$ à un niveau insignifiant. L'excédent d'erreur est alors déterminé par les erreurs d'approximation et d'estimation. Il suffit alors de prendre $n = n_{\max}$ pour retrouver le compromis approximation–estimation habituel.
- Nous parlerons d'*apprentissage à grande échelle* lorsque le programme (3) est contraint par le temps de calcul maximal $T_{\max}$. Il est alors possible d'atteindre une meilleure erreur de prévision avec une optimisation approximative – c'est à dire en choisissant $\rho > 0$ – car le temps de calcul ainsi économisé nous permet de traiter un plus grand nombre d'exemples d'apprentissage dans le temps imparti $T_{\max}$. Cela dépend bien sûr de la forme exacte de la fonction $T(\mathcal{F},\rho,n)$ pour l'algorithme d'optimisation retenu.

3 Analyse asymptotique

Dans la section précédente, nous avons étendu le compromis statistique habituel afin de tenir compte de l'erreur d'optimisation. Nous avons défini précisément la différence entre apprentissage à petite et grande échelle. Ce dernier cas est substantiellement différent parce qu'il faut tenir compte du coût de calcul de l'algorithme utilisé. Afin de clarifier les compromis de l'apprentissage à grande échelle, nous allons faire plusieurs simplifications. Il est possible de donner des analyses plus précises dans des cas particuliers [10].

- Nous travaillons avec des bornes supérieures des erreurs d'approximation et d'estimation (2). Ces bornes donnent en fait une bonne idée des vitesses de convergence [11, 12, 13, 14], à un facteur constant près.
- Nous étudions seulement les propriétés *asymptotiques* de la décomposition (2). Pour résoudre le programme (3) il nous suffira alors de faire en sorte que les trois erreurs décroissent avec des vitesses asymptotiques comparables.
- Nous considérons une famille de fonction $\mathcal{F}$ fixée et nous ignorons donc l'erreur d'approximation $\mathcal{E}_{\text{app}}$. Cette partie du compromis recouvre des réalités pratiques aussi diverses que choisir les modèles ou choisir les variables explicatives. Discuter ces pratiques dépasse les objectifs de ce document.
- Finalement, toujours pour simplifier l'analyse, nous supposons que la famille de fonctions $\mathcal{F}$ est paramétrée linéairement par un vecteur $w \in \mathbb{R}^d$. Nous supposons également que x, y et w sont bornés. Il existe alors une constante B telle que $0 \leq \ell(f_w(x), y) \leq B$ et $\ell(\cdot, y)$ sont des fonctions Lipschitziennes.

Après avoir montré comment la convergence uniforme des bornes traditionelles permet de prendre l'erreur d'optimisation en compte, nous comparerons les propriétés asymptotiques de quelques algorithmes d'apprentissage.

3.1 Convergence des erreurs d'estimation et d'optimisation

L'erreur d'optimization $\mathcal{E}_{\text{opt}}$ depend directement de la tolérance d'optimisation ρ. Cependant, cette tolérance borne la quantité empirique $E_n(\tilde{f}_n) - E_n(f_n)$ alors que l'erreur d'optimisation concerne sa contre-partie espérée $E(\tilde{f}_n) - E(f_n)$. Cette section discute l'impact de l'erreur d'optimisation $\mathcal{E}_{\text{opt}}$ et de la tolérance d'optimisation ρ sur les bornes qui s'appuient sur les concepts de convergence uniforme inventés par Vapnik et Chervonenkis [1]. Dans la suite, nous utiliserons la lettre c pour désigner toute constante positive. En particulier, deux occurences successives de la lettre c peuvent représenter des constantes de valeur différentes.

3.1.1 Le cas général

Comme la dimension de Vapnik-Chervonenkis [1] d'une famille de fonction paramétrée linéairement par $w \in \mathbb{R}^d$ est $d+1$, nous pouvons directement écrire

$$\mathbb{E}\left[\sup_{f\in\mathcal{F}} |E(f) - E_n(f)|\right] \leq c\sqrt{\frac{d}{n}},$$

où l'espérance s'entend vis-à-vis du tirage aléatoire de l'ensemble d'apprentissage.[4] Ce résultat donne immédiatement une borne sur l'erreur d'estimation :

$$\begin{aligned}\mathcal{E}_{\text{est}} &= \mathbb{E}\left[\, \big(E(f_n) - E_n(f_n)\big) + \big(E_n(f_n) - E_n(f^*_{\mathcal{F}})\big) + \big(E_n(f^*_{\mathcal{F}}) - E(f^*_{\mathcal{F}})\big) \,\right] \\ &\leq 2\,\mathbb{E}\left[\, \sup_{f\in\mathcal{F}} |E(f) - E_n(f)| \,\right] \;\leq\; c\sqrt{\frac{d}{n}}.\end{aligned}$$

On peut aussi obtenir une borne sur les erreurs combinées d'estimation et d'optimisation :

$$\begin{aligned}\mathcal{E}_{\text{est}} + \mathcal{E}_{\text{opt}} &= \mathbb{E}\big[E(\tilde{f}_n) - E_n(\tilde{f}_n)\big] + \mathbb{E}\big[E_n(\tilde{f}_n) - E_n(f_n)\big] \\ &+ \mathbb{E}\left[E_n(f_n) - E_n(f^*_{\mathcal{F}})\right] + \mathbb{E}\left[E_n(f^*_{\mathcal{F}}) - E(f^*_{\mathcal{F}})\right] \\ &\leq c\sqrt{\frac{d}{n}} + \rho + 0 + c\sqrt{\frac{d}{n}} \;=\; c\left(\rho + \sqrt{\frac{d}{n}}\right).\end{aligned}$$

Malheureusement, il est bien connu que cette vitesse de convergence est trop pessimiste pour un grand nombre de cas. Des bornes plus raffinées sont donc nécessaires.

3.1.2 Le cas réalisable

Quand la fonction de perte $\ell(\hat{y}, y)$ est positive, pour tout $\tau > 0$, les bornes relatives de convergence uniforme [1] affirment avec probabilité $1 - e^{-\tau}$ que

$$\sup_{f\in\mathcal{F}} \frac{E(f) - E_n(f)}{\sqrt{E(f)}} \leq c\sqrt{\frac{d}{n}\log\frac{n}{d} + \frac{\tau}{n}}\,.$$

Ce résultat est très utile car il décrit une convergence accélérée $\mathcal{O}(\log n/n)$ dans le *cas réalisable*, c'est à dire lorsque $\ell(f_n(x_i), y_i) = 0$ pour tous les exemples d'apprentissage. Nous avons alors $E_n(f_n) = 0$, $E_n(\tilde{f}_n) \leq \rho$ et pouvons écrire

$$E(\tilde{f}_n) - \rho \leq c\sqrt{E(\tilde{f}_n)}\,\sqrt{\frac{d}{n}\log\frac{n}{d} + \frac{\tau}{n}}\,.$$

En interprétant ce résultat comme une inégalite polynomiale du second degré,

$$E(\tilde{f}_n) \leq c\left(\rho + \frac{d}{n}\log\frac{n}{d} + \frac{\tau}{n}\right).$$

En intégrant cette inégalite avec les technique ordinaires (voir [15] par exemple), on obtient une vitesse de convergence accélérée pour les erreurs combinées d'estimation et d'optimisation :

$$\mathcal{E}_{\text{est}} + \mathcal{E}_{\text{opt}} = \mathbb{E}\left[E(\tilde{f}_n) - E(f^*_{\mathcal{F}})\right] \leq \mathbb{E}\left[E(\tilde{f}_n)\right] = c\left(\rho + \frac{d}{n}\log\frac{n}{d}\right).$$

[4] Bien que les bornes originales de Vapnik et Chervonenkis aient la forme $c\sqrt{\frac{d}{n}\log\frac{n}{d}}$, on peut éliminer le terme logarithmique avec la technique de "chaining" (e.g., [12].)

3.1.3 Convergence accélérée

De nombreux auteurs (e.g., [12, 4, 14]) proposent des bornes avec convergence accélérée dans des conditions plus générales. On peut en effet montrer que

$$\mathcal{E}_{\text{app}} + \mathcal{E}_{\text{est}} \leq c \left(\mathcal{E}_{\text{app}} + \left(\frac{d}{n} \log \frac{n}{d} \right)^{\alpha} \right) \quad \text{for} \quad \frac{1}{2} \leq \alpha \leq 1 \,. \tag{4}$$

si la variance de la fonction de perte satisfait

$$\forall f \in \mathcal{F} \quad \mathbb{E}\left[\left(\ell(f(X), Y) - \ell(f^*_{\mathcal{F}}(X), Y) \right)^2 \right] \;\leq\; c \left(E(f) - E(f^*_{\mathcal{F}}) \right)^{2 - \frac{1}{\alpha}} . \tag{5}$$

La vitesse asymptotique de convergence de (4) est alors donnée par l'exposant α qui apparaît dans la majoration de la variance (5). Il y a deux façons importantes d'établir une telle majoration :

- La première façon exploite la convexité stricte de certaines fonctions de perte [14, théorème 12]. Par exemple, Lee et al. [16] établissent une vitesse $\mathcal{O}(\log n/n)$ quand on utilise une fonction de perte quadratique $\ell(\hat{y}, y) = (\hat{y} - y)^2$.
- La deuxième façon consite à faire des hypothèses sur la distribution des données. Par exemple, dans le cas de problèmes de reconnaissance de formes, la "condition de Tsybakov" décrit comment les distribution conditionnelles $P(y|x)$ se croisent au voisinage de la frontière de décision optimale [13, 14]. Le cas réalisable discuté section 3.1.2 en est un cas particulier.

Les techniques illustrées sections 3.1.1 et 3.1.2 permettent d'englober l'erreur d'optimisation dans ces bornes accélérées malgré leur complexité accrue. On obtient alors une borne combinée de la forme

$$\mathcal{E} = \mathcal{E}_{\text{app}} + \mathcal{E}_{\text{est}} + \mathcal{E}_{\text{opt}} = \mathbb{E}\left[E(\tilde{f}_n) - E(f^*) \right] \leq c \left(\mathcal{E}_{\text{app}} + \left(\frac{d}{n} \log \frac{n}{d} \right)^{\alpha} + \rho \right) . \tag{6}$$

Par exemple, Massart [15, théorème 4.2] donne un résultat général avec $\alpha = 1$. En combinant ce résultat avec des bornes connues sur la capacité des classes de fonctions linéairement paramétrées (e.g., [12]), on obtient

$$\mathcal{E} = \mathcal{E}_{\text{app}} + \mathcal{E}_{\text{est}} + \mathcal{E}_{\text{opt}} = \mathbb{E}\left[E(\tilde{f}_n) - E(f^*) \right] \leq c \left(\mathcal{E}_{\text{app}} + \frac{d}{n} \log \frac{n}{d} + \rho \right) . \tag{7}$$

Le lecteur intéréssé trouvera également des bornes comparables dans [17, 4].

3.2 Algorithmes d'optimisation par descente de gradient

Nous sommes maintenant en mesure de comparer les propriétés asymptotiques pour l'apprentissage de trois versions de l'optimisation par descente de gradient. Rappelons que la famille de fonction $\mathcal{F}$ est paramétrisée linéairement par $w \in \mathbb{R}^d$. Soient $w^*_{\mathcal{F}}$ et w_n les paramètres correspondant aux fonctions $f^*_{\mathcal{F}}$ et f_n définies section 2. Nous faisons l'hypothèse que la fonction $w \mapsto \ell(f_w(x), y)$ est convexe et deux fois différentiable avec des dérivées secondes continues. Grace à cette convexité, le fonction de coût empirique $C(w) = E_n(f_w)$ possède un minimum unique.

Deux matrices jouent un rôle important dans cette analyse : la matrice Hessienne H et la matrice G de covariance des gradients à l'optimum empirique.

$$H \;=\; \frac{\partial^2 C}{\partial w^2}(w_n) \;=\; \mathbb{E}_n\left[\frac{\partial^2 \ell(f_{w_n}(x),y)}{\partial w^2}\right], \tag{8}$$

$$G \;=\; \mathbb{E}_n\left[\left(\frac{\partial \ell(f_{w_n}(x),y)}{\partial w}\right)\left(\frac{\partial \ell(f_{w_n}(x),y)}{\partial w}\right)'\right]. \tag{9}$$

Afin de résumer l'information contenue dans ces deux matrices, nous supposons qu'il existe des constantes $\lambda_{\max} \geq \lambda_{\min} > 0$ et $\nu > 0$ telles que l'on puisse choisir, pour tout $\eta > 0$, un nombre d'exemples suffisament grand pour faire en sorte que l'assertion suivante soit vraie avec une probabilité plus grande que $1-\eta$:

$$\mathrm{tr}(G\,H^{-1}) \leq \nu \qquad \text{et} \qquad \mathrm{Spectre}(H) \subset [\,\lambda_{\min}\,,\,\lambda_{\max}\,] \tag{10}$$

Le rapport de conditionnement $\kappa = \lambda_{\max}/\lambda_{\min}$ est un bon indicateur de la difficulté du problème d'optimisation [18]. L'hypothèse $\lambda_{\min} > 0$ permet d'éviter des complications dans le cas de l'algorithme de gradient stochastique. Cette condition n'implique une convexité stricte que dans un voisinage de l'optimum. Si la fonction de coût était partout convexe, l'argument de [14, théorème 12] donnerait une convergence accélérée avec $\alpha \approx 1$ pour (4) et (6). Ce n'est pas le cas, par exemple, lorsque la fonction de perte ℓ est obtenue en lissant localement une fonction $\ell(z,y) = \max\{0, 1-yz\}$. La fonction de coût est alors linéaire par morceaux avec des coins et des arêtes lissés. Bien que cette fonction ne soit pas partout strictement convexe, son optimum est vraisemblablement atteint sur un coin lissé avec une matrice Hessienne non singulière.

Les trois algorithmes que nous étudions dans cette section utilisent de l'information sur le gradient de la fonction de coût $C(w)$ pour mettre à jour itérativement leur estimée courante $w(t)$ du vecteur de paramètres optimaux.

– La **Descente de Gradient (GD)** consiste à itérer

$$w(t+1) \;=\; w(t) - \eta\frac{\partial C}{\partial w}(w(t)) \;=\; w(t) - \eta\frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial w}\ell\big(f_{w(t)}(x_i),y_i\big)$$

où le gain $\eta > 0$ est suffisament faible. L'algorithm GD possède une vitesse de convergence "linéaire" [18]. Lorsque $\eta = 1/\lambda_{\max}$, cet algorithme atteint une précision ρ après $\mathcal{O}(\kappa\log(1/\rho))$ itérations. Le nombre exact d'itérations depend bien sûr du choix des paramètres initiaux $w(0)$.

– La **Descent de Gradient de Second Ordre (2GD)** consiste à itérer

$$w(t+1) \;=\; w(t) - H^{-1}\frac{\partial C}{\partial w}(w(t)) \;=\; w(t) - \frac{1}{n}H^{-1}\sum_{i=1}^{n}\frac{\partial}{\partial w}\ell\big(f_{w(t)}(x_i),y_i\big)$$

où la matrice H^{-1} est l'inverse de la matrice Hessienne (8). C'est un cas plus favorable que l'algorithme de Newton parce nous n'évaluons pas la matrice Hessienne locale à chaque itération, mais nous supposons simplement que la matrice Hessienne optimale nous est donnée par avance. L'algorithme 2GD possède une vitesse

de convergence "superlinéaire" et même "quadratique" [18] : une seule itération suffit lorsque le coût est quadratique. Dans le cas général, cet algorithme atteint une précision ρ après au plus $\mathcal{O}(\log\log(1/\rho))$ itérations.

– La **Descente Stochastique de Gradient (SGD)** consiste, pour chaque itération, à tirer un example d'apprentissage (x_t, y_t) au hasard, et à mettre à jour $w(t)$ sur la base du gradient de la fonction de perte pour cet exemple seulement.

$$w(t+1) \;=\; w(t) - \frac{\eta}{t}\,\frac{\partial}{\partial w}\ell\big(f_{w(t)}(x_t), y_t\big).$$

Les $w(t)$ forment donc un processus stochastique induit par le tirage aléatoire d'un nouvel exemple à chaque itération. Murata [19, section 2.2] en a calculé la moyenne $\mathbb{E}_{\mathcal{S}}[w(t)]$ et la variance $\mathbb{V}\mathrm{ar}_{\mathcal{S}}[w(t)]$. En appliquant ce résultat à la distribution discrète engendrée par l'ensemble d'apprentissage, en prenant $\eta = 1/\lambda_{\min}$, et en définissant $\delta w(t) = w(t) - w_n$, on obtient $\delta w(t)^2 = \mathcal{O}(1/t)$.
Nous pouvons alors écrire

$$\begin{aligned}
\mathbb{E}_{\mathcal{S}}[\,C(w(t)) - \inf C\,] &= \mathbb{E}_{\mathcal{S}}\big[\mathrm{tr}\big(H\,\delta w(t)\,\delta w(t)'\big)\big] + \mathrm{o}\big(\tfrac{1}{t}\big) \\
&= \mathrm{tr}\,\big(\,H\,\mathbb{E}_{\mathcal{S}}[\delta w(t)]\,\mathbb{E}_{\mathcal{S}}[\delta w(t)]' + H\,\mathbb{V}\mathrm{ar}_{\mathcal{S}}[w(t)]\,\big) + \mathrm{o}\big(\tfrac{1}{t}\big) \qquad (11) \\
&\le \tfrac{\mathrm{tr}(GH)}{t} + \mathrm{o}\big(\tfrac{1}{t}\big) \;\le\; \tfrac{\nu\kappa^2}{t} + \mathrm{o}\big(\tfrac{1}{t}\big)\,.
\end{aligned}$$

L'algorithme SGD atteint donc une précision moyenne ρ après au plus $\nu\kappa^2/\rho + \mathrm{o}(1/\rho)$ itérations. Sa convergence est en fait limitée par le bruit stochastique induit par le choix aléatoire d'un exemple unique à chaque itération. Ni la valeur initiale $w(0)$ du paramètre, ni le nombre total d'exemples d'apprentissage n'apparaissent dans cette borne. Lorsque l'ensemble d'apprentissage est grand, il est possible d'atteindre la précision ρ visée sans même avoir visité tous les exemples d'apprentissage. C'est en fait une forme de borne sur la généralisation.

Les trois premières colonnes de la table 2 donnent, pour chaque algorithme, le temps requis pour une itération, le nombre d'itérations requises pour atteindre une précision d'optimisation prédéfinie ρ, et leur produit, c'est à dire le temps requis pour atteindre cette précision. Ces résultats asymptotiques sont valides avec une probabilité 1 parce que la probabilité de leur négation est inférieure à η pour tout $\eta > 0$.

La dernière colonne majore le temps nécessaire pour ramener l'excédent d'erreur de prévision en dessous de la valeur $c\,(\mathcal{E}_{\mathrm{app}} + \varepsilon)$ où c est la constante qui apparaît dans la borne (6). On calcule cela en observant que choisir $\rho \sim \big(\frac{d}{n}\log\frac{n}{d}\big)^{\alpha}$ dans (6) donne une vitesse asymptotique de convergence optimale pour ε avec un temps de calcul minimal. On utilise ensuite les équivalences asymptotiques $\rho \sim \varepsilon$ et $n \sim \frac{d}{\varepsilon^{1/\alpha}}\log\frac{1}{\varepsilon}$.

Réciproquement, on peut considérer la valeur de ϵ pour laquelle cette dernière colonne est égale à $T_{\max}$. Cette valeur est alors le *meilleur excédent d'erreur de prévision* que chaque algorithme peut atteindre dans la limite de temps de calcul $T_{\max}$. C'est donc la solution du programme (3) pour un problème d'apprentissage à grande échelle satisfaisant nos hypothèses simplificatrices.

Ces résultats asymptotiques montrent clairement que la performance de prévision des systèmes d'apprentissage à grande échelle dépend à la fois des propriétés statistiques du modèle et des propriétés calculatoires de l'algorithme d'optimisation retenu. Cette double dépendence conduit à des résultats surprenants.

TAB. 2 – Comportement asymptotique (avec probabilité 1) des trois algorithmes de descente de gradient. Il est intéressant de comparer les deux dernières colonnes (temps requis pour optimiser avec une tolérance ρ, et temps requis pour atteindre un excédent d'erreur inférieur à ϵ). *Légende* : n nombre d'exemples ; d dimension du vecteur de paramètres ; κ, ν voir equation (10).

Algo.	**Coût d'une itération**	**Itérations pour atteindre ρ**	**Temps pour atteindre ρ**	**Temps pour atteindre $\mathcal{E} \leq c\,(\mathcal{E}_{\text{app}} + \varepsilon)$**
GD	$\mathcal{O}(nd)$	$\mathcal{O}\Big(\kappa \log \frac{1}{\rho}\Big)$	$\mathcal{O}\Big(nd\kappa \log \frac{1}{\rho}\Big)$	$\mathcal{O}\Big(\frac{d^2\,\kappa}{\varepsilon^{1/\alpha}} \log^2 \frac{1}{\varepsilon}\Big)$
2GD	$\mathcal{O}\big(d^2 + nd\big)$	$\mathcal{O}\Big(\log\log \frac{1}{\rho}\Big)$	$\mathcal{O}\Big(\big(d^2 + nd\big) \log\log \frac{1}{\rho}\Big)$	$\mathcal{O}\Big(\frac{d^2}{\varepsilon^{1/\alpha}} \log \frac{1}{\varepsilon} \log\log \frac{1}{\varepsilon}\Big)$
SGD	$\mathcal{O}(d)$	$\frac{\nu\kappa^2}{\rho} + \mathrm{o}\Big(\frac{1}{\rho}\Big)$	$\mathcal{O}\Big(\frac{d\nu\kappa^2}{\rho}\Big)$	$\mathcal{O}\Big(\frac{d\,\nu\,\kappa^2}{\varepsilon}\Big)$

– Le résultat pour SGD *ne dépend pas du coefficient* α de la vitesse asymptotique d'estimation statistique. Lorsque ce coefficient est médiocre, il est moins nécessaire d'optimiser avec précision, ce qui laisse le temps de traiter plus d'exemples.

– Utiliser un algorithme d'*optimisation superlinéaire n'améliore pas sensiblement* la vitesse de convergence asymptotique de l'excédent d'erreur de prévision ε. Bien que l'algorithme superlinéaire 2GD améliore le terme logarithmique, la vitesse d'apprentissage est dominée par le terme polynomial en $(1/\varepsilon)$. Utiliser un algorithme d'optimisation sophistiqué est souvent moins efficace que travailler les constantes d, κ et ν avec des méthodes simples de préconditionnement ou avec une meilleure implémentation [20].

– L'algorithme SGD offre à la fois *la pire vitesse d'optimisation* et *la meilleure vitesse d'apprentissage.* Cela avait été prédit théoriquement et observé expérimentalement dans le cas d'un algorithme de gradient stochastique de second ordre [21].

Au contraire, dans le cas de l'*apprentissage à petite échelle*, on peut réduire l'erreur d'optimisation $\mathcal{E}_{\text{opt}}$ à des niveaux insignifiants. La performance en prévision est alors déterminée uniquement par les propriétés statistiques du modèle.

4 Expériences

Cette section vise à confirmer les résultats précédents avec quelques expériences simples sur une tâche bien connue de catégorisation de documents. Cette tâche consiste à identifier les documents appartenant à la catégorie CCAT dans le corpus RCV1-v2 [22] à l'aide d'un séparateur à vaste marge (SVM). Les programmes utilisés et quelques résultats supplémentaires sont disponibles sur le site web `http://leon.bottou.org/projects/sgd`.

Afin d'augmenter le nombre d'exemples d'apprentissage, nous avons renversé les rôles des ensembles d'apprentissage et de test spécifiés par [22]. Nous avons ainsi 781265 exemples d'apprentissage et 23149 exemples réservés pour tester la performance en prédiction. Chaque document est représenté par 47152 variables explicatives que nous avons recalculées afin que leur normalisation ne dépende que de nos exemples d'appren-

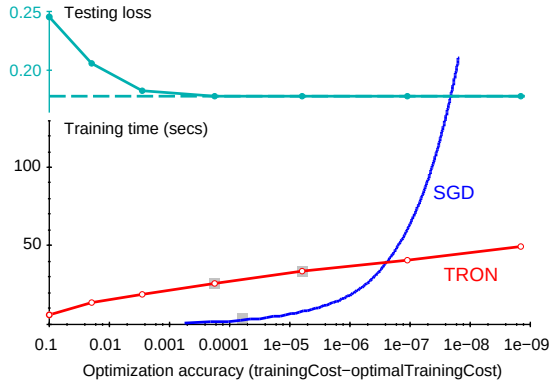

FIG. 1 – Temps d'apprentissage et coût de prévision en fonction de la précision ρ de l'optimisation pour SGD et TRON [23].

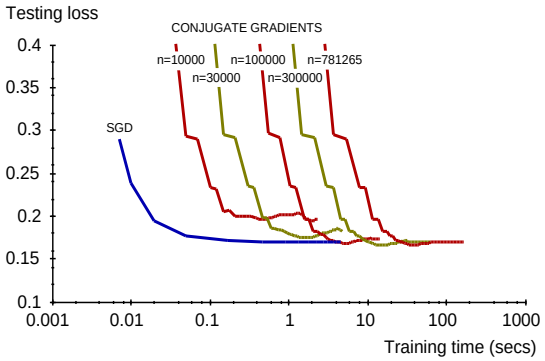

FIG. 2 – Coût de prévision en fonction du temps d'apprentissage pour SGD et pour l'algorithme CG appliqué à une fraction des exemples.

TAB. 3 – Résultats obtenus avec une SVM linéaire sur le corpus RCV1.

Model	**Algo.**	**Temps d'apprentissage**	**Coût**	**Erreur de prévision**
Coût SVM, $\lambda = 10^{-4}$ *Voir [24, 25].*	SVMLight	23,642 secs	0.2275	6.02%
	SVMPerf	66 secs	0.2278	6.03%
	SGD	**1.4** secs	0.2275	6.02%
Coût logistique, $\lambda = 10^{-5}$ *Voir [23].*	TRON (`-e .01`)	30 secs	0.18907	5.68%
	TRON (`-e .001`)	44 secs	0.18890	5.70%
	SGD	**2.3** secs	0.18893	5.66%

tissage. Nous utilisons une fonction de discrimination linéaire avec la fonction de perte "charnière" traditionnellement utilisée pour les SVMs.

$$\min_w \quad C(w,b) \;=\; \frac{\lambda}{2} + \frac{1}{n}\sum_{i=1}^{n} \ell(y_t(wx_t+b)) \quad \text{avec} \quad \ell(z) = \max\{0, 1-z\}\,.$$

Les deux première lignes de la table 3 reproduisent des résultats antérieurs [24] obtenus sur les mêmes données avec la même valeur de l'hyperparamètre λ. La troisième ligne donne les résultats obtenus avec un algorithme SGD dont chaque itération consiste à tirer un example d'apprentissage aléatoire $(x_t, y_t) \in \mathbb{R}^{47152} \times \{\pm 1\}$ et à mettre à jour le paramètre w de la fonction discriminante :

$$w_{t+1} = w_t - \eta_t \left(\lambda w + \frac{\partial \ell(y_t(wx_t+b))}{\partial w} \right) \quad \text{avec} \quad \eta_t = \frac{1}{\lambda(t+t_0)}\,.$$

Le biais b est mis à jour de façon identique. Le gain η_t retenu est une approximation du gain optimal (voir section 3.2) dans laquelle nous remplaçons la plus petite valeur propre de la matrice Hessienne par son minorant λ. Le décalage t_0 est choisi de façon à faire en sorte que le gain initial soit comparable avec la taille attendue du paramètre optimal.

Les résultats montrent sans aucun doute que l'algorithme SGD surclasse les algorithmes classiques d'optimisation des SVM. Des résultats comparables [25] ont été obtenus avec un algorithme SGD corrigé par une opération de projection. Notre résultat montre que cette opération n'est pas nécessaire.

La table 3 donne également les résultats que l'on obtient avec la fonction de perte logistique $\ell(z) = \log(1 + e^{-z})$. Comme cette fonction de perte est infiniment dérivable, nous pouvons comparer l'algorithme SGD avec un algorithme d'optimisation superlinéaire. Nous avons utilisé l'algorithme TRON [23] avec deux critères d'arrêt différents. Ces résultats apparaissent également dans la figure 1. L'algorithme TRON prend moins d'une minute pour calculer l'optimum avec 10 chiffres significatifs. Bien que l'algorithme SGD soit incapable d'une telle performance d'optimisation, il est initialement plus rapide que TRON. La partie supérieure de la figure 1 montre que l'erreur en prévision cesse de diminuer bien avant que TRON ne commence à surclasser SGD.

La figure 2 montre comment l'erreur en prévision évolue en fonction du temps d'apprentissage. Nous avons cette fois comparé l'algorithme SGD avec un algorithme de gradients conjugués (CG) appliqué à divers sous-ensembles des exemples d'apprentissage.[5] Supposons, par exemple, que nous ne disposons que d'une seconde de temps de calcul. Lancer l'algorithme CG sur seulement 30000 exemples est alors bien plus efficace que le lancer sur tout l'ensemble d'apprentissage. En revanche il est difficile de savoir à l'avance quel est le meilleur nombre d'exemples à considérer, et l'algorithme SGD appliqué à tous les exemples reste généralement plus rapide.

5 Conclusion

En prenant à la fois en compte les contraintes sur le temps de calcul et sur le nombre maximal d'exemples, nous avons mis en évidence des différences qualitatives entre les performances des systèmes d'apprentissage à petite échelle et à grande échelle. La performance en prévision d'un système d'apprentissage à grande échelle dépend à la fois des propriétés statistiques du modèle et des propriétés calculatoires de l'algorithme d'optimisation retenu. Des résultats théoriques asymptotiques et des résultats expérimentaux montrent alors qu'un algorithme d'optimisation médiocre, l'algorithme SGD, se révéle un excellent algorithme d'apprentissage à grande échelle.

Cette analyse peut donner lieu à de nombreux raffinements. Shalev-Shwartz et Srebro [10] l'ont étendu aux risques régularisés. On peut également attendre quelques effets intéressants relatifs au choix de fonctions de perte surrogées [8, 14].

Références

[1] Vladimir N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer Series in Statistics. Springer-Verlag, Berlin, 1982.

[5] Cette expérience, suggérée par Olivier Chapelle, utilise une variante de l'algorithme superlinéaire CG optimisée pour offrir une bonne vitesse initiale de convergence.

[2] Leslie G. Valiant. A theory of learnable. *Proc. of the 1984 STOC*, pages 436–445, 1984.

[3] Stéphane Boucheron, Olivier Bousquet, and Gábor Lugosi. Theory of classification : a survey of recent advances. *ESAIM : Probability and Statistics*, 9 :323–375, 2005.

[4] Peter L. Bartlett and Shahar Mendelson. Empirical minimization. *Probability Theory and Related Fields*, 135(3) :311–334, 2006.

[5] J. Stephen Judd. On the complexity of loading shallow neural networks. *Journal of Complexity*, 4(3) :177–192, 1988.

[6] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems (NIPS 2007)*, volume 20. MIT Press, 2008.

[7] Richard O. Duda and Peter E. Hart. *Pattern Classification And Scene Analysis*. Wiley and Son, 1973.

[8] Tong Zhang. Statistical behavior and consistency of classification methods based on convex risk minimization. *The Annals of Statistics*, 32 :56–85, 2004.

[9] Clint Scovel and Ingo Steinwart. Fast rates for support vector machines. In Peter Auer and Ron Meir, editors, *Proceedings of the 18th Conference on Learning Theory (COLT 2005)*, volume 3559 of *Lecture Notes in Computer Science*, pages 279–294, Bertinoro, Italy, June 2005. Springer-Verlag.

[10] Shai Shalev-Shwartz and Nathan Srebro. SVM optimization : inverse dependence on training set size. In *Proceedings of the 25th International Machine Learning Conference (ICML 2008)*, pages 928–935. ACM, 2008.

[11] Vladimir N. Vapnik, Esther Levin, and Yann LeCun. Measuring the VC-dimension of a learning machine. *Neural Computation*, 6(5) :851–876, 1994.

[12] Olivier Bousquet. *Concentration Inequalities and Empirical Processes Theory Applied to the Analysis of Learning Algorithms*. PhD thesis, Ecole Polytechnique, 2002.

[13] Alexandre B. Tsybakov. Optimal aggregation of classifiers in statistical learning. *Annals of Statististics*, 32(1), 2004.

[14] Peter L. Bartlett, Michael I. Jordan, and Jon D. McAuliffe. Convexity, classification and risk bounds. *Journal of the American Statistical Association*, 101(473) :138–156, March 2006.

[15] Pascal Massart. Some applications of concentration inequalities to statistics. *Annales de la Faculté des Sciences de Toulouse*, series 6, 9(2) :245–303, 2000.

[16] Wee S. Lee, Peter L. Bartlett, and Robert C. Williamson. The importance of convexity in learning with squared loss. *IEEE Transactions on Information Theory*, 44(5) :1974–1980, 1998.

[17] Shahar Mendelson. A few notes on statistical learning theory. In Shahar Mendelson and Alexander J. Smola, editors, *Advanced Lectures in Machine Learning*, volume 2600 of *Lecture Notes in Computer Science*, pages 1–40. Springer-Verlag, Berlin, 2003.

[18] John E. Dennis, Jr. and Robert B. Schnabel. *Numerical Methods For Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983.

[19] Noboru Murata. A statistical study of on-line learning. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

[20] Yann Le Cun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks, Tricks of the Trade*, Lecture Notes in Computer Science LNCS 1524. Springer Verlag, 1998.

[21] Léon Bottou and Yann Le Cun. Large scale online learning. In Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[22] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. RCV1 : A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5 :361–397, 2004.

[23] Chih-Jen Lin, Ruby C. Weng, and S. Sathiya Keerthi. Trust region Newton methods for large-scale logistic regression. In Zoubin Ghahramani, editor, *Proceedings of the 24th International Machine Learning Conference*, pages 561–568, Corvallis, OR, June 2007. ACM.

[24] Thorsten Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD International Conference*, Philadelphia, PA, August 2006. ACM Press.

[25] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos : Primal estimated subgradient solver for SVM. In Zoubin Ghahramani, editor, *Proceedings of the 24th International Machine Learning Conference*, pages 807–814, Corvallis, OR, June 2007. ACM.