

# Polytech Paris-Sud 5e année - Big Data

T. Sanchez - C. Barras

## Projet de reconnaissance d'images - 1ère partie

Le but de ce projet, à réaliser en binôme, est de mettre en place et d'évaluer un système de reconnaissance automatique d'images en Python.

### Données

Les données sont disponibles sur le site <http://ufldl.stanford.edu/housenumbers> sous la rubrique *Downloads* → *Format 2*. Les fichiers d'entraînement (`train_32x32.mat`) et de test (`test_32x32.mat`) contiennent respectivement 73257 et 26032 images de chiffres photographiés dans la rue et leurs étiquettes associées. Les images ont une taille de  $32 \times 32$  pixels codés en RVB avec 256 niveaux d'intensité.

Pour charger les données au format `matlab`, on peut utiliser la bibliothèque `scipy` :

```
import numpy as np
from scipy.io import loadmat

train_data = loadmat('train_32x32.mat')
test_data = loadmat('test_32x32.mat')
```

La clé `X`, des dictionnaires `train_data` et `test_data`, permet d'accéder aux images contenues dans une matrice `numpy` à 4 dimensions (ligne du pixel  $\times$  colonne du pixel  $\times$  canal RVB  $\times$  index de l'image). La clé `y` donne accès au tableau des étiquettes associées à ces images. Il est possible d'afficher une image à partir d'une matrice avec la bibliothèque `matplotlib` :

```
import matplotlib.pyplot as plt

image_idx = 0
print('Label:', train_data['y'][image_idx])
plt.imshow(train_data['X'][:, :, :, image_idx])
plt.show()
```

### Travail à réaliser

Les différents algorithmes devront être entraînés avec le jeu d'entraînement et les taux d'erreurs devront être calculés pour le jeu d'entraînement et le jeu de test.

1. Développement d'une méthode permettant de nettoyer les données : il s'agit d'apporter des modifications simples et automatiques aux images afin d'améliorer les prédictions et les performances des algorithmes de classification. Libre à vous d'implémenter les modifications de votre choix, cependant il ne faudra pas retirer ou ajouter d'images au jeu de données. Vous testerez l'impact de votre méthode sur les classifieurs suivants.
2. Implémentation du classifieur à distance minimum (DMIN) : chaque classe est représentée par la moyenne des vecteurs d'apprentissage de cette classe. Ce classifieur doit être réalisée en Python en utilisant uniquement les librairies Numpy et Matplotlib. Vous calculerez la performance sur l'ensemble de développement grâce au taux d'erreur défini comme le nombre d'exemples mal classés divisé par le nombre total d'exemples.
3. Réduction de la dimension des vecteurs : l'analyse en composantes principales (ACP) permet de réduire la dimension des vecteurs d'images de 1024 ( $32 \times 32$ ) points à un vecteur de paramètres de plus petite taille (voir l'annexe pour la théorie, en pratique vous n'avez pas à l'implémenter vous-même car vous utiliserez la bibliothèque Scikit-Learn `sklearn.decomposition.PCA`). Vous testerez l'impact de la réduction de dimension sur le classifieur précédent (PCA+DMIN) en fonction de la dimension de l'ACP (par exemple sur une dizaine de valeurs représentatives).

4. Implémentation d'un classifieur de Scikit-Learn choisi parmi les Support Vector Machines (`sklearn.svm`) et les plus proches voisins (`sklearn.neighbors`). Vous tracerez la performance du classifieur en fonction de différentes configurations que vous choisirez ainsi que les matrices de confusion des meilleurs systèmes (avec ou sans ACP préalable).

Le compte-rendu aura la forme d'un rapport scientifique (4 parties : introduction, méthodes, résultats et discussion), dans lequel vous justifierez les choix d'implémentation et commenterez les résultats des classifieurs. Plusieurs éléments sont attendus :

- Les courbes d'erreur des classifieurs selon les différents paramètres testés.
- Les matrices de confusion des meilleurs paramètres pour chaque configuration (exemples : Preprocessing+PCA+SVM, Preprocessing+DMIN...).
- Un tableau résumant les temps d'exécution, ainsi que les taux d'erreur sur les jeux d'entraînement et de test.
- Les scripts commentés que utilisés pour la rédaction du rapport.
- Un script python `classif.py` affichant les résultats de votre meilleur système pour des données passées en paramètre (il doit pouvoir être lancé avec la commande `python classif.py --data new_data.mat`).

L'archive au format .zip et portant les noms du binôme devra être déposée dans l'espace « Travaux » du cours Et5BigData.

## Annexe - Analyse en composante principale (ACP)

L'analyse en composante principale est une technique qui permet de réduire la dimension de l'espace de représentation. L'idée consiste à trouver des axes de projection qui maximisent la variance des données (selon l'axe) et à n'en conserver qu'un petit nombre. De cette façon on espère conserver le maximum d'information pour un nombre réduit d'axes. En pratique, l'ACP permet de déterminer une matrice de projection permettant de passer des  $d$  coordonnées initiales à  $p$  coordonnées après projection. Ce nombre  $p$  d'axes (et donc de coordonnées à traiter ultérieurement) sera choisi en étudiant les performances du système sur les données de développement en fonction de ce nombre. L'ordre de grandeur est entre une dizaine et quelques dizaines.

Le principe de l'ACP est de calculer la matrice de covariance des données, de la diagonaliser, et de construire la matrice de projection sur les axes principaux à partir des vecteurs propres issus de la diagonalisation. En effet, les vecteurs propres sont les vecteurs directeurs des axes principaux. De plus, les valeurs propres associées à ces vecteurs propres sont d'autant plus grandes que l'axe est "principal". Il suffit donc de trier les valeurs propres et de ne garder que les premiers vecteurs propres associés pour construire la matrice de projection.

En principe l'ACP est effectuée sur l'ensemble des données (indépendamment de la classe). Une fois la matrice de projection déterminée, elle est appliquée à toutes les données avant traitement ultérieur : l'apprentissage et la reconnaissance se feront donc sur des vecteurs de dimension  $p$ .

### Implémentation de l'ACP

Étant donné un ensemble de vecteurs colonnes  $x_k, k = 1, n$  de dimension  $d$ , les vecteurs colonnes projetés  $y_k, k = 1, n$  de dimension  $p < d$  sont obtenus par l'ACP en diagonalisant la matrice de covariance des données  $x_k$ , et en projetant sur les vecteurs propres de plus grandes valeurs propres.

La matrice de covariance des données (de dimension  $d \times d$ ) se calcule par :

$$C = \frac{1}{n-1} \sum_{k=1}^n (x_k - \hat{\mu})(x_k - \hat{\mu})^t$$

La diagonalisation de cette matrice fournit une matrice de vecteurs propres  $V$  et une matrice diagonale  $L$  dont les éléments diagonaux sont les valeurs propres. Ces matrices de dimension  $d \times d$  vérifient :

$$C = VL V^{-1}$$

Le procédé de diagonalisation, lorsqu'il est possible, garantit que les vecteurs propres sont orthonormés (i.e. orthogonaux et de norme 1), ce qui se traduit du point de vue matriciel par :  $V^{-1} = V^t$ . Ceci est très important en pratique car l'inversion d'une matrice est une opération coûteuse alors que la transposition est immédiate. De plus, les valeurs propres et les vecteurs propres se correspondent, c'est-à-dire que la

colonne  $i$  de  $V$  contient les coordonnées du vecteur propres associé à la  $i^{\text{ème}}$  valeur propre dont la valeur est donnée par l'élément diagonal  $(i, i)$  de  $L$ .

La projection d'un vecteur  $x_k$  donné (de dimension  $d$ ) sur un vecteur  $y_k$  (de dimension  $p$ ) consiste donc simplement à calculer la projection de  $x_k$  sur les  $p$  premiers vecteurs propres. La projection doit théoriquement se faire sur le vecteur rapporté au centre des données ayant servi au calcul de la projection :

$$y_k = P(x_k - \hat{\mu})$$

où  $P$  est la matrice de projection de dimension  $(p \times d)$  définie comme les  $p$  premières lignes de la matrice  $V^{-1}$ . Mais en pratique, on peut ignorer le centrage des données qui est identique pour tous les vecteurs. Bien entendu, cette transformation peut ensuite être utilisée pour n'importe quel autre vecteur  $x$  pour fournir un vecteur  $y$  de dimension  $p$ . Il s'agit donc d'une représentation des vecteurs d'entrée d'un système de RF. Elle nécessite de mémoriser la matrice  $P$  ainsi que le vecteur  $\hat{\mu}$ .

L'implémentation de cette projection est directe. Python dispose d'une fonction de diagonalisation de matrice réelle symétrique, `eigh`, qui renvoie des valeurs propres réelles :

```
v, w = np.linalg.eigh(C);
```

où  $w$  est le tableau des valeurs propres,  $v$  la matrice des vecteurs propres,  $C$  la matrice de covariance qui peut être calculée avec la fonction `np.cov()` - attention à l'orientation des axes, les matrices d'images fournies devront être transposées. Les valeurs propres résultantes étant rangées en ordre croissant, il faudra simplement conserver les  $p$  derniers vecteurs propres de  $V$  pour construire  $P$ .