Application de la méthode des $k - NN$ à la reconnaissance des caractères manuscrits

Introduction

Nous profiterons de ce TP pour faire une introduction à l'utilisation d'outils pour la rédaction scientifique et technique : LaTeX, gnuplot, ... Vous trouverez dans les fichiers qui accompagne ce TP le fichier aatp-latex.tex qui pourra vous servir d'exemple pour la rédaction de voter compte rendu.

Le travail à rendre : une archive contenant

- le rapport au format pdf, ainsi que le code source en LaTeX avec toutes les ressources incluses,
- le code java complet.

Nous allons appliquer dans ce tp la méthodes des k-plus-proches-voisins à la reconnaissances de caractères manuscrits (des chiffres).

Les données proviennent de la base de test MNIST.

Les images sont linéarisées sous forme de vecteurs. Par exemple, une image de dimension $28x28$ sera représentées par un vecteur de 784 entier. Ainsi en Java, un glyphe sera représenté par un tableau d'entiers sur un octet (`byte[] glyph=new byte[784];`), chaque entier représentant les valeurs d'un niveau de gris de 0 à 127.

Les glyphes sont déjà classifiés et sont sauvegardés dans un fichier binaire par classe (respectivement les fichiers `classe0`, `classe1`, ..., `classe9`).

Pour manipuler ces 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. Par exemple, l'exécution de

```
1  ArrayList<ArrayList<byte[]>> dataset=Utils.loadImages("/home/toto
       /tp1/classe");
```

  chargera dans la liste `dataset.get(0)` les glyphes stockés dans `classe0`, dans la liste `dataset.get(1)` les glyphes stockés dans `classe1` et ainsi de suite.

- **Les glyphes et leurs étiquettes :** les glyphes auront besoin d'être placés dans des ensembles avec leur étiquette. La classe imbriquée `Utils.LabelledData` vous évitera d'avoir à implémenter cette association :

```
1  class LabelledData {
2    private int cls;
3    private byte[] glyph;
4
5    public LabelledData(int cls, byte[] glyph) { this.cls=cls; this.
         glyph = glyph; }
6    int getCls() { return cls; }
7    byte[] getGlyph() {return glyph; }
8  }
```

- **Visualisation des glyphes :**

Exercice 1 – *Écriture des fonctions de calcul*

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 : Rappelez la formule qui permet de calculer une distance entre deux points dans un espace à n dimensions. Écrivez une fonction qui calcul 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`.

Facile

Question 1.3 : Écrivez une fonction qui constitue une liste de glyphes étiquetés (`List<LabelledData>`) qui prendra en paramètre le jeu de données (`dataset`) et le nombre de glyphes à prendre dans chaque classe. La taille de la liste retournée sera donc de 10 fois la valeur passée et vous emploierez la méthode de votre choix pour la sélection des glyphes. Cependant, on souhaitera pour la suite pouvoir appeler plusieurs fois cette fonction pour constituer des ensembles disjoints. On pourra en programmer plusieurs pour tester différentes méthodes.
Question 1.4 : Rappelez le principe de la méthodes des k-plus-proches-voisins (KPPV ou KNN en anglais).
Question 1.5 : Implémentez une fonction qui, étant un glyphe particulier donné en paramètre, une liste de glyphes étiquetés (`List<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 `SortedMap<Float, LabelledData>`).
Question 1.6 : Écrivez une fonction qui, à partir d'une liste de voisins, détermine la classe du glyphe concerné.
Question 1.7 : Choisissez quelques glyphes dans le jeu de données en prenant soin de faire en sorte qu'il n'appartiennent pas aux données d'apprentissage et appliquez leur la classification par KPPV.

Exercice 2 – *Évaluation du biais ?*
La méthode *semble* efficace. Estimons ses taux d'erreurs.
Question 2.1 : Écrivez une fonction qui permet de constituer m ensembles d'apprentissage de taille n. On pourra réutiliser la fonction de la question 1.2. De la même manière, cette fonction nous permettra de constituer un ensemble de test.
Question 2.2 : Observons l'importance de k sur la qualité des résultats : écrivez une fonction qui évalue le taux d'erreurs d'apprentissage (disons sur la moyenne de 20 ensembles de 50 éléments) et de test (disons Écrivez une fonction qui génère `nbModels` ensembles d'apprentissages comportant `modelSize` glyphes de chaque classe, ainsi qu'un ensemble de test disjoint de taille comportant `testSize` glyphes de chaque classe et qui, pour toutes les valeurs de k de 1 à `maxK`, calcul pour chaque modèle le taux d'erreurs d'apprentissage et de tests et produit leur moyenne pour chaque valeur de k.

Le programme produira en sortie, une ligne pour chaque valeur de k comportant : la valeur de k, la moyenne des erreurs d'apprentissage, la moyenne des erreurs de test. Cette sortie sur trois colonnes pourra être envoyée dans un fichier qui sera traçable directement par *gnuplot*. Voir l'annexe.

On pourra essayer avec les valeurs $maxK = 100$, $nbModels = 10$, $modelSize = 50$ et $testSize = 50$ par exemple (à adapter en fonction des performance de calcul de la machine).

Tracez vos résultats sur un graphique.
Question 2.3 : Qu'en est-il de la taille du modèle. Écrivez un programme similaire qui, pour un k fixé, fait varier cette fois la taille du modèle (par exemple de 10 à 100.

Tracez vos résultats sur un graphique.

1 Annexe : rédaction

Vous trouverez dans les fichiers du TP un fichier `aatp-latex.tex` qui vous donne une souche de départ pour rédiger votre compte rendu en latex. Pour compiler un tel fichier et obtenir un pdf, il suffit d'utiliser la commande :

```
$ pdflatex aatp-latex.tex
```

Notez qu'il faudra compiler au moins deux fois pour que les références des figures soient correctement liées.

Le programme gnuplot permet de générer très facilement des tracés de courbes à partir de données. Soit le fichier `test.plot` contenant les données suivantes :

```
1        # This file is called   force.dat
2        # Force-Deflection data for a beam and a bar
3        # Deflection    Col-Force       Beam-Force
4        0.000              0              0
```

```
5       0.001           104             51
6       0.002           202             101
7       0.003           298             148
8       0.0031          290             149
9       0.004           289             201
10      0.0041          291             209
11      0.005           310             250
12      0.010           311             260
13      0.020           280             240
```

À l'invite de gnuplot, la commande

```
plot "force.dat" using 1:2 title 'Column', "force.dat" using 1:3 title 'Beam'
```

affiche les données sur un graphique interactif (zoom et déplacement). Il est possible de copier l'image de ce graphique dans le presse-papier, toutefois, une image vectorielle aura un meilleur rendu dans votre document et une taille moindre. Pour cela, dans gnuplot, taper les commandes :

```
set terminal postscript color
set output "mafigure.eps"
plot ...
```

puis dans le terminal, convertir le fichier eps produit au format pdf à l'aide de la commande :

```
$ epstopdf mafigure.eps
```

qui produira un fichier `mafigure.pdf` que vous pourrez inclure dans votre rapport en LaTeX comme montré dans l'exemple.