Fouille de Données Massives
TD - Autour du SVM

M2 Informatique - SISE

Guillaume Metzler

Institut de Communication (ICOM)
Université de Lyon, Université Lumière Lyon 2
Laboratoire ERIC UR 3083, Lyon, France

guillaume.metzler@univ-lyon2.fr

Abstract

L'objectif de cette séance est de mettre en œuvre l'algorithme des SVM sur différents jeux de données présentant des caractéristiques différentes afin de mieux saisir l'impact des différents paramètres.

Nous allons, par exemple, étudier l'impact du nombre d'exemples sur la vitesse d'apprentissage du modèle, mais aussi le rôle des différents hyper-paramètres sur les performances du modèle appris.
Ce dernier point va montrer l'intérêt d'effectuer de la cross-validation pour tuner ces hyper-paramètres là.

1 Mise en oeuvre

Dans cette première partie, nous allons implémenter l'algorithme des séparateurs à vaste marge sur des jeux de données présentant des caractéristiques différentes. On va considérer les jeux de données : Balance, Pima, Satimage, Segmentation, Yeast6.

Pour ces différents jeux de données, vous allez apprendre, en utilisant une 5-CV

- un SVM avec noyau linéaire en tunant l'hyper-paramètre C parmi les valeurs $\{0.1, 0.5, 1, 2, 4\}$
- un SVM avec un noyau gaussien (ou *radial* sous Python) en tunant l'hyper-paramètre C parmi les valeurs $\{0.1, 0.5, 1, 2, 4\}$ et le paramètre γ parmi les valeurs $\{0.01, 0.1, 1, 10\}$

On souhaite ensuite étudier les performances de ces différents algorithmes.

1. Observer les performances sur les différents jeux de données
2. Comparer les résultats obtenus entre un SVM linéaire et une régression logistique (on prendra un seuil de 0.5 pour l'affectation dans une classe), les résultats sont-ils différents ?

On va maintenant se concentrer sur les jeux de données Satimage, Segmentation et Yeast6 en se concentrant sur le SVM linéaire avec l'hyper-paramètre C égal à 1.

3. Que peut-on dire du taux de classification sur la classe positive et sur la classe négative pour ces trois jeux de données ? On pourra par exemple étudier les matrices de confusion pour ces trois jeux de données.
4. Utiliser le paramètre `class_weight` avec l'option "balanced". Cela permet de modifier le poids des classes lorsque les jeux de données est déséquilibré, de sorte à ce que les classes en présence aient le même poids. Comparer les nouvelles matrices de confusions à celles obtenues à la question précédente.
5. A l'aide des options `dual_coef_` et `support_`, comparer le nombre d'exemples de votre jeu de données, à la dimension de ces deux objets. Que remarquez vous ?
6. En vous basant sur la relation permettant de prédire l'étiquette d'une donnée

$$h(\mathbf{x}') = \text{sign}\left(\sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}', \mathbf{x}_i)\right).$$

et de l'aide concernant la fonction SVC et l'output de l'option `dual_coef_`, quels sont les exemples qui participent à la construction de la frontière de décision ?

2 Evaluation du temps de calcul

L'objectif de cette section est d'évaluer le temps d'apprentissage d'un SVM avec un noyau linéaire et gaussien afin de voir quelles sont les limites de cet algorithme.

Pour cela, on va considérer un jeu de données synthétiques que l'on pourra générer à partir du code qui se trouve au lien suivant

Jeux de données

Les performances des modèles ont peu d'importance dans cette section, on va simplement étudier le temps d'apprentissage, on fixera donc $C = 1$ et $\gamma = 1$ pour les versions linéaire et gaussien des méthodes à noyaux.

1. Représenter, sur un graphique le temps d'apprentissage d'un SVM linéaire et/ou gaussien en fonction de la taille de l'échantillon.
2. Est-ce que cet algorithme est adapté dans un contexte de données massives ? Pourquoi ? Justifier en vous basant sur vos connaissances sur ces méthodes là.
3. Quelles méthodes vues en cours, ou parmi vos connaissances personnelles, pourriez vous employer pour réduire ce temps d'apprentissage.

3 Vers une nécessité de faire une approximation

Les expériences précédentes ont montré que les méthodes à noyaux sont très limitées lorsque le nombre de données est considérable. Il convient donc de trouver des approches permettant de palier à ce problème.

Dans cette section, on se propose d'employer d'étudier deux méthodes qui consistent à employer des *points de repères*, appelés aussi des *landmarks*, pour l'apprentissage des noyaux.

On rappelle que les méthodes à noyaux nécessitent d'apprendre toute les similarités deux à deux $k(\mathbf{x}_i, \mathbf{x}_j)$ de la forme

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right).$$

Ce qui nécessite de calculer m^2 similarités, où m représente le nombre d'exemples. Le modèle appris comporte également m paramètres qui sont donc à apprendre.

L'idée est de chercher des points de repère (disons $k < m$ points) dans notre jeu de données afin d'apprendre notre modèle, pour ainsi réduire le nombre de similarités à calculer ainsi que le nombre d'exemples.
Ces points de repères peuvent être sélectionnés

- aléatoirement
- à l'aide d'un algorithme, ici on se propose de faire cela avec un algorithme k-means pour une certaine valeurs de k

Malheureusement, il n'existe pas de librairies sous Python permettant de faire cela d'un bloc et il faudra décomposer votre code en deux étapes :

1. Ecrire un code Python permettant de
   (a) sélectionner un nombre fixe de landmarks dans votre jeu de données (aléatoirement ou avec un k-means)
   (b) calculer la similarité entre vos landmarks (linéaire ou gaussien)
   (c) apprendre votre approximation de la méthode à noyau à partir des landmarks
2. Comparer les résultats obtenus ainsi que le temps d'apprentissage entre le modèle initial et la version approchée.
3. Est-ce judicieux de choisir les points aléatoirement ? Quels sont les avantages/inconvénients de faire cela avec un algorithme k-means ?