Apprentissage statistique: DM
Implémentation de quelques algorithmes de classification

Emile Contal

pour le 25 mars 2016

Table des matières

1 Introduction

Ce projet consiste en l'implémentation de quelques algorithmes de classification vus en cours, par binôme, en Matlab. Il faudra appliquer ces algorithmes sur quatre jeux de données : deux simulés, et deux provenant de réels problèmes académiques. Le but de ce travail est de comparer empiriquement les avantages et inconvénients pratiques de ces algorithmes. Vous devrez fournir votre code ainsi qu'un rapport comportant vos résultats expérimentaux commentés.

2 Les Jeux de Données

Les données sont accessibles sur la page `http://econtal.perso.math.cnrs.fr/teaching/apprentissage/dm.zip`. Dans les quatre cas les variables ont été normalisées à $\mathcal{X} \subseteq [0,1]^d$ et $\mathcal{Y} = \{0,1\}$ pour simplifier. Vous pourrez les récupérer sous forme matricielle avec la commande Matlab `load(filename)`. Les variables `Xtr` et `Ytr` de taille respectives (n,d) et $(1,n)$ correspondent aux données d'entrées de l'algorithme [1]. On évaluera empiriquement le risque quadratique d'un classifieur g en calculant :

$$\texttt{mean((g(Xte)-Yte)\^{}2)}\,, \tag{1}$$

où l'on suppose que `g` appliqué à la matrice `Xte` de taille (t,d) renvoie un vecteur dans $\{0,1\}^t$.

2.1 Un Disque dans le Plan

L'exemple le plus simple est `disc.mat`, où les données suivent la loi

$$\Pr\left[Y = 1 \mid X = x\right] = \begin{cases} 0.8 & \text{si } \mathbb{1}_{\|x-c\|<0.4} \\ 0.2 & \text{sinon,} \end{cases}$$

avec $c = [\frac{1}{2}, \frac{1}{2}]$ et X suit une loi uniforme.

2.2 Un Mélange de Gaussiennes en Dimension 3

Les données du fichier `gaussian_mix.mat` ont été générées suivant deux Gaussiennes de paramètres différents :

$$\begin{aligned} \mathcal{L}(X \mid Y = 0) &= \mathcal{N}(m_0, \Sigma_0) \\ \mathcal{L}(X \mid Y = 1) &= \mathcal{N}(m_1, \Sigma_1)\,. \end{aligned}$$

et $\Pr[Y = 0] = 1/2$.

2.3 Astrologie

Le jeu de données `astroparticles.mat` provient d'un problème réel de classification en astrologie (par Jan Conrad de Uppsala University). Il comporte 3089 données en dimension 4.

2.4 Bioinformatique

Le jeu de données `bioinformatics.mat` provient d'un problème réel de classification de code ARN (par Andrew Uzilov, Joshua Keegan et David Mathews). Il comporte 59535 données en dimension 8.

3 Les Algorithmes

Cette section comporte les quatre algorithmes à implémenter.

3.1 Le Perceptron et les Noyaux

Cet algorithme correspond à celui vu au TD3, Exercice 1, paragraphe 3. Vous choisirez comme noyau la fonction suivante :

$$k(a,b) = e^{-\frac{\|a-b\|_2^2}{\ell^2}}\,,$$

où ℓ est un paramètre à définir.

1. `Xtr` pour *train*, `Xte` pour *test*

3.2 Les k-Plus-Proches-Voisins

Il s'agit de la version simple des Plus-Proches-Voisins, telle que définie dans le TD5, où vous choisirez k en fonction de n tel que $k \to \infty$ et $\frac{k}{n} \to 0$.

3.3 Le Plus-Proche-Voisin Randomisé

Cet algorithme est une approximation empirique du classifieur décrit au TD7, Exercice 1. La règle du plus-proche-voisin randomisé reste identique, avec $m \to \infty$ et $\frac{m}{n} \to 0$, mais au lieu de prendre le classifieur par espérance vous implémenterez le classifieur par vote sur l réalisations de la variable uniforme U. Le nombre l est un paramètre à définir (par exemple le plus grand nombre tel que le coût en calcul ne soit pas rébarbatif).

3.4 Les Forêts Aléatoires

On s'intéresse ici aux forêts purement aléatoires telles que définies dans le TD6, Exercice 3. Vous choisirez une taille des arbres k fonction de n telle que $k \to \infty$ et $\frac{k}{n} \to 0$. Et le nombre d'arbres l est un paramètre à définir (par exemple comme précédemment).

4 Consignes

4.1 Tâches à faire

Les algorithmes. Vous implémenterez les algorithmes précédents sous forme de fonctions Matlab dans des fichiers séparés, de sorte qu'il soit aisé de les appliquer à l'un ou l'autre jeu de données. Pour que le temps de calcul ne soit pas trop long, il faudra veiller à utiliser le moins possible les boucles `for` ou `while` quand il s'agit de parcourir un grand ensemble, mais préférer les calculs matriciels. Vos fonctions de classifications devraient idéalement calculer les prédictions sur une matrice d'entrée `Xte` de taille (t, d), sans faire une boucle sur t. Il est aisé d'écrire l'algorithme du Perceptron de telle sorte, mais plus technique avec les arbres, voir en Section 5 pour quelques pistes.

Le choix des paramètres. Les définitions des algorithmes précédents impliquent des paramètres inconnus. Vous choisirez et commenterez une méthode empirique pour déterminer de manière automatique la valeur de ces paramètres en fonction des données d'entrées seulement (`Xtr` et `Ytr`).

4.2 Le Code

Vous apporterez suffisamment de soin pour une lecture décente de votre code (noms de variables explicites, quelques commentaires, etc.). De même, vous découperez les sous-tâches en fonctions pour éviter de copier-coller des blocs de code.

4.3 Les Expériences

Pour chaque jeu de données il faudra réaliser l'expérience suivante. Si `Xtr` est de taille (n, d), on choisira N entiers croissants $n_1 < \cdots < n_N = n$. Vous entraînerez alors chaque algorithme successivement sur les N sous-ensembles :

$$\big(\texttt{Xtr}(1{:}n_1, :), \texttt{Ytr}(1{:}n_1)\big), \ldots, \big(\texttt{Xtr}(1{:}n_N, :), \texttt{Ytr}(1{:}n_N)\big)\,.$$

A chaque étape, vous mesurerez l'erreur empirique sur $(\texttt{Xte}, \texttt{Yte})$ avec l'Equation 1. Vous reporterez cette erreur en fonction de n_i, sur la même figure pour les 4 algorithmes. Codée naïvement, cette procédure sera coûteuse en calcul et N devra être petit. En implémentant pour chaque algorithme une fonction de mise à jour efficace du classifieur après ajout des nouvelles données $\big(\texttt{Xtr}(n_i{:}n_{i+1}, :), \texttt{Ytr}(n_i{:}n_{i+1})\big)$, le coût en calcul sera grandement réduit et vous pourrez augmenter N pour raffiner vos résultats.

4.4 Rapport

Votre rapport devra expliquer les choix que vous aurez effectués et les difficultés rencontrées. Préférez un rapport succinct, trois pages suffisent. Il comportera les quatre figures, une par jeu de données, issues de l'expérience précédente.

5 Outils Utiles pour le Calcul Matriciel

Cette section comporte quelques conseils pour vous aider à écrire les algorithmes de façon matricielle, sans utiliser de boucle `for` ou `while`.

5.1 Les Tests

Soit `X`, `Y`, `T`, `Right` et `Left` des vecteurs de taille `n`. Plutôt que d'écrire :

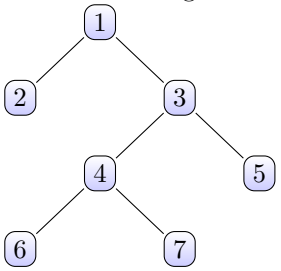
```
for i=1:n
    if(X(i) > T(i))
        Y(i) = Right(i);
    else
        Y(i) = Left(i);
    end
end
```

on peut écrire de façon efficace :

```
isGreater = X > T;
Y(isGreater) = Right(isGreater);
Y(~isGreater) = Left(~isGreater);
```

5.2 Implémentation Vectorielle des Arbres

Un arbre binaire de n noeuds peut être représenté vectoriellement par deux vecteurs de taille n, comportant respectivement les indices des fils gauches et les indices des fils droits, par exemple :




```
LeftChilds = [2 0 4 6 0 0 0];
RightChilds = [3 0 5 7 0 0 0];
```

avec la convention 0 pour les feuilles.

5.3 Indices dans une Forêt (avancé)

De même, il est alors possible de représenter une forêts de m arbres de taille k par deux matrices `L` et `R` de taille (m, k). Il n'est pas évident de manipuler des vecteurs d'indices dans une telle forêt. Si l'on dispose d'un vecteur `I` d'indices de taille m, c'est à dire un noeud par arbre, on peut accéder aux fils de ces noeuds avec la fonction `sub2ind` (voir documentation sur www.mathworks.com) :

```
LeftChildsOfI = L(sub2ind(size(L), 1:m, I));
RightChildsOfI = R(sub2ind(size(L), 1:m, I));
```