M. BENDELLA, H. FERREIRA CHAME, M.V. LE LANN, M. SIALA
2020- 2021

Objectifs du TP:

- Prise en main de la bibliothèque **scikit-learn** de Python, dédiée à l'apprentissage automatique,
- Premiers exemples de classification supervisée par la méthode des **k-plus proches voisins (k-nn)**,
- Evaluation de l'erreur d'un classifieur,
- Sélection de modèles.

Scikit-learn est un logiciel écrit en Python, qui nécessite l'installation préalable du langage Python et des librairies **NumPy** et **SciPy** (pour le calcul scientifique), dans des versions qui doivent vérifier certaines contraintes de compatibilité. Le plus simple est d'installer une distribution de Python complète, comme Anaconda 3, qui comprend la plupart des librairies courantes développées en Python, dont les trois citées plus haut. Le site officiel du logiciel **Scikit-learn** est :

http://scikit-learn.org/stable/index.html.

1. Jeux de données

Un certain nombre de jeux de données sont disponibles dans **scikit-learn**. Il est également possible de générer des données artificielles ou de récupérer des données externes (voir TPs suivants). Documentation relative au chargement de jeux de données :

http://scikit-learn.org/stable/datasets/

Les jeux de données comprennent un certain nombres d'attributs parmi les suivants (tous ne sont pas toujours définis) : **.data**, **.target**, **.target_names**, **.feature_names**, **.DESCR** :

.data est un tableau de dimensions **n x m** où n est le nombre d'instances, et **m** le nombre d'attributs.

.target stocke les classes (étiquettes) de chaque instance (dans le cas d'un apprentissage supervisé).

.target_names contient le nom des classes.

.feature_names contient le nom des attributs.

.DESCR est une description complète du jeu de données au format texte.

Le jeu de données MNIST

La base de données MNIST pour Modified ou Mixed National Institute of Standards and Technology, est une base de données de chiffres écrits à la main. MNIST a été développée par les précurseurs du deep learning, Y. LeCun et Y. Bengio, en 1998. Il contient des données d'écriture manuelle des chiffres de 0 à 9. Par définition, il constitue un problème de classification multi-classes à 10 classes.

Un exemple en entrée est une image de taille fixe 28 x 28.

Figure 1 : Quelques exemples de chiffres manuscrits de MNIST.

Un exemple est donc un vecteur de 28*28=784 composantes correspondant à un niveau de gris pour chacun des 784 pixels.

Exercice 1: Manipulation de la base de données

Les instructions Python suivantes permettent de charger le jeu de données MNIST
- Utiliser mnist_784:

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784')
```

- from sklearn.datasets import fetch_mldata

```
mnist = fetch_mldata('MNIST original')
```

- Ou chercher mnist.mat sur internet : par exemple à l'adresse : https://github.com/daniel-e/mnist_octave/raw/master/mnist.mat) et le mettre dans ~/scikit_learn_data/mldata

1. Exécutez les commandes suivantes et comprenez ce qu'elles réalisent (vous aurez à les réutiliser).

```
print(mnist)
print (mnist.data)
print (mnist.target)
len(mnist.data)
help(len)
print (mnist.data.shape)
print (mnist.target.shape)
mnist.data[0]
mnist.data[0][1]
mnist.data[:,1]
mnist.data[:100]
```

2. Visualisez les données :
   SciKitLearn intègre la librairie **matplotlib** qui propose de très nombreuses primitives permettant de générer des courbes et graphiques.
   La base de données minst comprend des chiffres manuscrits sous forme d'images de taille fixe 28 x 28.

Exécutez les commandes suivantes et comprenez ce qu'elles réalisent.

```
from sklearn import datasets
import matplotlib.pyplot as plt
mnist = datasets.fetch_mldata('MNIST original')
images = mnist.data.reshape((-1, 28, 28))
plt.imshow(images[0],cmap=plt.cm.gray_r,interpolation="nearest")
plt.show()
```

Affichez la classe correspondante à l'image affichée.

Au lieu d'exécuter commande par commande, créez le fichier TP1_prog1.py contenant le programme précédent et exécutez le sur le terminal d'Anaconda à l'aide de la commande suivante :
python [Le lien vers le fichier] TP1_prog1.py
[Le lien vers le fichier] → D:\TP1\

3. *Explorez d'autres jeux de données.*

La méthode des k-plus proches voisins (rappel)

L'algorithme des k-plus proches voisins (k-nn: pour k-neighrest neighbors en anglais) est un algorithme très intuitif, paramétrable et souvent performant pour traiter un problème de classification.

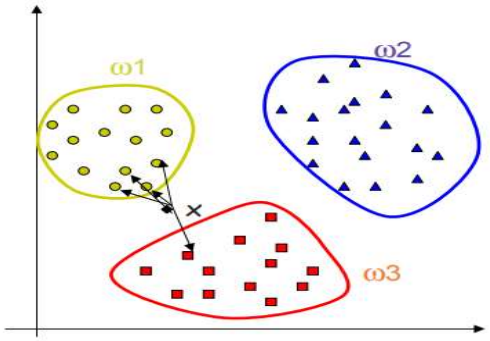
Le k-nn nécessite :

- Un entier k
- Une base d'apprentissage
- Une métrique pour la proximité

Le principe de l'algorithme est le suivant :

Pour un nouvel exemple non étiqueté **x**, trouvrer les **k** "plus proches exemples étiquetés (voisins) de la base d'apprentissage. La classe associée à **x** est la classe qui apparaît le plus souvent (majoritaire).

Dans l'exemple suivant, nous avons **3** classes **(ω1, ω2, ω3)** et le but est de trouver la valeur de la classe de l'exemple inconnu **x**. Nous prenons la distance Euclidienne comme métrique de proximité et **k=5** voisins.



Parmi les **5** plus proches voisins, **4** appartiennent à **ω1** et **1** appartient à **ω3**, donc **x** est affecté à **ω1**, la classe majoritaire.

Informations :

Nous utiliserons la bibliothèque **sklearn.neighbors** pour exécuter un exemple de cet algorithme de classification.

La commande **train_test_split** , implémentée dans **model_selection** de **sklearn** permet de diviser le jeu de données en, deux : un propre à l'apprentissage et un pour le test :

xtrain, xtest, ytrain, ytest = train_test_split(data, target, train_size=pourcentage_de_données**)**

Exemple de pourcentage_de_données = 0.7 (70%)

L'algorithme k-nn est implémenté dans un package appelé **neighbors**.
Voici les différentes commandes principales :

- La ligne **clf = neighbors.KNeighborsClassifier (n_neighbors**) crée un objet de type classifieur basé sur les **n_neighbors** plus proches voisins,
- L'instruction **clf.fit(X, y)** utilise les données pour définir le classifieur (apprentissage),
- La commande **clf.predict()** est utilisée pour classer de nouveaux exemples,
- La commande **clf.predict_proba()** permet d'estimer la probabilité de la classification proposée,
- La commande **clf.score(xtest, ytest)** calcule le score global du classifieur sur un jeu de données.

Exercice 2: La méthode des k-nn

Ouvrez le navigateur anaconda spider et écrivez un programme TP1_prog2.py qui permet de :

- Charger le jeu de données mnist,
- Prendre un échantillon de données appelé **data** avec une taille de 5000 exemples à l'aide de la fonction **np.random.randint**(70000, size=5000).
- Diviser la base de données à 80% pour l'apprentissage (training) et à 20% pour les tests,
- Entrainer un classifieur k-nn avec k = 10 sur le jeu de données chargé.
- Afficher la classe de l'image 4 et sa classe prédite.
- Afficher le score sur l'échantillon de test
- Quel est le taux d'erreur sur vos données d'apprentissage ? Est-ce normal ?
- Faire varier le nombre de voisins (k) de 2 jusqu'à 15 et afficher le score. Quel est le k optimal ?
  - Utilisez une boucle
  - Utilisez la function KFold(len(X),n_folds=10,shuffle=True) de la classe *class* sklearn.model_selection
- Faites varier le pourcentage des échantillons (training et test) et affichez le score. Quel est le pourcentage remarquable ?

- Faites varier la taille de l'échantillon training et affichez la précision. Qu'est-ce que vous remarquez
- Faites varier les types de distances (p). Quelle est la meilleure distance ?
- Fixez n_job à 1 puis à -1 et calculez le temps de chacun.
- A votre avis, quels sont les avantages et les inconvénients des k-nn : optimalité ? temps de calcul ? passage à l'échelle ?

Définitions complémentaires issues du domaine médical:

On définit principalement les:

- VP (vrais positifs) représente le nombre d'individus malades avec un test positif,
- FP (faux positifs) représente le nombre d'individus non malades avec un test positif,
- FN (faux négatifs) représente le nombre d'individus malades avec un test négatif,
- VN (vrais négatifs) représente le nombre d'individus non malades avec un test négatif.
-

Deux mesures permettent de juger d'un bon classifieur :

1. SENSIBILITE : « **true positive rate, the recall**, or probability of detection » ➔ La sensibilité, ou la probabilité que le test soit positif si la maladie est présente, se mesure chez les malades seulement (VP+FN) : proportion de positifs correctement identifiés VP/(VP+FN)

2. SPECIFICITE: true negative rate: une mesure de la sensibilité s'accompagne toujours d'une mesure de la spécificité. Cette dernière se mesure chez les non-malades seulement. Ainsi, la spécificité, ou la probabilité d'obtenir un test négatif chez les non-malades, est donné par : VN/(VN+FP)