Table des matières

Chapitre 11

Méthodes pour l'apprentissage de données massives

Nathalie Villa-Vialaneix et Fabrice Rossi

11.1 Introduction

11.1.1 Statistique et données massives

Le « big data », souvent traduit en français par *données massives* ou bien parfois *mégadonnées*, désigne l'ensemble des problèmes pour lesquels le nombre d'observations est tel que les outils classiques de gestion et traitement des données ne sont plus utilisables tels quels. Les données massives sont généralement décrites par 3 caractéristiques, auxquelles il est fait référence sous le terme de « 3V », et qui sont le **V**olume, la **V**itesse, dans le sens de vitesse d'arrivée de nouvelles données sous la forme de flux de données, et la **V**ariété qui indique que ces nouvelles données sont de types multiples, numériques, textuelles, etc. (voir the Gartner, Inc., the advisory company about information technology research[1]).

Le problème posé par des données volumineuses est double : d'une part, la plupart des approches statistiques standard ont été développées sans tenir compte des coûts de calcul et peuvent amener à des temps de mise en œuvre prohibitifs. D'autre part, les données sont parfois si volumineuses qu'elles ne peuvent être contenues dans la mémoire d'un seul ordinateur : le volume de données géré par Google est estimé en 2013 à 15 EB ($15 \times 260 \times 8$ bits)[2], ce qui correspond à 15 millions de disques durs standards d'une capacité de 1 TB). On parle alors parfois de données « à l'échelle Google » [Chamandy *et al.*, 2012].

[1] `http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Contolling-Data-Volume-Velocity-and-Variety.pdf`

[2] Source : Wikipedia, `https://en.wikipedia.org/wiki/Orders_of_magnitude_(data)`, non confirmé mais réaliste.

Sans aller jusqu'à ces échelles de volume de données, les problèmes liés aux données massives peuvent se poser dans des cas beaucoup moins complexes et sont fortement dépendant des capacités de calcul à la disposition du statisticien : Kane *et al.* [2013] font ainsi remarquer que, pour un utilisateur du logiciel R[3], par exemple, les données peuvent être considérées comme « volumineuses » si leur taille excède 20% de la mémoire vive (RAM) de l'ordinateur et « massives » si elle excède 50% de la mémoire vive. Dans les deux cas, beaucoup de méthodes d'apprentissage seront alors difficiles à mettre en œuvre.

Comme le fait remarquer Jordan [2013], dans un futur proche, un des grands enjeux de la statistique sera de s'adapter à ce type de problèmes et en particulier, à proposer des solutions pour le passage à l'échelle des méthodes statistiques et pour le contrôle de la complexité de calcul de celles-ci. Ces défis ne pourront être relevés qu'avec une collaboration étroite entre statisticiens et informaticiens, pour proposer des approches de calcul efficaces, utilisant au mieux les capacités des environnements de calcul parallèles ou répartis, tout en conservant les bonnes propriétés statistiques de convergence et d'approximation des méthodes.

11.1.2 Objectifs du chapitre

Ce chapitre ne s'intéresse pas à la mise en œuvre pratique de méthodes d'apprentissage sur données massives ni aux environnements matériels (informatiques) et logiciels permettant de gérer ce type de données. Pour une plus ample discussion sur ce point, nous renvoyons le lecteur au chapitre **??**.

Nous nous attachons, au contraire, à montrer quelques exemples de méthodes statistiques ou méthodologiques standard pouvant être utilisées pour traiter des volumes de données importants. Ce chapitre ne se veut pas une revue exhaustive des méthodes existant dans le domaine mais plutôt une illustration de quelques techniques de passage à l'échelle appliquées à des méthodes d'apprentissage présentées dans les précédents chapitres.

En particulier, différentes stratégies d'ordre méthodologique ont été développées pour le traitement de données massives que nous classons en trois grandes familles qui sont illustrées dans diverses sections de ce chapitre :

- les méthodes de *sous-échantillonnage* (section 11.2) qui utilisent un sous-ensemble réduit des données pour approcher l'analyse qui aurait été obtenue avec l'intégralité des données disponibles ;
- les méthodes de type « *diviser pour mieux régner* » (section 11.3) qui divisent les données en sous-ensembles de tailles restreintes sur lesquelles l'analyse est effectuée. Les résultats de l'ensemble des analyses sont ensuite combinés selon diverses stratégies ;

[3] `https://www.R-project.org`

- les méthodes d'apprentissage *en ligne* (section 11.4), dans lesquelles les résultats sont mis à jour par des étapes successives, chacune ayant un coût de réalisation faible.

Lorsque cela est pertinent et/ou possible, nous essayons de donner l'ordre de grandeur de la complexité des méthodes (ou du gain de complexité qu'elles apportent), mesurée en nombre d'opérations élémentaires (addition, multiplication, etc) qu'elles nécessitent. Parfois, nous donnons également des informations sur les ordres de grandeur des coûts en terme de stockage de données supplémentaires requis par la méthode.

Dans la suite, nous reprenons les notations introduites dans le chapitre **??** et nous supposons connu un échantillon d'apprentissage $D_n = (X_i,Y_i)_{1\leqslant i\leqslant n}$ qui sont des observations i.i.d. d'un couple de variables aléatoire (X,Y) de loi P dans lequel $X \in \mathcal{X}$ et $Y \in \mathcal{Y}$ où $\mathcal{Y} = \mathbb{R}$ (régression) ou bien $\mathcal{Y} = \{1, \ldots, L-1\}$ (classification supervisée). Le but est de construire un prédicteur $\widehat{f}_n : x \in \mathcal{X} \mapsto \mathcal{Y}$ à partir des observations de D_n. Les méthodes de ce chapitre sont présentées de manière privilégiée dans le cadre plus simple des problèmes de régression mais sont souvent généralisables au cadre de la classification.

11.2 Échantillonnage

Cette section est organisée en deux parties qui présentent des approches utilisant des techniques de sous-échantillonnage pour traiter des données massives :

1. la première présente le « Bag of Little Bootstrap » (BLB, Kleiner *et al.* [2012, 2014]) qui est utilisé pour mettre en œuvre des méthodes basées sur le bootstrap lorsque le nombre d'observations est grand ; par extension, nous présenterons cette méthode dans le cadre d'algorithmes de bagging (comme les forêts aléatoires, voir chapitre **??**) ;

2. la seconde présente l'approximation de Nyström [Williams and Seeger, 2000] qui est utilisée pour approcher les calculs matriciels nécessaires dans les méthodes à noyaux.

11.2.1 Bag of Little Bootstrap

Rappel sur le bagging et limites dans le cadre de données massives

Dans cette partie, nous nous intéressons à une méthode générique basée sur le bootstrap [Efron, 1979]. Nous présenterons une application de cette approche dans le cadre particulier (et simplifié) du « bagging » (« bootstrap aggregating ») [Breiman, 1996] qui correspond au cadre de la méthode des forêts aléatoires décrite dans le chapitre **??**.

Les approches de bagging consistent à tirer aléatoirement dans $\{(X_i,Y_i)\}_{i=1,\dots,n}$, B échantillons dits « bootstrap », c'est-à-dire, B échantillons de tailles n tirés avec remise. Pour tout $b = 1, \dots, B$, on notera τ_b l'ensemble des indices de $\{1, \dots, n\}$ (avec remise) correspondant aux observations de l'échantillon bootstrap numéro b et on notera $\widehat{f}^b$ l'estimateur obtenu à partir de $\{(X_i,Y_i)\}_{i\in\tau_b}$ pour résoudre le problème de régression (si $\mathcal{Y} = \mathbb{R}$) associé à l'observation de (X,Y). Le bagging consiste simplement à agréger (par la moyenne) les estimateurs obtenus dans les divers échantillons bootstrap : ainsi, dans le cadre de la régression, l'estimateur bagging de la fonction de prédiction s'écrit

$$\forall\, \mathbf{x} \in \mathcal{X}, \qquad \widehat{f}^{\text{bag}}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} \widehat{f}^b(\mathbf{x}).$$

Récemment, les approches bootstrap ont été confrontées à un double changement du point de vue de la forme des données et des capacités de calcul : d'une part, comme expliqué en introduction, la taille croissante des jeux de données a rendu la mise en œuvre de ces méthodes plus difficile, la taille des échantillons bootstrap étant identique à la taille du jeu de données initial et le nombre d'observations uniques dans ces échantillons étant de l'ordre de $0{,}63\times n$. D'autre part, les environnements informatiques répartis et de calcul parallèle se sont développés, donnant la possibilité d'accélérer considérablement la mise en œuvre des approches de type bootstrap en répartissant le calcul de chacun des estimateurs $\widehat{f}^b$ sur des processus indépendants[4]. Toutefois, cette solution ne peut être mise en œuvre que si l'estimation de $\widehat{f}^b$ elle-même est réalisable malgré la valeur de n.

Pour résoudre le problème des données de grande taille dans les approches bootstrap, une première approche consiste à utiliser des échantillons bootstrap de taille m ($m < n$) : c'est le *m parmi n bootstrap* proposé par Bickel *et al.* [1997]. Toutefois, les résultats d'un tel estimateur sont fortement dépendants de la valeur de m (en particulier, la variabilité de l'estimateur basé sur un échantillon bootstrap de taille m est différente de la variabilité d'un estimateur basé sur un échantillon de taille n) et, en pratique, les solutions proposées pour répondre à ce problème, comme celle de Bickel and Sakov [2008], se basent sur la mise en œuvre de la méthode pour diverses valeurs de m, limitant fortement le gain de calcul que pourrait amener l'utilisation d'un échantillon de taille réduite m.

Approche « Bag of Little Bootstrap »

Kleiner *et al.* [2012, 2014] proposent une approche permettant de construire des échantillons bootstrap de tailles n mais dont le coût computationnel et le coût de stockage mémoire sont réduits par rapport au bootstrap classique. La méthode est basée sur deux procédures imbriquées, comme schématisé dans la

[4] Le terme processus désigne ici une unité de calcul abstraite sur un système réparti.

figure 11.1. Tout aussi simplement que dans le bootstrap, les calculs peuvent être réalisés en parallèle. Dans une première étape, B_1 échantillons de m ($m \ll$

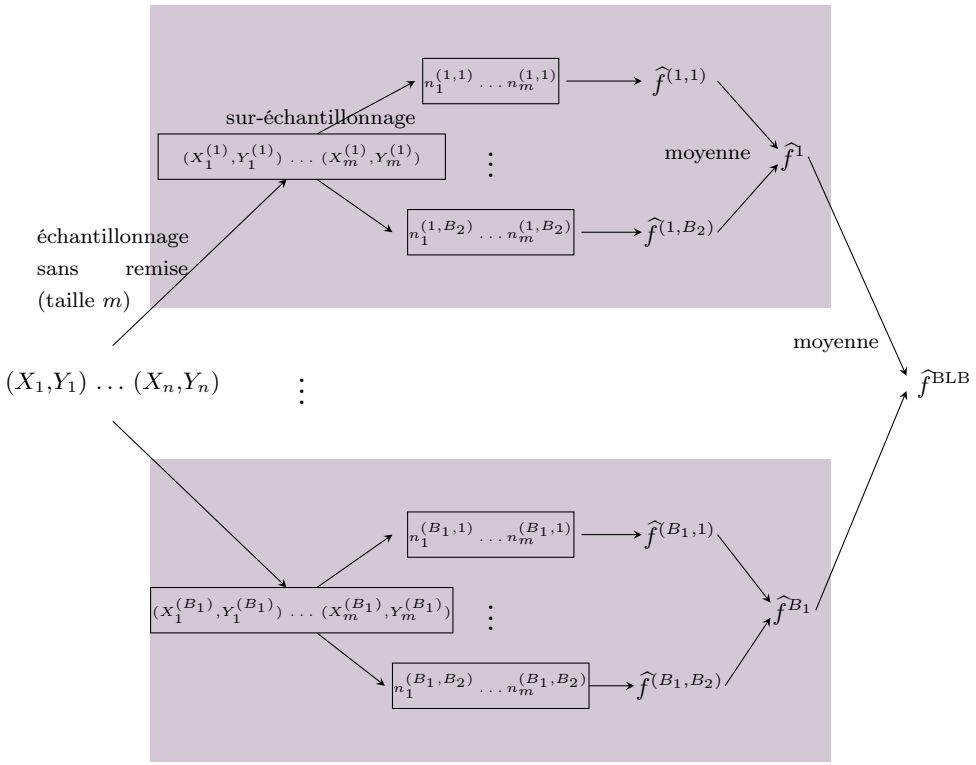

Figure 11.1 : *Représentation schématique de l'approche « Bag of Little Bootstrap ».*

n) observations, $\left\{(X_1^{(b)},Y_1^{(b)}), \ldots, (X_m^{(b)},Y_m^{(b)})\right\}$ ($b = 1, \ldots, B_1$) sont tirés sans remise, dans $\{(X_i,Y_i)\}_{i=1,\ldots,n}$. On note, pour tout $b = 1, \ldots, B_1$, τ_b les indices des observations de l'échantillon numéro b. On a donc $\operatorname{Card} \tau_b = m$ où m est généralement égal à n^γ avec $\gamma \in [0{,}5; 1]$.

Dans une seconde étape, pour chaque $b = 1, \ldots, B_1$, on affecte B_2 fois des poids à chacune des observations de τ_b de telle sorte que la somme des poids soit égale à n : on construit donc B_2 échantillons de tailles n issus des données initiales, chacun ne comportant que m observations distinctes. Pour tout $r = 1, \ldots, B_2$, cela revient à simuler des valeurs $(n_1^{(b,r)}, \ldots, n_m^{(b,r)})$ selon une loi multinomiale de paramètres n et $\frac{1}{m}\mathbf{1}_m$ (où $\mathbf{1}_m$ désigne le vecteur de taille m dont les coordonnées sont toutes égales à 1). Chacun des échantillons $\{(X_i,Y_i)\}_{i\in\tau_b}$ pour lequel les observations sont répétées, respectivement, $(n_1^{(b,r)}, \ldots, n_m^{(b,r)})$ fois est donc un sous-échantillon de taille n de l'échantillon initial, appelé *échantillon BLB*. Son coût de stockage est très réduit par rapport

à un échantillon bootstrap habituel puisque sa taille est de l'ordre de n^γ contre $0{,}63 \times n$ pour un échantillon bootstrap standard[5].

La dernière étape consiste à construire, à partir de chacun des $B_1 \times B_2$ échantillons BLB, un estimateur $\widehat{f}^{(b,r)}$. Là encore, le calcul d'un tel estimateur peut être réalisé de manière efficace dès que la méthode d'estimation choisie permet de prendre en compte de manière naturelle des observations pondérées. Dans le cadre des forêts aléatoires , par exemple, $\widehat{f}^{(b,r)}$ est basé sur le calcul des variances intra-classes de la variable Y pour des observations (X_i,Y_i) d'un sous-ensemble $\tau \subset \{1, \ldots, n\}$ de l'ensemble de départ (qui correspondent aux observations affectées au nœud courant à partitionner), dans lequel les groupes sont définis par $\{X^j \geqslant d\}$ et par $\{X^j < d\}$. Si un de ces ensembles contient les observations $i \in \tau$, répétées chacune n_i fois alors la variance intra-classe s'exprime simplement comme

$$\frac{1}{\sum_{i\in\tau} n_i} \sum_{i\in\tau} n_i (Y_i - \overline{Y})^2 \qquad \text{avec } \overline{Y} = \frac{1}{\sum_{i\in\tau} n_i} \sum_{i\in\tau} n_i Y_i.$$

Enfin, les résultats sont agrégés comme suit (dans un cadre de régression) :

$$\forall\, b = 1, \ldots, B_1, \qquad \widehat{f}^b = \frac{1}{B_2} \sum_{r=1}^{B_2} \widehat{f}^{(b,r)} \tag{11.1}$$

puis

$$\widehat{f}^{\text{BLB}} = \frac{1}{B_1} \sum_{b=1}^{B_1} \widehat{f}^b. \tag{11.2}$$

L'approche complète est décrite dans l'algorithme 11.1.

Les deux étapes d'agrégation sont similaires dans un cadre bagging (et peuvent être simplifiées en une seule étape d'agrégation par la moyenne) mais le contexte plus général du bootstrap leur donne une utilité distincte. Dans le cadre standard, la première étape (équation (11.1)) correspond à l'estimation, par une approche Monte-Carlo, de la distribution empirique d'une statistique donnée à partir du sous-échantillon τ_b. Une estimation d'une mesure de qualité d'intérêt de cette statistique (par exemple, sa variance) en est déduite. La seconde étape, par contre, est très semblable à celle de l'équation (11.2), et procède au calcul de la moyenne des résultats de la première étape. Un résultat de consistance de la procédure est démontré, pour l'estimation bootstrap de la mesure de qualité d'intérêt, lorsque $B_1 \sim \frac{n}{m}$ et que $m \sim n^\gamma$ avec $\gamma \in [0{,}5; 1]$. Nous renvoyons le lecteur intéressé à Kleiner *et al.* [2014] pour plus de détails.

11.2.2 Approximation de Nyström

Les méthodes à noyau, dont font partie les SVM (voir chapitre **??**), ne passent pas bien à l'échelle lorsque le nombre d'observations n croît. En effet, typi-

[5] Pour $n = 1\,000\,000$ et $\gamma = 0{,}6$, les tailles respectives sont (environ) 3981 contre 630 000.

Algorithme 11.1 Algorithme « bag of little bootstrap » dans le cadre du bagging (cas de la régression)

Nécessite Une règle d'apprentissage d'un problème de régression qui associe, à tout ensemble d'apprentissage D_n, $\widehat{f}(D_n) : \mathbf{x} \in \mathcal{X} \mapsto \widehat{f}(D_n;\mathbf{x}) \in \mathbb{R}$
  Pour $b = 1 \to B_1$ **Faire**
    Tirer aléatoirement un ensemble de m indices τ_b dans $\{1, \ldots, n\}$
    Pour $r = 1 \to B_2$ **Faire**
      Simuler $(n_1^{(b,r)}, \ldots, n_m^{(b,r)}) \sim$ Multinomiale $\left(n, \frac{1}{m}\mathbf{1}_m\right)$
      Déterminer $\widehat{f}^{(b,r)} = \widehat{f}(D_n^{(b,r)})$ où $D_n^{(b,r)} = \left\{ \underbrace{(X_i,Y_i) \ldots (X_i,Y_i)}_{\text{répété } n_i^{(b,r)} \text{ fois}} \right\}_{i \in \tau_b}$
    Fin Pour
    $\widehat{f}^b \leftarrow \frac{1}{B_2} \sum_{r=1}^{B_2} \widehat{f}^{(b,r)}$
  Fin Pour
  $\widehat{f}^{\mathrm{BLB}} \leftarrow \frac{1}{B_1} \sum_{b=1}^{B_1} \widehat{f}^b$

quement, la complexité des approches utilisant des noyaux sur n observations est de l'ordre de n^2, ou même plus pour certaines méthodes (l'ACP à noyau [Schölkopf *et al.*, 1998], par exemple, a une complexité en n^3). Une approche classique pour réduire la complexité des méthodes à noyau est d'utiliser des approximations du noyau par une matrice de faible rang [Fine and Sheinberg, 2001; Bach, 2013]. La mise en œuvre des méthodes d'apprentissage connaît alors un gain de complexité qui est alors typiquement de l'ordre de $\mathcal{O}(nk^2)$ où k est le rang (faible, *ie* $k \ll n$) de la matrice utilisée dans l'approximation.

Dans ce chapitre, nous présentons une de ces approximations de faible rang, l'approximation de Nyström et montrons quelques-unes de ces applications, notamment à l'apprentissage de SVM à partir de données massives. Dans la suite, on supposera connu un noyau $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ qui est une fonction symétrique et semi-définie positive et on notera $\mathbf{K} = (K(X_i,X_j))_{i,j=1,\ldots,n}$ la matrice de Gram, qui est la matrice d'évaluation de ce noyau sur les valeurs observées de X. On note, de plus, $\mathcal{H}$ l'espace de Hilbert à noyau auto-reproduisant associé à K et $\Phi : \mathcal{X} \to \mathcal{H}$ la fonction de plongement associée à K, qui vérifie [Aronszajn, 1950] :

$$\forall\, \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \qquad K(\mathbf{x},\mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle_{\mathcal{H}}.$$

L'approximation de Nyström [Williams and Seeger, 2000; Drineas and Mahoney, 2005] est basée sur la sélection de m indices dans $\{1, \ldots, n\}$. Ces observations sont généralement choisies de manière aléatoire bien que des procédures de sélection plus efficaces aient également été développées [Kumar *et al.*, 2012]. Sans perte de généralité, on peut supposer que les indices sélectionnés sont les

indices $\{1, \ldots, m\}$ et on peut alors ré-écrire :

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}^{(m)} & \mathbf{K}^{(m,n-m)} \\ \mathbf{K}^{(n-m,m)} & \mathbf{K}^{(n-m,n-m)} \end{bmatrix} \qquad \text{et} \qquad \mathbf{K}^{(n,m)} = \begin{bmatrix} \mathbf{K}^{(m)} \\ \mathbf{K}^{(n-m,m)} \end{bmatrix},$$

avec $\mathbf{K}^{(n-m,m)} = (\mathbf{K}^{(m,n-m)})^\top$.

Approximation de la décomposition spectrale de K

Dans Williams and Seeger [2000], les auteurs montrent que l'approximation de Nyström peut être utilisée pour fournir une approximation de la décomposition spectrale de $\mathbf{K}$ (qui a une complexité en $\mathcal{O}(n^3)$) par la décomposition spectrale de $\mathbf{K}^{(m)}$ (qui a une complexité en $\mathcal{O}(m^3)$). Soient $(\mathbf{v}_j)_{j=1,\ldots,n} \subset \mathbb{R}^n$ (resp. $(\mathbf{v}_j^{(m)})_{j=1,\ldots,m} \subset \mathbb{R}^m$) les vecteurs propres de $\mathbf{K}$ (resp. $\mathbf{K}^{(m)}$) associés aux valeurs propres positives ou nulles, ordonnées par ordre décroissant, $(\mu_j)_{j=1,\ldots,n}$ (resp. $(\mu_j^{(m)})_{j=1,\ldots,n}$). Alors,

$$\forall\, j = 1, \ldots, m, \qquad \mu_j \simeq \frac{n}{m}\mu_j^{(m)} \qquad \text{et } \mathbf{v}_j \simeq \sqrt{\frac{m}{n}}\frac{1}{\mu_j^{(m)}}\mathbf{K}^{(n,m)}\mathbf{v}_j^{(m)}. \quad (11.3)$$

Dans la suite, on notera $\mu_j^{(n,m)}$ et $\mathbf{v}_j^{(n,m)}$, respectivement, les approximations de μ_j et $\mathbf{v}_j$ comme décrites dans l'équation (11.3) ci-dessus.

Le coût total de l'approximation de la décomposition spectrale de $\mathbf{K}$ par la décomposition spectrale de $\mathbf{K}^{(m)}$ est donc $\mathcal{O}(m^3) + \mathcal{O}(nm^2)$. Comme m est petit devant n, le coût est donc dominé par $\mathcal{O}(nm^2)$. Lorsque $\mathbf{K}^{(m)}$ et $\mathbf{K}$ sont de rangs m, l'approximation est exacte (*ie* le signe « $\simeq$ » de l'équation (11.3) devient une égalité).

Approximation de faible rang de K

Une approximation de faible rang $k \leqslant m$ de $\mathbf{K}$ est obtenue en calculant

$$\widetilde{\mathbf{K}} = \mathbf{K}^{(n,m)} \left(\mathbf{K}_k^{(m)}\right)^+ \mathbf{K}^{(m,n)} \quad (11.4)$$

avec $\mathbf{K}^{(m,n)} = \left(\mathbf{K}^{(n,m)}\right)^\top$ et $\mathbf{K}_k^{(m)}$ est la meilleure approximation de rang k de $\mathbf{K}^{(m)}$ pour la norme de Frobenius : $\mathbf{K}_k^{(m)} = \text{argmin}_{\text{Rang}(\mathbf{V})=k} \|\mathbf{K}^{(m)} - \mathbf{V}\|_F^2$. Enfin, $\left(\mathbf{K}_k^{(m)}\right)^+$ est la pseudo-inverse de $\mathbf{K}_k^{(m)}$. Dès que le rang de $\mathbf{K}^{(m)}$ est supérieur ou égal à k (ce que nous supposerons dans la suite), celle-ci s'obtient facilement à partir de la décomposition spectrale de $\mathbf{K}^{(m)}$:

$$\left(\mathbf{K}_k^{(m)}\right)^+ = \sum_{j=1}^{k} \left(\mu_j^{(m)}\right)^{-1} \mathbf{v}_j^{(m)} \left(\mathbf{v}_j^{(m)}\right)^\top. \quad (11.5)$$

En introduisant, dans les équations (11.4) et (11.5), les valeurs de $\mu_j^{(n,m)}$ et $\mathbf{v}_j^{(n,m)}$ définies dans l'équation (11.3), on obtient facilement

$$\widetilde{\mathbf{K}} = \sum_{j=1}^{k} \mu_j^{(n,m)} \mathbf{v}_j^{(n,m)} \left(\mathbf{v}_j^{(n,m)}\right)^\top .$$

Le rang de $\widetilde{\mathbf{K}}$ est donc égal à k. De plus, si le rang de $\mathbf{K}$ est aussi égal à k, alors $\mathbf{K} = \widetilde{\mathbf{K}}$.

L'analyse théorique de la qualité de l'approximation est faite par Drineas and Mahoney [2005]. On donne ci-dessous une version simplifiée des résultats de Drineas and Mahoney [2005] qui est utilisée dans Cortes *et al.* [2010] pour proposer un résultat d'approximation de la qualité de la prévision dans le cadre de la régression ridge à noyau :

Théorème 11.1 *Si la sélection des m observations utilisées pour l'approximation de Nyström de $\mathbf{K}$ est faite selon un tirage uniforme sur $\{1, \ldots, n\}$ alors avec probabilité au moins égale à $1 - \delta$,*

$$\|\widetilde{\mathbf{K}} - \mathbf{K}\|_2 \leqslant \|\mathbf{K}_k - \mathbf{K}\|_2 + \frac{n}{\sqrt{m}} \max_{i=1,\ldots,n} \mathbf{K}_{ii} \left(2 + \log\left(\frac{1}{\delta}\right)\right)$$

où $\mathbf{K}_k$ est la meilleure approximation de $\mathbf{K}$ de rang k et $\mathbf{K}_{ii}$ sont les termes diagonaux de $\mathbf{K}$.

Application en apprentissage

Nous présentons une application de l'approximation de Nyström dans le cadre de la régression ridge à noyau (voir Saunders *et al.* [1998] ou le chapitre **??**). Dans ce type de problèmes, il s'agit de minimiser l'erreur quadratique moyenne pénalisée suivante :

$$\operatorname*{argmin}_{\mathbf{w}\in\mathcal{H}} \sum_{i=1}^{n} \left(Y_i - \langle \mathbf{w}, \Phi(X_i)\rangle_{\mathcal{H}}\right)^2 + \frac{\lambda}{2}\|\mathbf{w}\|_{\mathcal{H}}^2.$$

La solution de ce problème conduit à une fonction de régression

$$\widehat{f} : \ \mathbf{x} \in \mathcal{X} \mapsto \langle \widehat{\mathbf{w}}, \Phi(\mathbf{x})\rangle_{\mathcal{H}}$$

qui, d'après le théorème de représentation (« Representer Theorem » ; voir [Kimeldorf and Wahba, 1971; Schölkopf *et al.*, 2001]), prend la forme

$$\widehat{f} : \ \mathbf{x} \in \mathcal{X} \mapsto \sum_{i=1}^{n} \alpha_i K(X_i, \mathbf{x})$$

où

$$\begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = (\mathbf{K} + \lambda \mathbb{I}_n)^{-1} \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix}. \tag{11.6}$$

Williams and Seeger [2000] montrent qu'en remplaçant $\mathbf{K}$ par son approximation $\widetilde{\mathbf{K}}$ dans l'équation (11.6), la complexité du calcul matriciel est fortement diminuée grâce à la formule de Woodbury [Press *et al.*, 1992]. En effet, si $\tilde{\boldsymbol{\alpha}}$ désigne la solution de l'équation (11.6) lorsque l'on remplace $\mathbf{K}$ par son approximation $\widetilde{\mathbf{K}}$, $\tilde{\boldsymbol{\alpha}}$ se calcule par :

$$\tilde{\boldsymbol{\alpha}} = \frac{1}{\lambda}\left(\mathbf{Y} - \mathbf{V}_k\left(\lambda \mathbb{I}_n + \boldsymbol{\Lambda}_k \mathbf{V}_k^\top \mathbf{V}_k\right)^{-1} \boldsymbol{\Lambda}_k \mathbf{V}_k^\top \mathbf{Y}\right), \qquad \text{où } \mathbf{Y} = \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix},$$

$\mathbf{V}_k$ est la matrice de dimensions $n \times k$ dont les colonnes sont les vecteurs propres $(\mathbf{v}_j^{(n,m)})_{j=1,\ldots,k}$ de $\widetilde{\mathbf{K}}$ et $\boldsymbol{\Lambda}_k$ est la matrice diagonale de taille k dont les éléments sont les valeurs propres de $\widetilde{\mathbf{K}}$: $\boldsymbol{\Lambda}_k = \text{Diag}\left(\mu_1^{(n,m)}, \ldots, \mu_k^{(n,m)}\right)$.

Des résultats sur l'impact de cette approximation sur la prédiction et l'erreur de généralisation sont fournis dans [Cortes *et al.*, 2010; Bach, 2013]. De manière similaire, l'approximation de Nyström est utilisée pour permettre le passage à l'échelle des SVM (voir chapitre **??**) ou de nombreuses autres méthodes d'apprentissage et de fouille de données basées sur un noyau.

11.3 Approches « Diviser pour mieux régner »

11.3.1 Présentation du paradigme « Map Reduce »

Map Reduce (MR) est un paradigme de programmation simple qui a été spécifiquement conçu pour s'adapter à des environnements de calcul parallèles ou répartis. Cette méthode a été développée par Google en 2004 [Dean and Ghemawat, 2004].

De manière simplifiée, un algorithme de calcul effectué avec MR se décompose en deux tâches, illustrées dans la figure **??** du chapitre **??** (voir aussi Chamandy *et al.* [2012] pour quelques considérations sur les modalités pratiques d'utilisation chez Google).

1. une étape « Map » qui est appliquée à un sous-ensemble d'observations $(X_i,Y_i)_{i\in\tau_r}$ qui produit une liste de *clés* (que l'on peut voir comme une indexation des données) et de *valeurs* (souvent le résultat d'un calcul donné). Q différentes tâches de type « Map » sont effectuées en parallèle et de manière indépendante sur des sous-échantillons de données de telle sorte que les $(\tau_r)_{r=1,\ldots,Q}$ forment une partition[6] de $\{1, \ldots, n\}$;

[6] Notons que pour des raisons techniques, liées essentiellement à une volonté de tolérance aux pannes du paradigme, il est possible en pratique que plusieurs tâches « Map » soient appliquées aux mêmes données. Cependant, le système s'arrange, dans ce cas, pour que, dans la partie « Reduce », tout se passe comme si on avait bien une partition des données et aucun calcul redondant.

2. une étape « Reduce » qui est appliquée aux sorties de l'étape « Map » qui correspondent à une clé donnée. Là aussi, différentes tâches de type « Reduce » sont exécutées en parallèle, correspondant aux différentes valeurs de clés fournies par les tâches « Map ».

Exemple 11.1 Un exemple typique d'utilisation de MR est illustré dans le cas où $\mathcal{X} = \mathbb{R}$ et $\mathcal{Y} = \{1, \ldots, L-1\}$. Si on souhaite calculer une statistique sur les valeurs de X conditionnelle aux valeurs de Y (par exemple, les moyennes conditionnelles), on peut décomposer le calcul de la manière suivante :

1. $\forall\, i = 1, \ldots, n$, l'étape « Map » produira l'ensemble de (clé,valeur) : (Y_i, X_i) ;
2. l'étape « Reduce » calculera alors la statistique sur les valeurs de X_i correspondant à une clé unique dans $\{1, \ldots, L-1\}$.

La formulation de la décomposition d'un problème donné sous une forme MR n'est pas toujours aussi directe. Dans la suite, nous discutons deux exemples d'adaptation de méthodes statistiques au paradigme MR : celui du modèle linéaire et celui des forêts aléatoires (voir le chapitre **??**).

11.3.2 Adaptation du modèle linéaire au paradigme MR

Dans [Chu *et al.*, 2010], les auteurs montrent que plusieurs méthodes d'apprentissage ou de fouille de données sont facilement adaptables au paradigme MR. En particulier, dès qu'une méthode statistique est basée sur le calcul d'une ou de plusieurs sommes du type $\sum_{i=1}^{n}[...]$, les sommes partielles $S_r = \sum_{i\in\tau_r}[...]$ peuvent être calculées en parallèle sur différentes tâches de type « Map » avec en sortie la clé constante « 1 ». Une unique tâche « Reduce » effectue la somme des $(S_r)_{r=1,\ldots,Q}$ et effectue les calculs additionnels requis.

Si l'on prend le cas simple du modèle linéaire, $Y = \boldsymbol{\beta}^\top X + \epsilon$, la solution de l'estimation de $\boldsymbol{\beta}$ par minimisation du risque quadratique empirique s'écrit sous la forme

$$\boldsymbol{\Sigma}_n \hat{\boldsymbol{\beta}} = \boldsymbol{\Gamma}_n \tag{11.7}$$

avec $\boldsymbol{\Sigma}_n = \frac{1}{n}\sum_{i=1}^{n} X_i X_i^\top$ et $\boldsymbol{\Gamma}_n = \frac{1}{n}\sum_{i=1}^{n} X_i Y_i$.

En suivant l'idée de Chu *et al.* [2010], on peut retrouver de manière exacte l'estimateur de l'équation (11.7) :

1. chaque tâche « Map » calcule les sommes partielles

$$\forall\, r = 1, \ldots, Q, \qquad n_r = \operatorname{Card}\tau_r,\ \boldsymbol{\Sigma}_n^r = \sum_{i\in\tau_r} X_i X_i^\top \text{ et } \boldsymbol{\Gamma}_n^r = \sum_{i\in\tau_r} X_i Y_i;$$

2. une tâche « Reduce » calcule, successivement,

$$n = \sum_{r=1}^{Q} n_r,\ \boldsymbol{\Sigma}_n = \frac{\sum_{r=1}^{Q}\boldsymbol{\Sigma}_n^r}{n},\ \boldsymbol{\Gamma}_n = \frac{\sum_{r=1}^{Q}\boldsymbol{\Gamma}_n^r}{n} \text{ et finalement } \hat{\boldsymbol{\beta}} = \boldsymbol{\Sigma}_n^{-1}\boldsymbol{\Gamma}_n.$$

Notons que cette approche est intéressante si n est grand mais que la dimension de X reste faible. En effet l'inversion de la matrice Σ_n est réalisée dans une tâche « Reduce » qui est exécutée de façon séquentielle. Le gain en temps de calcul se fait sur le calcul simultané des sommes partielles, ce qui est avant tout intéressant si n est grand.

Si l'on considère maintenant le cadre de la régression linéaire pénalisée dans lequel $\boldsymbol{\beta}$ est estimé par minimisation d'un risque empirique pénalisé :

$$\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \boldsymbol{\beta}^\top X_i\right)^2 + \lambda\,\mathrm{pen}(\boldsymbol{\beta})$$

où, $\lambda \in \mathbb{R}^+$ et, classiquement

- $\mathrm{pen}(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|_2^2 = \sum_{j=1}^{p}\beta_j^2$: dans ce cas, on parle de régression ridge (ou régression d'arête) ou de régularisation de Tikhonov [Tikhonov, 1963] ; en particulier, cette approche est un cas particulier, avec un noyau linéaire, de la régression ridge à noyau décrite dans la section 11.2.2) ;

- ou bien $\mathrm{pen}(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|_1 = \sum_{j=1}^{p}|\beta_j|$: dans ce cas, on parle de régression Lasso [Tibshirani, 1996]. La contrainte sur la norme L_1 du paramètre $\boldsymbol{\beta}$ va imposer une solution parcimonieuse (« sparse ») où certains coefficients β_j seront exactement égaux à zéro.

Dans ce cadre-ci, Chen and Xie [2014] montrent que l'approche proposée par Chu *et al.* [2010] n'est pas applicable directement. Ils proposent alors une méthode de pondération des différents estimateurs pénalisés obtenus à partir des différents échantillons $(\tau_r)_{r=1,\ldots,Q}$ qui soit asymptotiquement équivalente (quand $n \to +\infty$ et avec $Q = n^\delta$ pour $0 \leqslant \delta \leqslant 1/2$) à l'estimateur qui aurait été obtenu avec l'intégralité des données.

11.3.3 Adaptation des forêts aléatoires au paradigme MR

Les forêts aléatoires ne sont pas basées sur le calcul de sommes $\sum_{i=1}^{n}[...]$. Aussi, comme décrit dans del Rio *et al.* [2014], la méthode classique pour utiliser l'algorithme des forêts aléatoires dans le paradigme MR consiste à construire en parallèle des forêts indépendantes basées sur un sous-échantillon des données. De manière plus précise,

1. l'étape « Map » va construire une forêt de T arbres basés sur les données $(X_i,Y_i)_{i\in\tau_r}$. Dans la suite, on notera $\widehat{f}^{r,t}$ pour $t = 1, \ldots, T$ les T arbres de la forêt. La sortie de chaque tâche « Map » sont les couples de (clé,valeur) $(1,\widehat{f}^{r,t})$ pour t variant de 1 à T ;

2. une unique étape « Reduce » collecte tous les arbres et les agrège en une

forêt unique, ce qui donne la fonction de prédiction

$$\forall \mathbf{x} \in \mathcal{X}, \qquad \widehat{f}(\mathbf{x}) = \frac{1}{TQ} \sum_{r=1}^{Q} \sum_{t=1}^{T} \widehat{f}^{r,t}(\mathbf{x})$$

dans le cadre de la régression ($\mathcal{Y} = \mathbb{R}$) et

$$\forall \mathbf{x} \in \mathcal{X}, \qquad \widehat{f}(\mathbf{x}) = \underset{y \in \{1,\ldots,L-1\}}{\operatorname{argmax}} \operatorname{Card} \left\{ (r,t) : \ \widehat{f}^{r,t}(\mathbf{x}) = y \right\}$$

dans le cadre de la classification ($\mathcal{Y} = \{1, \ldots, L-1\}$).

Une approximation de l'erreur « out-of-bag » (OOB, voir chapitre **??**) de la forêt est alors obtenue par :

$$\widehat{\mathcal{R}}^{\mathrm{MR}}_{\mathrm{OOB}}(\widehat{f}) = \frac{1}{n} \sum_{r=1}^{Q} n_r \widehat{\mathcal{R}}^{\tau_r}_{\mathrm{OOB}}(\widehat{f}^r) \tag{11.8}$$

où $n_r = \operatorname{Card} \tau_r$, $\widehat{f}^r$ est la forêt aléatoire issue de la r-ième tâche « Map » (qui est basée sur les arbres $(\widehat{f}^{r,t})_{t=1,\ldots,T}$) et $\widehat{\mathcal{R}^{\tau_r}_{\mathrm{OOB}}}$ désigne l'erreur OOB restreinte aux observations de τ_r. De manière similaire, l'erreur OOB d'un arbre $\widehat{f}^{r,t}$ peut être estimée en restreignant sa définition aux observations de τ_r : cela permet également d'obtenir une estimation de l'importance des variables dans le cadre de MR (voir Genuer *et al.* [2015] pour de plus amples détails).

Comme souligné dans Genuer *et al.* [2015], la définition de la forêt ainsi que l'estimation de son erreur OOB et l'importance des variables n'est pas strictement équivalente au cadre standard dans lequel toutes les données sont utilisées pour définir les échantillons bootstrap et faire l'apprentissage des arbres. En particulier, des biais d'apprentissage peuvent apparaître dans le cas où les observations ne sont pas réparties de manière aléatoires entre les différentes tâches « Map », qui sont des conséquences directes de la propriété de traitement localisé des données (voir section 11.3.4). D'autre part, l'erreur OOB telle que définie dans l'équation (11.8) est souvent une estimation très grossière de la véritable erreur OOB de $\widehat{f}$ calculée avec l'ensemble des données et elle peut ne pas bien refléter l'erreur de généralisation de $\widehat{f}$ (voir Genuer *et al.* [2015] pour une discussion plus approfondie de ce point).

Une approche alternative pour l'utilisation des forêts aléatoires dans un cadre MR, plus proche de l'algorithme initial, est l'utilisation du bootstrap Poisson [Oza and Russel, 2001; Lee and Clyde, 2004; Hanley and MacGibbon, 2006] dont les propriétés théoriques et pratiques ont été étudiées dans Chamandy *et al.* [2012] qui montrent que la technique est proche du bootstrap multinomial standard pour le cadre de l'échantillonnage bootstrap en ligne. Comme pour la méthode BLB (décrite en section 11.2.1), cette approche est facilement utilisable dans un cadre de bagging .

Le principe de la méthode est de simuler l'effectif d'apparition d'une observation donnée dans un échantillon bootstrap donné par simulation à partir de la loi de Poisson de paramètre 1. Cette idée est basée sur l'approximation asymptotique

$$\text{Binomiale}\left(n, \frac{1}{n}\right) \sim \text{Poisson}(1).$$

Une implémentation des forêts aléatoires, basée sur ce principe, est décrite ci-dessous :

1. l'étape « Map » reçoit des indices τ_r (pour $r = 1, \ldots, Q$). $\forall\, i \in \tau_r$, un vecteur B variables aléatoire i.i.d. de loi Poisson(1) sont simulées n_i^b ($b = 1, \ldots, B$). Les tâches « Map » retournent les paires (clé,valeur) $(b, (i, n_i^b))$ pour l'ensemble des couples (i,b) tels que $n_i^b \neq 0$. Les indices i tels que $n_i^b \neq 0$ constituent l'échantillon bootstrap numéro b et n_i^b est l'effectif de l'observation d'indice i dans cet échantillon ;

2. l'étape « Reduce » traite séparément chaque échantillon bootstrap indicé par $b \in \{1, \ldots, B\}$: l'arbre de la forêt aléatoire qui est associé à l'indice b est défini à partir des observations d'indices i tels que $n_i^b \neq 0$, répétées respectivement n_i^b fois, c'est-à-dire, sur l'ensemble d'apprentissage $\left\{ \underbrace{(X_i,Y_i) \ldots (X_i,Y_i)}_{n_i^b \text{ fois}} \right\}_{i:\ n_i^b \neq 0}$. La sortie de l'étape « Reduce » est donc une collection d'arbres correspondant à la forêt aléatoire finale.

Contrairement à l'implémentation MR décrite dans del Rio *et al.* [2014], les forêts aléatoires utilisant le bootstrap Poisson sont basées sur des échantillons bootstrap construits à partir de l'intégralité des données et ne connaissent donc pas les mêmes problèmes de biais d'échantillonnage. Par contre, dans le cas où n est très grand, les étapes « Reduce » doivent définir des arbres à partir d'approximativement $0{,}63 \times n$ observations distinctes (voir la remarque de l'introduction de la section 11.2.1), ce qui peut être prohibitif. Dans ce cas, la solution décrite dans la section 11.2.1, basée sur des échantillons bootstrap de tailles plus restreintes, est probablement plus adaptée.

11.3.4 Limitations de MR

Le paradigme « Map Reduce » est en théorie utilisable pour réaliser des calculs sur un environnement réparti quelconque. Toutefois, en pratique, il est essentiellement mis en œuvre dans l'environnement logiciel « Hadoop[7] » et est surtout conçu pour réaliser des opérations sur un grand nombre de machines qui constituent un système de stockage de données réparti. Les tâches « Map »

[7] `https://hadoop.apache.org`

sont réalisées localement sur les machines qui stockent les données traitées et cette particularité assure l'efficacité de l'approche : le cas d'application typique original était lié à Google (nous renvoyons le lecteur intéressé à la lecture de Chamandy *et al.* [2012] qui décrivent ce que sont les données « à échelle Google » et la manière dont celles-ci sont traitées). Si « Map Reduce » est généralement adapté à ce type de configurations, il peut être assez lent pour traiter d'autres tâches de calcul.

En particulier, « Map Reduce » n'est pas adapté pour la mise en œuvre de processus itératifs nécessitant plusieurs passages sur les données. En effet, la façon naturelle de traduire ce type de processus dans le paradigme consiste à englober une tâche « Map Reduce » dans une boucle. En pratique cela induit un chargement des données depuis les disques durs et une écriture des résultats intermédiaires à chaque itération, ce qui ralentit considérablement l'exécution (cf le chapitre **??** pour un exemple détaillé autour de l'algorithme de *Forgy*). Même en contournant la phase de chargement répétée par l'intégration d'une forme de boucle élémentaire dans le mécanisme, comme dans Durut and Rossi [2011], par exemple, les communications entre les tâches « Map » et les tâches « Reduce », puis la communication des résultats de ces dernières à de nouvelles tâches « Map », induisent un ralentissement important.

Plusieurs autres paradigmes ont été conçus afin de répondre à ce problème, comme par exemple Isard *et al.* [2007]; Low *et al.* [2010]. En raison notamment de leur complexité, ces paradigmes n'ont pas eu le succès de « Map Reduce ». Ils n'ont pas non plus bénéficié d'un large écosystème informatique comme celui fourni par Hadoop. Au moment de la rédaction de cet ouvrage, l'équilibre entre efficacité et simplicité semble être fourni par la technologie *Spark* Zaharia *et al.* [2012], intégrée dans Hadoop (cf le chapitre **??**). Cette approche s'appuie sur la notion de *Resilient Distributed Dataset* (RDD) qui peut être vue comme une représentation adaptative d'un ensemble de données sur un système informatique réparti. L'adaptativité de *Spark* lui permet de conserver les données dans l'ensemble des mémoires vives des ordinateurs qui forment le système réparti, quand cela est possible. Ceci règle le problème des chargements répétés qui ralentissent « Map Reduce » dans les calculs itératifs.

En outre, plutôt que de demander à l'analyste de décrire un ensemble de traitements sous forme d'un graphe de tâches comme dans Isard *et al.* [2007]; Low *et al.* [2010], *Spark* propose un large ensemble d'opérations de haut niveau sur les RDD. C'est le moteur d'exécution qui se charge de traduire les traitements, exprimés ainsi de façon relativement classique, en un graphe de tâches. Comme pour une requête dans un gestionnaire de bases de données, l'exécution du graphe de tâches est optimisée en tenant compte notamment des communications entre machines induites par les traitements. Les opérations disponibles ont deux formes : les *transformations* (qui rappellent les tâches « Map ») et les *actions* (qui rappellent les tâches « Reduce »). Une transformation *Spark* typique consiste par exemple à calculer une nouvelle variable à partir des variables des données considérées. Concrètement, le calcul de $X_i X_i^\top$ utilisé dans

la version « Map Reduce » du modèle linéaire, est une transformation. Une action *Spark* typique consiste à agréger des valeurs sur plusieurs observations. Par exemple le calcul de $\sum_i X_i X_i^\top$ à partir des $X_i X_i^\top$ est une transformation.

La relative simplicité de la programmation sous le paradigme *Spark*, permise notamment par la richesse des transformations et actions proposées, a permis le développement de bibliothèques de haut niveau, par exemple autour des algorithmes d'apprentissage. Ces bibliothèques sont décrites et mise en œuvre dans le chapitre **??**.

11.4 Mise à jour en ligne

11.4.1 Introduction et notations

Dans ce chapitre sont présentés des exemples de méthodes d'apprentissage en ligne. Contrairement à ce qui a été présenté dans les chapitres précédents, il ne s'agit pas ici simplement de considérer un apprentissage dans lequel un ensemble d'observations, connues à l'avance, est utilisé pour l'apprentissage d'une fonction de régression ou de classification à l'aide d'un algorithme stochastique. Au contraire, nous nous intéressons au cas où les données elles-mêmes arrivent de manière séquentielle (c'est le « V » de **V**itesse). De manière plus précise, on supposera qu'une fonction de régression ou un classifieur, $\widehat{f}_n$, a été construit à partir d'un ensemble d'apprentissage $(X_i,Y_i)_{i=1,\dots,n}$ et que de nouvelles observations $(X_i,Y_i)_{i=n+1,\dots,n+m}$ se présentent et sont également des tirages i.i.d. du couple aléatoire (X,Y)[8]. Le problème consiste à ne pas apprendre une nouvelle fonction de régression ou un nouveau classifieur à partir de l'ensemble des données $(X_i,Y_i)_{i=1,\dots,n+m}$ mais à profiter de la connaissance déjà acquise pour mettre à jour $\widehat{f}_n$ à partir des nouvelles données. Ce problème porte le nom d'*apprentissage en ligne* (« online learning ») ou bien de *apprentissage pour données de flux* (« stream data learning »). Le principe de ces méthodes est que le coût de mise à jour à partir d'un ensemble de données de faible taille (m petit) doit être peu élevé. Dans ce cas, l'apprentissage en ligne peut même être utile pour faire de l'apprentissage à partir d'un gros volume de données : l'ensemble initial est découpé en ensembles de petites tailles qui sont utilisés pour mettre à jour un prédicteur de manière séquentielle.

Prenons un premier exemple simple d'un problème pour lequel la mise en ligne peut être formulée de manière naturelle : le prédicteur des k plus proches voisins (voir section **??**) . Dans le cadre de la régression k-NN ($\mathcal{Y} = \mathbb{R}$), $\widehat{f}_n$ est donnée par

$$\forall\, \mathbf{x} \in \mathcal{X}, \qquad \widehat{f}_n(\mathbf{x}) = \sum_{i=1}^{n} W_i^{k\text{-ppv}}(X_{1\dots n}; \mathbf{x}) Y_i$$

[8] Dans toute cette section, on fera l'hypothèse implicite que le modèle sous-jacent ne change pas. On ne traitera donc pas le cas du « concept drift » (*dérive de concept*), dans lequel la relation entre Y et X n'est pas stationnaire au cours du temps.

où $W_i^{k\text{-ppv}}(X_{1\dots n};\mathbf{x}) = \frac{1}{k}$ si X_i est dans les k-plus proches voisins de $\mathbf{x}$ parmi $(X_i)_{i=1,\dots,n}$ pour une certaine distance d dans $\mathcal{X}$ et est égal à 0 sinon. La mise à jour en ligne de $\widehat{f}(\mathbf{x})$ (pour $\mathbf{x}$ fixé) se fait de manière relativement simple et requiert de conserver en mémoire (et mettre à jour) les quantités suivantes :

- le vecteur (de taille k) des indices des plus proches voisins de $\mathbf{x}$, $\mathcal{N}^n(\mathbf{x}) \in \{1, \dots, n\}^k$. Sans perte de généralité, on supposera que les coefficients de $\mathcal{N}^n(\mathbf{x})$, notés $\mathcal{N}_l^n(\mathbf{x})$, sont ordonnés par valeur croissante de la distance à $\mathbf{x}$;
- le vecteur des distances entre les observations de l'ensemble $(X_i)_{i\in\mathcal{N}_n(\mathbf{x})}$ et l'observation $\mathbf{x}$ à prédire : $\mathbf{d}^n(\mathbf{x}) = (d(X_i,\mathbf{x}))_{i\in\mathcal{N}^n(\mathbf{x})}$. On notera $d_l^n(\mathbf{x})$ le l-ème coefficient de ce vecteur ;
- $\widehat{f}_n(\mathbf{x})$.

Lorsque m nouvelles observations $(X_i,Y_i)_{i=n+1,\dots,n+m}$ se présentent, la mise à jour se fait comme décrit dans l'algorithme 11.2. Dans cet exemple, le coût

Algorithme 11.2 Mise à jour en ligne de la fonction de régression k-NN au point $\mathbf{x}$

Initialiser $\mathcal{N}^+_{n,n+m}(\mathbf{x}) = \mathcal{N}^-_{n,n+m}(\mathbf{x}) \leftarrow \emptyset$
Pour $i = n+1 \to n+m$ **Faire**
  Calculer $d(X_i,\mathbf{x})$
  Pour $l = 1 \to k$ **Faire**
    Si $d(X_i,\mathbf{x}) < d_l^{n+i-1}(\mathbf{x})$ **Alors**
      $\mathbf{d}^{n+i}(\mathbf{x}) \leftarrow \big[d_1^{n+i-1}(\mathbf{x}), \dots, d_{l-1}^{n+i-1}(\mathbf{x}), d(X_i,\mathbf{x}), d_{l+1}^{n+i-1}(\mathbf{x}), \dots, d_{k-1}^{n+i-1}(\mathbf{x})\big]$
      $\mathcal{N}^+_{n,n+m}(\mathbf{x}) \leftarrow \mathcal{N}^+_{n,n+m}(\mathbf{x}) \cup \{i\}$
      $\mathcal{N}^-_{n,n+m}(\mathbf{x}) \leftarrow \mathcal{N}^-_{n,n+m}(\mathbf{x}) \cup \{\mathcal{N}_k^{n+i-1}(\mathbf{x})\}$
      $\mathcal{N}^{n+i}(\mathbf{x}) \leftarrow \big[\mathcal{N}_1^{n+i-1}(\mathbf{x}), \dots, \mathcal{N}_{l-1}^{n+i-1}(\mathbf{x}), i, \mathcal{N}_{l+1}^{n+i-1}(\mathbf{x}), \dots, \mathcal{N}_{k-1}^{n+i-1}(\mathbf{x})\big]$
      Arrêt de la procédure pour l'observation i
    Fin Si
  Fin Pour
Fin Pour
Mettre à jour la fonction de prédiction :

$$\widehat{f}_{n+m}(\mathbf{x}) \leftarrow \widehat{f}_n(\mathbf{x}) + \frac{1}{k}\left[\sum_{i\in\mathcal{N}^+_{n,n+m}(\mathbf{x})} Y_i \; - \sum_{i\in\mathcal{N}^-_{n,n+m}(\mathbf{x})} Y_i\right]$$

de la mise à jour est de m calculs de distances, au plus $m \times k$ comparaisons et

k opérations pour mettre à jour la fonction de prédiction. Le coût de stockage correspond à 2 vecteurs de taille k et un nombre réel. Ces coûts restent très inférieurs aux coûts de calcul que nécessiterait la recherche directe des k plus proches voisins dans l'ensemble $(X_i)_{i=1,\dots,n+m}$.

Dans la suite, nous décrivons deux approches pour l'apprentissage de flux de données : la première présente un exemple de mise à jour en ligne de méthodes à noyau et la seconde montre comment une combinaison d'estimateurs de type « bagging » peut être réalisée pour combiner des prédicteurs qui admettent eux-mêmes une mise à jour en ligne.

11.4.2 Exemple de la régression ridge à noyau en ligne

Dans ce chapitre nous prenons un exemple assez typique de mise à jour en ligne en discutant des approches qui peuvent être pratiquées pour la régression ridge à noyau (voir section 11.2.2 ou le chapitre **??** ou bien encore Saunders *et al.* [1998] pour une description de la méthode). On rappelle que dans ce cadre, la solution du problème de régression obtenue à partir des observations $(X_i,Y_i)_{i=1,\dots,n}$ est donnée par

$$\widehat{f}_n : \ \mathbf{x} \in \mathcal{X} \mapsto \sum_{i=1}^{n} \alpha_i^{(n)} K(X_i,\mathbf{x}) \tag{11.9}$$

où

$$\begin{pmatrix} \alpha_1^{(n)} \\ \vdots \\ \alpha_n^{(n)} \end{pmatrix} = \left(\mathbf{K}^{(n)} + \lambda \mathbb{I}_n\right)^{-1} \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix}. \tag{11.10}$$

et $\mathbf{K}^{(n)}$ est la matrice de taille $n \times n$ du noyau évalué sur les n observations de X : $\mathbf{K}^{(n)} = (K(X_i,X_j))_{i,j=1,\dots,n}$.

Différentes approches peuvent être pratiquées pour la mise à jour en ligne de cette solution. Nous nous restreindrons ici à une mise à jour effectuée à partir d'une seule nouvelle observation (X_{n+1},Y_{n+1}). Une méthode typique pour effectuer celle-ci consiste à utiliser le prédicteur courant, $\widehat{f}_n$, pour estimer la valeur de Y correspondant à X_{n+1} : $\widehat{Y}_{n+1} = \widehat{f}_n(X_{n+1})$ et à mettre à jour $\widehat{f}_n$ à partir de l'erreur $e_{n+1} = Y_{n+1} - \widehat{Y}_{n+1}$.

Si on note $\mathbf{Q}^{(n)} = \left(\mathbf{K}^{(n)} + \lambda \mathbb{I}_n\right)^{-1}$, Engel *et al.* [2003]; van Vaerenbergh *et al.* [2012] montrent que l'équation (11.10) peut être mise à jour en remarquant que :

$$\mathbf{K}^{(n+1)} = \begin{pmatrix} \mathbf{K}^{(n)} & \mathbf{k}_{n+1} \\ \mathbf{k}_{n+1}^\top & K(X_{n+1},X_{n+1}) \end{pmatrix}$$

avec $\mathbf{k}_{n+1}$ le vecteur $(K(X_{n+1},X_i))_{i=1,\dots,n}$. On peut en peut déduire que

$$\mathbf{Q}^{(n+1)} = \frac{1}{\gamma_{n+1}} \begin{pmatrix} \gamma_{n+1}\mathbf{Q}^{(n)} + \mathbf{a}_{n+1}\mathbf{a}_{n+1}^\top & -\mathbf{a}_{n+1} \\ -\mathbf{a}_{n+1}^\top & 1 \end{pmatrix}$$

avec $\mathbf{a}_{n+1} = \mathbf{Q}^{(n)}\mathbf{k}_{n+1}$, un vecteur de $\mathbb{R}^n$, et $\gamma_{n+1} = K(X_{n+1},X_{n+1}) + \lambda - \mathbf{k}_{n+1}^\top \mathbf{a}_{n+1}$. En remarquant que

$$\mathbf{a}_{n+1}^\top \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix} = \mathbf{k}_{n+1}^\top \mathbf{Q}^{(n)} \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix} = \mathbf{k}_{n+1}^\top \boldsymbol{\alpha}^{(n)} = \widehat{Y}_{n+1},$$

ceci conduit finalement à la solution

$$\widehat{f}_{n+1} : \ \mathbf{x} \in \mathcal{X} \mapsto \sum_{i=1}^{n+1} \alpha_i^{(n+1)} K(X_i,\mathbf{x})$$

où

$$\begin{pmatrix} \alpha_1^{(n+1)} \\ \vdots \\ \alpha_{n+1}^{(n+1)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\alpha}^{(n)} - \frac{1}{\gamma_{n+1}} \mathbf{a}_{n+1} e_{n+1} \\ \frac{1}{\gamma_{n+1}} e_{n+1} \end{pmatrix}. \tag{11.11}$$

Le coût de calcul de la solution en ligne précédente requiert $\mathcal{O}(n^2)$ opérations à comparer au coût de la solution directe qui demande de l'ordre de $\mathcal{O}(n^3)$ opérations. Cette approche peut être rendue encore plus efficace en utilisant un *dictionnaire*, qui correspond à un ensemble d'indices $i \in \mathrm{Dico}_n \subset \{1, \ldots, n\}$, de taille $m < n$, pour lesquels les coefficients $\alpha_i^{(n)}$ de l'équation (11.9) sont non nuls. La fonction de prédiction est alors donnée par

$$\widehat{f}_n(\mathbf{x}) = \sum_{i \in \mathrm{Dico}_n} \alpha_i^{(n)} K(X_i,\mathbf{x})$$

et une mise à jour en ligne efficace consiste, par exemple, à faire grossir le dictionnaire comme dans Engel *et al.* [2003] :

1. définir un critère pour tester la pertinence d'ajouter la nouvelle observation d'indice $n+1$ au dictionnaire ;
2. dans le cas où le critère est accepté, utiliser la mise à jour en ligne de l'équation (11.11) pour mettre à jour le prédicteur.

Une alternative à faire grossir le dictionnaire consiste à mettre à jour la fonction de régression puis, *a posteriori*, à mettre à jour le dictionnaire comme dans van Vaerenbergh *et al.* [2012] :

1. utiliser la mise à jour en ligne de l'équation (11.11) pour mettre à jour le prédicteur en utilisant les données du dictionnaire $\mathcal{D}_n$ et la nouvelle donnée d'indice $n+1$;
2. dans le cas où le dictionnaire dépasse une certaine taille m, fixée à l'avance, définir un critère pour retirer du dictionnaire l'observation la moins pertinente.

Le lecteur intéressé trouvera plus de détails sur les approches permettant la mise à jour en ligne du prédicteur de la régression ridge à noyau dans van Vaerenbergh and Santamaría [2014], avec, en particulier, d'autres approches basées sur des algorithmes de descente de gradient stochastiques (voir aussi le chapitre **??** pour une description sommaire du principe de la descente de gradient stochastique). Enfin, une extension des SVM (voir chapitre **??**) pour une mise à jour en ligne est proposée dans Laskov *et al.* [2006]. Celle-ci utilise des principes similaires à ceux décrits dans le cadre de la régression ridge à noyau (mise à jour séquentielle de matrices et maintien séquentiel de listes de vecteurs supports).

11.4.3 Exemple du bagging en ligne

Dans cette section, on suppose connue une méthode d'apprentissage pour laquelle on sait faire une mise à jour en ligne (cette méthode peut être la méthode des k-NN ou bien les méthodes à noyau comme décrites dans la section précédente). On notera donc $\mathcal{T}(\widehat{f}_n; (X_i,Y_i)_{i\in\tau})$ la mise à jour de l'estimateur $\widehat{f}_n$ avec les nouvelles observations $(X_i,Y_i)_{i\in\tau}$, τ étant un ensemble d'indices nouveaux. On s'intéresse alors à la méthode de bagging dans laquelle B échantillons bootstrap de l'ensemble $(X_i,Y_i)_{i=1,\dots,n}$ ont permis l'apprentissage des fonctions de régression $(\widehat{f}_n^b)_{b=1,\dots,B}$ qui sont combinées en une nouvelle fonction de régression

$$\widehat{f}_n = \frac{1}{B}\sum_{b=1}^{B} \widehat{f}_n^b. \tag{11.12}$$

Le principe du bagging en ligne (voir Oza and Russel [2001]) est basé, comme pour l'adaptation du bagging au cadre MR (voir section 11.3.3), sur l'utilisation du bootstrap Poisson . Le bootstrap Poisson est utilisé pour mettre à jour chaque échantillon bootstrap τ_b ($b = 1, \dots, B$) à partir des nouvelles observations $(X_i,Y_i)_{i\in\tau}$. De manière plus précise, pour chaque observation $i \in \tau$, une variable aléatoire n_i^b est simulée selon une loi de Poisson de paramètre 1 et l'échantillon $\tau_b^+ = \left\{ \underbrace{(X_i,Y_i) \dots (X_i,Y_i)}_{n_i^b \text{ fois}} \right\}_{i:\ n_i^b \neq 0}$ constitue alors l'échantillon de mise à jour du prédicteur correspondant à l'échantillon bootstrap numéro b. Finalement, la procédure de mise à jour en ligne est appliquée à chaque prédicteur $\widehat{f}_n^b$, $\mathcal{T}(\widehat{f}_n^b; \tau_b^+)$. L'étape de combinaison du prédicteur final reste inchangée et identique à celle décrite dans l'équation (11.12). La procédure précise est décrite dans l'algorithme 11.3.

Cette méthode s'applique directement à la mise à jour en ligne des forêts purement aléatoires [Breiman, 2000; Biau *et al.*, 2008; Arlot and Genuer, 2014] (voir chapitre **??**). Dans cette approche, le prédicteur bootstrap, $\widehat{f}_n$ est constitué de B arbres, $(\widehat{f}_n^b)_{b=1,\dots,B}$, dont les nœuds ont été obtenus sans tenir compte

Algorithme 11.3 Bagging en ligne

Nécessite $(\widehat{f}_n^b)_{b=1,\dots,B}$, les prédicteurs issus de B échantillons bootstrap de $(X_i,Y_i)_{i=1,\dots,n}$

Pour $b = 1 \to B$ **Faire**

Pour $i = n+1 \to n+m$ **Faire**

Générer $n_i^b \sim$ Poisson(1)

Fin Pour

Ensemble de mise à jour de l'échantillon bootstrap b : $\tau_b^+ = \left\{ \underbrace{(X_i,Y_i) \dots (X_i,Y_i)}_{n_i^b \text{ fois}} \right\}_{i:\ n_i^b \neq 0}$

Mise à jour du prédicteur bootstrap b :

$$\widehat{f}_{n+m}^b \leftarrow \mathcal{T}(\widehat{f}_n^b; \tau_b^+).$$

Fin Pour

Recalculer la fonction de prédiction :

$$\widehat{f}_{n+m} = \frac{1}{B} \sum_{b=1}^{B} \widehat{f}_{n+m}^b.$$

des données, de manière complètement aléatoires. De manière plus précise, chaque arbre contient Q nœuds (où Q est choisi en avance), construits de manière récursive, comme suit :

1. choisir aléatoirement un nœud $\mathcal{N}$ à diviser parmi toutes les feuilles de l'arbre en cours de construction ;
2. choisir aléatoirement une variable X^j parmi les variables $(X^j)_{j=1,\dots,p}$;
3. choisir aléatoirement de manière uniforme un point de coupe, $d \in [0,1]$, de manière à former les nœuds fils $\{X^j \geqslant d\}$ et $\{X^j < d\}$ parmi les observations de $\mathcal{N}$.

Les données d'apprentissage sont utilisées *a posteriori* pour définir la règle de décision : pour $\mathbf{x} \in \mathbb{R}^p$, $\widehat{f}_n^b(\mathbf{x})$ correspond, dans le cadre de la régression, à la moyenne des valeurs Y_i pour les observations X_i avec $(i \in \{1, \dots, n\})$ qui appartiennent au même nœud terminal que $\mathbf{x}$. La décision, $\widehat{f}_n(\mathbf{x})$ est obtenue de manière classique, par l'équation (11.12).

La mise à jour en ligne de $\widehat{f}_n^b$ est, dans ce cas-ci, relativement simple à mettre en œuvre et demande de conserver (et mettre à jour), pour chaque arbre $\widehat{f}_n^b$, les quantités suivantes :

- pour tous les nœuds terminaux $\mathcal{N}$ de $\widehat{f}_n^b$, on note $N_n^{b,\mathcal{N}}$ le nombre d'observations dans $(X_i)_{i=1,\dots,n}$ qui appartiennent au nœud $\mathcal{N}$;
- pour tous les nœuds terminaux $\mathcal{N}$ de $\widehat{f}_n^b$, on note $\overline{Y}_n^{b,\mathcal{N}}$ la moyenne des $N_n^{b,\mathcal{N}}$ valeurs Y_i pour les indices $i \in \{1, \dots, n\}$ des observations X_i qui appartiennent au nœud $\mathcal{N}$.

La mise à jour $(\widehat{f}_n^b; \tau_b^+)$ est alors donnée par le traitement séquentiel suivant des observations $i \in \{n+1, \dots, n+m\}$ qui vérifient $n_i^b \neq 0$:

- si on note $\mathcal{N}$ le nœud terminal de X_i dans $\widehat{f}_n^b$,

$$\overline{Y}_i^{b,\mathcal{N}} = \frac{N_{i-1}^{b,\mathcal{N}} \times \overline{Y}_{i-1}^{b,\mathcal{N}} + n_i^b \times Y_i}{N_{i-1}^{b,\mathcal{N}} + n_i^b}$$

  qui est basée sur une simple mise à jour en ligne de la moyenne dans le nœud terminal $\mathcal{N}$;
- $N_i^{b,\mathcal{N}} = N_{i-1}^{b,\mathcal{N}} + n_i^b$.

Cette description est une approche simplifiée de la version de Saffari *et al.* [2009] qui est décrite dans le chapitre **??**.

Remerciements

Nous tenons à remercier noes collègues qui ont participé à la réflexion sur ce sujet, en particulier, Jean-Michel Poggi, Robin Genuer, Christine Tuleau-Malot, Jérôme Mariette, Madalina Olteanu et Philippe Besse.

Bibliographie

Arlot, S. and Genuer, R. [2014]. Analysis of purely random forests bias. *arXiv preprint arXiv :1407.3939*.

Aronszajn, N. [1950]. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, **68**(3), 337–404.

Bach, F. [2013]. Sharp analysis of low-rank kernel matrix approximations. *Journal of Machine Learning Research, Workshop and Conference Proceedings*, **30**, 185–209.

Biau, G., Devroye, L. and Lugosi, G. [2008]. Consistency of random forests and other averaging classifiers. *Journal of Machine Learning Research*, **9**(Sep), 2015–2033.

Bickel, P., Götze, F. and van Zwet, W. [1997]. Resampling fewer than n observations : gains, losses and remedies for losses. *Statistica Sinica*, **7**(1), 1–31.

Bickel, P. and Sakov, A. [2008]. On the choice of m in the m out of n bootstrap and confidence bounds for extrema. *Statistica Sinica*, **18**(3), 967–985.

Breiman, L. [1996]. Bagging predictors. *Machine*, **24**, 123–140.

Breiman, L. [2000]. Some infinity theory for predictor ensembles. *Technical report*, Technical Report 579, Statistics Dept. UCB.

Chamandy, N., Muralidharan, O., Najmi, A. and Naidu, S. [2012]. Estimating uncertainty for massive data streams. *Technical report*, Google.

Chen, X. and Xie, M. [2014]. A split-and-conquer approach for analysis of extraordinarily large data. *Statistica Sinica*, **24**, 1655–1684.

Chu, C., Kim, S., Lin, Y., Yu, Y., Bradski, G., Ng, A. and Olukotun, K. [2010]. Map-Reduce for machine learning on multicore. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, A. Culotta (editors), *Advances in Neural Information Processing Systems (NIPS 2010)*, **23**, 281–288. Hyatt Regency, Vancouver, Canada.

Cortes, C., Mohri, M. and Talwalkar, A. [2010]. On the impact of kernel approximation on learning accuracy. *Journal of Machine Learning Research, Workshop and Conference Proceedings*, **9**, 113–120.

Dean, J. and Ghemawat, S. [2004]. MapReduce : simplified data processing on large clusters. In *Proceedings of Sixth Symposium on Operating System Design and Implementation (OSDI 2004)*, URL `http://research.google.com/archive/mapreduce.html`.

del Rio, S., López, V., Beniítez, J. and Herrera, F. [2014]. On the use of MapReduce for imbalanced big data using random forest. *Information Sciences*, **285**, 112–137. doi :10.1016/j.ins.2014.03.043.

Drineas, P. and Mahoney, M. [2005]. On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, **6**, 2153–2175.

Durut, M. and Rossi, F. [2011]. Communication Challenges in Cloud K-means. In *Proceedings of XIXth European Symposium on Artificial Neural Networks (ESANN 2011)*, 387–392. Bruges (Belgium).

Efron, B. [1979]. Bootstrap methods : another look at the Jackknife. *Annals of Statistics*, **1**.

Engel, Y., Mannor, S. and Meir, R. [2003]. The kernel recursive least squares algorithm. *IEEE Transactions on Signal Processing*, **52**(8), 2165–6176.

Fine, S. and Sheinberg, K. [2001]. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, **2**, 243–264.

Genuer, R., Poggi, J.-M., Tuleau-Malot, C. and Villa-Vialaneix, N. [2015]. Random Forests for Big Data. *ArXiv e-prints*. `1511.08327`.

Hanley, J. and MacGibbon, B. [2006]. Creating non-parametric bootstrap samples using Poisson frequencies. *Computer Methods and Programs in Biomedicine*, **83**(57-62).

Isard, M., Budiu, M., Yu, Y., Birrell, A. and Fetterly, D. [2007]. Dryad : distributed data-parallel programs from sequential building blocks. In *EuroSys '07 : Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, 59–72. ACM, New York, USA. ISBN 978-1-59593-636-3. doi :http://doi.acm.org/10.1145/1272996.1273005.

Jordan, M. [2013]. On statistics, computation and scalability. *Bernoulli*, **19**(4), 1378–1390. doi :10.3150/12-BEJSP17.

Kane, M., Emerson, J. and Weston, S. [2013]. Scalable strategies for computing with massive data. *Journal of Statistical Software*, **55**(14). ISSN 1548-7660. URL `http://www.jstatsoft.org/v55/i14`.

Kimeldorf, G. and Wahba, G. [1971]. Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, **33**(1), 82–95.

Kleiner, A., Talwalkar, A., Sarkar, P. and Jordan, M. [2012]. The big data bootstrap. In *Proceedings of 29th International Conference on Machine Learning (ICML 2012)*, Edinburgh, Scotland, UK.

Kleiner, A., Talwalkar, A., Sarkar, P. and Jordan, M. [2014]. A scalable bootstrap for massive data. *Journal of the Royal Statistical Society : Series B (Statistical Methodology)*, **76**(4), 795–816.

Kumar, S., Mohri, M. and Talwalkar, A. [2012]. Sampling techniques for the Nyström method. *Journal of Machine Learning Research*, **13**, 981–1006.

Laskov, P., Gehl, C., Krüger, S. and Müller, K. [2006]. Incremental support vector learning : analysis, implementation and application. *Journal of Machine Learning Research*, **7**, 1909–1936.

Lee, H. and Clyde, M. [2004]. Online Bayesian bagging. *Journal of Maching Learning Research*, **5**, 143–151.

Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C. and Hellerstein, J. M. [2010]. GraphLab : A New Parallel Framework for Machine Learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, Catalina Island, California.

Oza, N. and Russel, S. [2001]. Online bagging and boosting. In M. Kaufmann (editor), *Proceedings of Eighth International Workshop on Artificial Intelligence and Statistics*, 105–112. Key West, Florida, USA.

Press, W., Teukolsky, S., Vetterling, W. and Flannery, B. [1992]. *Numerical Recipes in C*. Cambridge University Press, 2nd edition.

Saffari, A., Leistner, C., Santner, J., Godec, M. and Bischof, H. [2009]. On-line random forests. In *Proceedings of IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops)*, 1393–1400. IEEE.

Saunders, G., Gammerman, A. and Vovk, V. [1998]. Ridge regression learning algorithm in dual variables. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML'98)*, 515–521. Madison, Wisconsin, USA.

Schölkopf, B., Herbrich, R. and Smola, A. [2001]. A generalized representer theorem. In D. Heimbold, B. Williamson (editors), *Proceedings of the 14th Conference on Computational Learning Theory (COLT)*, volume 2111 of *Lecture Notes in Computer Science*, 416–426. Springer Berlin Heidelberg.

Schölkopf, B., Smola, A. and Müller, K. [1998]. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, **10**(5), 1299–1319.

Tibshirani, R. [1996]. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, series B*, **58**(1), 267–288.

Tikhonov, A. [1963]. Solution of incorrectly formulated problems and the regularization method. *Soviet mathematics - Doklady*, **4**, 1036–1038.

van Vaerenbergh, S., Lázaro-Gredilla, M. and Santamaría, I. [2012]. Kernel recursive least-squares tracker for time-varying regression. *IEEE Transactions on Neural Networks and Learning Systems*, **23**(8), 1313–1326.

van Vaerenbergh, S. and Santamaría, I. [2014]. Online regression with kernels. In J. Suykens, M. Signoretto, A. Argyriou (editors), *Regularization, Optimization, Kernels, and Support Vector Machines*, CRC Machine Learning & Pattern Recognition. CRC Press, Taylor & Francis Group.

Williams, C. and Seeger, M. [2000]. Using the Nyström method to speed up kernel machines. In T. Leen, T. Dietterich, V. Tresp (editors), *Advances in Neural Information Processing Systems (Proceedings of NIPS 2000)*, **13**. Neural Information Processing Systems Foundation, Denver, CO, USA.

Zaharia, M., Chowdhury, M., Das, D., Dave, A., Ma, J., McCauly, M., Franklin, M. J.and Shenker, S. and Stoica, I. [2012]. Resilient Distributed Datasets : A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 15–28. USENIX.

Index