Apprentissage Automatique, TP2

M1, année 2016-2017

Méthode des K-NN pour la reconnaissance de caractères manuscrits

1 Introduction

La méthode des K-NN (K-Nearest Neighbors) ou des K-PPV (K-Plus Proches Voisins) en français permet, en apprentissage supervisé, de prédire la classe d'une donnée par sa "proximité" avec d'autres données déjà classées.
Cette "proximité" est calculée par une distance qui peut être de plusieurs natures. Classiquement, on utilise une distance euclidienne lorsque les données sont représentées par des nombres. Cette distance est calculable sur n dimensions. La formule mathématique qui calcule la distance euclidienne entre deux points dans un espace à n dimensions est la suivante :

$$d(a,b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

2 But du TP

Lors de ce TP, nous allons appliquer la méthodes des K-NN à la reconnaissance de caractères manuscrits (chiffres).
Les données proviennent de la base publique de test MNIST.
Les images sont linéarisées sous forme de vecteurs. Par exemple, une image de dimension 28x28 sera representée par un vecteur de 784 entiers. Ainsi en Java, un glyphe sera representé par un tableau d'entiers sur un octet (`byte[] glyph=new byte[784];`), chaque entier representant les valeurs d'un niveau de gris de 0 à 127.
Les glyphes sont déjà classés et sont sauvegardés dans un fichier binaire par classe (respectivement les fichiers classe0, classe1, ..., classe9).

3 Chargement des données

Pour manipuler les données, une classe Java nommée `Utils` est fournie qui offre un certain nombre de fonctionnalités :

— **Chargement des données :** la méthode `loadImages()` permet de charger un jeu d'images classifiées. Elle retourne une double liste de glyphes. Le chemin passé en paramètre est le préfixe des noms des fichiers à charger.
— **Les glyphes et leurs étiquettes :** les glyphes auront besoin d'être placés dans des ensembles avec leur étiquette. La classe `LabelledData` vous évitera d'avoir à implementer cette association.
— **Visualisation des glyphes :** la méthode `viewGlyph(byte[] glyph)` créée un `JFrame` pour afficher un glyphe dans une fenêtre.

4 Fonctions de calcul et préparation des jeux de données

Question 1.1 : Premier contact avec les données : écrivez un programme qui charge le jeu de données comme indiqué, et utilisez la fonction `Utils.viewGlyph()` pour afficher un glyphe de votre choix.
Question 1.2 : Écrivez une fonction qui calcule la distance euclidienne entre deux glyphes. Pour plus de précision, celle-ci retournera le résultat du calcul sous forme d'une valeur de type `double`.
Question 1.3 : Écrivez une fonction qui prendra en paramètre l'ensemble des glyphes chargés et retournera N glyphes de chaque classe choisis au hasard (`ArrayList<LabelledData>`). NB : attention, les fonctions `Random` de Java ont tendance à renvoyer souvent la même valeur quand on les appelle trop fréquemment dans un délai très court, il faut contourner ce problème.
Question 1.4 : Implémentez une fonction qui, étant un glyphe particulier donne en paramètre, une liste de glyphes étiquetés (`ArrayList<LabelledData>`) ainsi qu'une valeur pour k, retourne une sous-liste composée des k plus proches voisins. (Indication : on pourra utiliser une structure de données comme `TreeMap<Double,LabelledData>` pour faciliter le tri des données par leurs distances)
Question 1.5 : Écrivez une fonction qui, à partir d'une liste de voisins, détermine la classe du glyphe concerné.
Question 1.6 : Choisissez quelques glyphes dans le jeu de données et appliquez leur la classification par la méthode des KNN. Pensez bien à afficher les classes des plus proches voisins trouvés et la classe finale pour vérifier vos algorithmes.

5 Évaluation du biais

La méthode *semble* efficace. Estimons ses taux d'erreurs en faisant varier k.
Question 2.1 : En repartant de la fonction écrite en **Q1.3**, écrivez une nouvelle fonction qui construit m ensembles de glyphes étiquetés où chaque classe est représentée n fois. **Attention, ces ensembles doivent être disjoints, c'est-à-dire qu'un glyphe ne doit pas se retrouver dans plusieurs sous-ensembles.** (Indication : une solution possible consiste à enlever le glyphe du jeu de données initial quand il est choisi afin d'éviter de le retrouver plus tard dans un autre sous-ensemble)
Question 2.2 : Observons l'importance de k sur la qualité des résultats : écrivez une fonction qui évalue le taux d'erreurs d'apprentissage. Pour cela, constituez 20 ensembles de 100 glyphes de chaque classe qui serviront d'ensembles d'apprentissage et également un ensemble de test de 50 éléments (encore une fois disjoints des éléments d'apprentissage). Calculez ensuite le taux d'erreurs

de prédictions des données test contre chaque ensemble d'apprentissage. Cela donnera une moyenne de taux d'erreurs pour chaque valeur de k.
Le programme produira donc en sortie un tableau avec 2 colonnes : la valeur de k et la moyenne des taux d'erreurs. Faites varier k de 1 à 200.
Question 2.3 : Tracez vos résultats sur un graphique en utilisant le programme de votre choix, comme R par exemple.
Question 2.4 : Regardons maintenant l'influence de la taille du modèle. Faites alors varier ce nombre de 10 à 300 pour une valeur fixe de k. Tracez vos résultats sur un graphique.