TELECOM ParisTech **MDI341**
2020 **A. Bellet, G. Staerman**

TP n° 2 : Méthodes à noyaux à grande échelle

Le compte rendu de TP est à rendre avant lundi 16/03/2020, 23h55. Vous devez déposer un **UNIQUE** fichier sous format ipynb (ipython notebook) sur le site pédagogique du cours, dans le dossier correspondant au TP (Large-scale kernel methods > Rendu TP). Merci d'effectuer **un seul rendu par groupe de 2 ou 3**.

La note totale est sur **20** points, répartis comme suit :

— qualité des réponses aux questions : **17** pts,

— qualité de rédaction, de présentation et d'orthographe : **2** pts,

— absence de bug : **1** pt.

Malus : **5** pts par tranche de 12h de retard ; 2 pts pour non respect des autres consignes de rendu.

Rappel : aucun travail par mail accepté

Introduction

On s'intéressera dans ce TP au passage à l'échelle des classifieurs de type SVM à noyaux sur des données de grande taille. On va étudier des techniques permettant d'approximer un noyau afin de se ramener à un problème de SVM linéaire, dont la résolution peut être obtenue de manière beaucoup plus efficace. On rappelle ici le cadre de la classification supervisée, et l'on présente les notations que l'on utilisera par la suite :

— $\mathcal{Y}$ l'ensemble des étiquettes des données (*labels* en anglais).

— $\mathbf{x} = (x_1, \dots, x_p)^\top \in \mathcal{X} \subset \mathbb{R}^p$ est une observation (ou un *sample* en anglais). La $j$ème coordonnée de $\mathbf{x}$ est la valeur prise par la $j$ème variable (*feature* en anglais).

— L'ensemble d'apprentissage sera noté $(X^{\text{train}}, y^{\text{train}})$ où $X^{\text{train}} = [\mathbf{x}_1^{\text{train}}, \dots, \mathbf{x}_{n_1}^{\text{train}}]^\top$, et l'ensemble de test $(X^{\text{test}}, y^{\text{test}})$ où $X^{\text{test}} = [\mathbf{x}_1^{\text{test}}, \dots, \mathbf{x}_{n_2}^{\text{test}}]^\top$.

Préliminaires

Le code au début du notebook qui vous est fourni permet de charger les données de la base de classification binaire ijcnn1, de sélectionner 60000 observations et de les normaliser avec StandardScaler. On sépare ensuite les données en deux ensembles $(X^{\text{train}}, y^{\text{train}})$ et $(X^{\text{test}}, y^{\text{test}})$ en utilisant train_test_split du module sklearn.model_selection avec l'option train_size=20000. On a donc $n_1 = 20000$ et $n_2 = 40000$.

1) Entraîner un SVM linéaire (sans noyau) et un SVM non linéaire (avec noyau) sur les données d'apprentissage. Utiliser les deux modèles pour prédire les étiquettes de l'ensemble de test. Pour cela, on utilisera LinearSVC avec option dual=**False** pour le SVM linéaire, et SVC avec option rbf (noyau Gaussien) pour le SVM non linéaire. Par simplicité, on pourra laisser les autres paramètres à leur valeur par défaut. Comparer le temps pris pour faire l'apprentissage avec les deux implémentations, puis pour faire la prédiction sur l'ensemble test. Comparer également leur score de prédiction. Vous commenterez les résultats. Utiliser le module time de la librairie standard pour mesurer le temps de calcul. Exemple pour récupérer l'instant courant utiliser : **import** time; t = time.time().

Rappels sur la SVD / décomposition spectrale

Dans les méthodes à noyau (type SVM), il est nécessaire de former la matrice de Gram des observations. Étant donnés un ensemble d'observations $\mathbf{x}_1, \dots, \mathbf{x}_n$ où chaque $\mathbf{x}_i \in \mathbb{R}^p$ et un noyau K, la matrice de Gram $G \in \mathbb{R}^{n \times n}$ est donnée par $G_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$. Hélas, si le nombre n d'observations est grand, il est coûteux de construire, stocker et manipuler une telle matrice.

Il peut ainsi être utile d'approcher la matrice G par une matrice qui se prête mieux aux calculs numériques. Une approximation classique consiste à utiliser une matrice de faible rang.

2) Compléter la fonction rank_trunc qui prend en entrée une matrice symétrique $G \in \mathbb{R}^{n\times n}$ et un entier $k \in \mathbb{N}$ et qui renvoie la meilleure approximation de rang k de G (au sens de la norme de Frobenius), notée G_k. Pour cela on utilisera la décomposition spectrale de G tronquée à l'ordre k. On proposera une option "rapide" utilisant la fonction scipy.sparse.linalg.svds, et une option "lente" utilisant la fonction scipy.linalg.svd.

3) Appliquer cette méthode sur une matrice G créée comme suit. On tire d'abord une matrice N^{bruit} dont les entrées sont des Gaussiennes (centrées réduites) de taille 100×200, et $G^{\text{bruit}} = N_{\text{bruit}}^\top N_{\text{bruit}}$. On tire ensuite une matrice N^{signal} dont les entrées sont des Gaussiennes (centrées réduites) de taille 20×200, et $G^{\text{signal}} = N_{\text{signal}}^\top N_{\text{signal}}$. On prend enfin $G = 50G^{\text{signal}} + G^{\text{bruit}}$. Afficher une courbe donnant l'écart relatif en norme de Frobenius (*i.e.,* $\|G_k - G\|_{Fro}/\|G\|_{Fro}$) en fonction de l'ordre k utilisé, pour $k = 1$ à $k = 100$. On affichera également le temps de calcul pour la version "rapide" et "lente" sur une même courbe pour les mêmes valeurs de k. Commenter les résultats.

Random Kernel Features

Quand les données sont volumineuses, il devient trop coûteux de calculer/stocker la matrice de Gram nécessaire pour les SVM à noyaux, et on ne peut donc pas calculer la meilleure approximation de rang faible comme dans la partie précédente.

On va étudier ici une autre stratégie. Rappelons que si K est une fonction noyau, il existe un certain $\phi : \mathbb{R}^p \to \mathcal{H}$ (où $\mathcal{H}$ est potentiellement de dimension infinie) tel que $K(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}')$, $\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$. La technique dite des *Random Kernel Features* est d'approcher $\phi(\mathbf{x}_i)$ par c *features* aléatoires ($c < +\infty$) afin de pouvoir appliquer une méthode linéaire efficace tout en bénéficiant de l'amélioration possible du score de prédiction grâce à la non-linéarité du noyau. Ici encore on prendra un noyau Gaussien (RBF) : $K_\gamma(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$. L'approche est basée sur des projections aléatoires et est simple à mettre en œuvre. Elle est résumée dans l'Algorithme 1.

Algorithme 1 : Random Kernel Features

Data : $X^{\text{train}} \in \mathbb{R}^{n_1\times p}$ matrice des données, $X^{\text{test}} \in \mathbb{R}^{n_2\times p}$, nombre de *features* aléatoires $c \geq 1$, paramètre du noyau Gaussien γ

Result : Nouvelle représentation des données $Z^{\text{train}} \in \mathbb{R}^{n_1\times c}$, $Z^{\text{test}} \in \mathbb{R}^{n_2\times c}$

Générer une matrice $W \in \mathbb{R}^{p\times c}$ dont les entrées sont indépendantes et tirées selon $\mathcal{N}(0, 2\gamma)$

Générer un vecteur $b \in \mathbb{R}^{1\times c}$ dont les entrées sont indépendantes et uniformes sur $[0, 2\pi]$

Calculer $Z^{\text{train}}_{i,:} = \sqrt{2/c}\cos(\mathbf{x}_i^{\text{train}} W + b)$ avec $i \in \{1, \ldots, n_1\}$

Calculer $Z^{\text{test}}_{i,:} = \sqrt{2/c}\cos(\mathbf{x}_i^{\text{test}} W + b)$ avec $i \in \{1, \ldots, n_2\}$

4) Compléter la fonction random_features qui permet de mettre en œuvre cette approche, c'est-à-dire qui crée les matrices Z^{test} et Z^{train} à partir des matrices X^{test} et X^{train} originales.

5) Appliquer cette méthode avec $c = 300$ et le paramètre du noyau par défaut de scikit-learn ($\gamma = 1/p$). Entraîner un modèle LinearSVC sur $(Z^{\text{train}}, y^{\text{train}})$ et tester sa performance sur $(Z^{\text{test}}, y^{\text{test}})$. Comparer la performance en temps et en score par rapport aux résultats de la question 1.

L'approximation de Nyström

L'approximation de Nyström est une alternative à la méthode précédente qui permet de calculer une approximation de la matrice de Gram en utilisant un sous-ensemble aléatoire de ses colonnes. Comme les *Random Kernel Features*, cette méthode permet aussi de calculer des *features* explicites et d'être ainsi combinée à une méthode linéaire. L'approximation de Nyström est résumée dans l'Algorithme 2.

6) Compléter la fonction nystrom qui reproduit l'algorithme 2 pour le noyau Gaussien. La fonction sklearn.metrics.pairwise.rbf_kernel vous sera utile pour calculer les valeurs de noyau. Conseil : utiliser scipy.linalg.svd pour la décomposition spectrale de de W, le résultat est plus stable.

7) Appliquer la méthode de Nyström aux même données qu'avant, en choisissant $c = 500$, $k = 300$ et $\gamma = 1/p$. Appliquer un SVM linéaire et discuter de nouveau les performances en temps et en score pour cette méthode.

Algorithme 2 : Approximation de Nyström

Data : $X^{\text{train}} \in \mathbb{R}^{n_1 \times p}$ matrice des données, $X^{\text{test}} \in \mathbb{R}^{n_2 \times p}$, nombre de colonnes $c \leq n_1$, nombre de valeurs singulières conservées $k \leq n_1$, paramètre du noyau Gaussien γ

Result : $Z^{\text{train}} \in \mathbb{R}^{n_1 \times k}$, $Z^{\text{test}} \in \mathbb{R}^{n_2 \times k}$

Tirer uniformément (avec remise) c lignes de $X = X^{\text{train}}$. On note I les indices des lignes tirées.

Créer la matrice de Gram réduite $W_{ij} = K_\gamma(\mathbf{x}_i^{\text{train}}, \mathbf{x}_j^{\text{train}})$ pour $(i,j) \in I^2$ ($W \in \mathbb{R}^{c \times c}$)

Calculer $W_k = V_k \Sigma_k V_k^\top$ décomposition spectrale tronquée à l'ordre k de W ($V_k \in \mathbb{R}^{c \times k}$, et $\Sigma \in \mathbb{R}^{k \times k}$ diagonale)

Calculer $M_k = V_k \Sigma_k^{-1/2} \in \mathbb{R}^{c \times k}$

Calculer $C^{\text{train}} \in \mathbb{R}^{n_1 \times c}$ avec $C_{ij}^{\text{train}} = K_\gamma(\mathbf{x}_i^{\text{train}}, \mathbf{x}_j^{\text{train}})$ avec $i \in \{1, \dots, n_1\}, j \in I$

Calculer $C^{\text{test}} \in \mathbb{R}^{n_2 \times c}$ avec $C_{ij}^{\text{test}} = K_\gamma(\mathbf{x}_i^{\text{test}}, \mathbf{x}_j^{\text{train}})$ avec $i \in \{1, \dots, n_2\}, j \in I$

Calculer $Z^{\text{train}} = C^{\text{train}} M_k$ et $Z^{\text{test}} = C^{\text{test}} M_k$

Synthèse des résultats

8) Synthétiser dans des graphiques la performance des Random Kernel Features et de Nyström en terme de score de prédiction et de temps calcul total (temps pris pour entraîner le modèle + temps pour prédire) en fonction de c. Pour Nyström, on pourra fixer $k = c - 10$ et faire varier les valeurs de c sur une grille en utilisant par exemple **range**(20, 600, 50). Tracer également les performances du SVM linéaire et du SVM à noyaux classiques entraînés en question 1. Commenter.

Pour aller plus loin

Pour ceux qui veulent en savoir plus sur ces méthodes :

— Articles introductifs : [1, 2, 3]

— Pour aller plus loin dans la compréhension de la méthode Nyström SVM : http://www.jmlr.org/proceedings/papers/v22/zhang12d/zhang12d.pdf

— Adapter la méthodologie sur le noyau de Laplace.

— Combiner les features obtenus avec différents noyaux.

Références

[1] P. Drineas and M. W. Mahoney. On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *J. Mach. Learn. Res.*, 6 :2153–2175, 2005. 3

[2] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In J.C. Platt, D. Koller, Y. Singer, and S.T. Roweis, editors, *NIPS*, pages 1177–1184. Curran Associates, Inc., 2008. 3

[3] A. Rahimi and B. Recht. Weighted sums of random kitchen sinks : Replacing minimization with randomization in learning. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *NIPS*, pages 1313–1320. Curran Associates, Inc., 2009. 3