Objectifs du TP :
- initiation à Python,
- 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,
- évaluation 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` [1] et `SciPy` [2] (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 `http://www.python-course.eu/index.php` contient un excellent tutoriel pour apprendre Python (version 2 ou version 3).
Le site officiel du logiciel `Scikit-learn` est : `http://scikit-learn.org/stable/index.html`
La documentation en ligne est complète, et devra être consultée chaque fois que nécessaire :

`http://scikit-learn.org/stable/documentation.html`.

Des tutoriaux sont disponibles à l'adresse suivante :

`http://scikit-learn.org/stable/tutorial/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 disponibles dans `scikit-learn` sont : `iris`, `boston`, `diabetes`, `digits`, `linnerud`, `wine`, `breast_cancer`, `sample_images`, `20newsgroups`.
Les jeux de données comprennent un certain nombres d'attributs parmi (tous ne sont pas toujours définis) : `data`, `target`, `target_names`, `feature_names`, `DESCR` :

.data est un tableau de dimensions $n \times m$ où n est le nombre d'instances, et m le nombre d'attributs

.target stocke les classes (étiquettes) de chaque instance (dans le cas 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.

1.1 Le jeu de données Iris (Fisher, 1936)

`Iris` est un ensemble (jeu) de données introduit en 1936 par R. Fisher comme un exemple d'application de l'*analyse discriminante linéaire*. Cet ensemble contient 150 exemples de critères observés sur 3 espèces différentes d'iris de Gaspésie (Setosa, Versicolor, Verginica). Chaque exemple est composé de quatre attributs (longueur et largeur des sépales en cm, longueur et largeur des pétales en cm) et d'une classe (l'espèce).

1. `http://www.numpy.org/`
2. `http://www.scipy.org/`
3. `https://www.continuum.io/`

Lancez Python dans un terminal. Les instructions Python suivantes [4] permettent de charger le jeu de données `Iris` :

```
from sklearn.datasets import load_iris # les données iris sont chargées
irisData=load_iris()
```

Pour afficher les attributs des données :

```
print(irisData.data)
print(irisData.target)
print(irisData.target_names)
print(irisData.feature_names)
print(irisData.DESCR)
```

Exercice :

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

   ```
   len(irisData.data)
   help(len) # pour quitter l'aide, tapez 'q' (pour quit)
   irisData.target_names[0]
   irisData.target_names[2]
   irisData.target_names[-1]
   irisData.target_names[len(irisData.target_names)-1]
   irisData.data.shape
   irisData.data[0]
   irisData.data[0][1]
   irisData.data[:,1]
   ```

2. Explorez d'autres jeux de données.

1.2 Visualiser les données

`SciKitLearn` intègre la librairie `matplotlib` (`http://matplotlib.org/`) qui propose de très nombreuses primitives permettant de faire des dessins, et la librairie `pylab`, qui intègre les librairies `NumPy`, `SciPy` et `Matplotlib`.

Exercice :

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

   ```
   import pylab as pl # permet de remplacer le nom "pylab" par "pl"
   X=irisData.data
   Y=irisData.target
   x = 0
   y = 1
   pl.scatter(X[:, x], X[:, y],c=Y) # les fonctions
   définies dans une librairie doivent être préfixées par son nom
   pl.show()
   help(pl.scatter)
   pl.xlabel(irisData.feature_names[x])
   pl.ylabel(irisData.feature_names[y])
   pl.scatter(X[:, x], X[:, y],c=Y)
   pl.show()
   ```

2. Une autre méthode

4. pour ne pas avoir à retaper les commandes, vous pouvez les copier-coller à partir de la feuille de TP au format pdf.

```
import pylab as pl
X=irisData.data
Y=irisData.target
x = 0
y = 1
Y==0
X[Y==0]
X[Y==0][:, x]
pl.scatter(X[Y==0][:, x],X[Y==0][:,y],
   color="red",label=irisData.target_names[0])
pl.scatter(X[Y==1][:, x],X[Y==1][:, y],
   color="green",label=irisData.target_names[1])
pl.scatter(X[Y==2][:, x],X[Y==2][:, y],
   color="blue",label=irisData.target_names[2])
pl.legend()
pl.show()
```

3. Une troisième méthode. Créez le fichier `TP1prog1.py` contenant le programme suivant :

```
# -*- coding: utf-8 -*-
import pylab as pl

from sklearn.datasets import load_iris
irisData=load_iris()


X=irisData.data
Y=irisData.target

x = 0
y = 1

colors=['red','green','blue']

for i in range(3):
    pl.scatter(X[Y==i][:, x],X[Y==i][:,y],color=colors[i],\
               label=irisData.target_names[i])

pl.legend()
pl.xlabel(irisData.feature_names[x])
pl.ylabel(irisData.feature_names[y])
pl.title(u"Données Iris - dimension des sépales uniquement")
pl.show()
```

   Dans un terminal, lancez l'instruction `python TP1prog1.py`.

Exercice Les données `iris` sont décrites par 4 attributs. Il y a 6 manières d'en extraire 2. En modifiant le programme ci-dessus, déterminez visuellement le couple d'attributs qui semble le mieux à même de discriminer les 3 classes d'iris.

2 Classification par les k-plus proches voisins

Documentation `Scikit learn` sur les k plus proches voisins :

`http://scikit-learn.org/stable/modules/neighbors.html`

2.1 Création et entrainement d'un classifieur

La méthode des k plus proches voisins est implémentée dans un package appelé *neighbors*. Exécutez les commandes suivantes. La ligne `clf = neighbors.KNeighborsClassifier(n_neighbors)` crée un objet du type *classifieur des n_neighbors plus proches voisins*, l'instruction `clf.fit(X, y)` utilise les données pour définir le classifieur, la commande `clf.predict` peut-être utilisée pour classer de nouveaux exemples, la commande `clf.predict_proba` permet d'estimer la probabilité de la classification proposée, et la commande `clf.score` calcule le score global du classifieur sur un jeu de données.

```
from sklearn import neighbors
nb_voisins = 15
help(neighbors.KNeighborsClassifier)
clf = neighbors.KNeighborsClassifier(nb_voisins)
help(clf.fit)
clf.fit(X, Y)
help(clf.predict)
print(clf.predict([[ 5.4,  3.2,  1.6,  0.4]]))
print(clf.predict_proba([[ 5.4,  3.2,  1.6,  0.4]]))
print(clf.score(X,Y))
Z = clf.predict(X)
print(X[Z!=Y])
```

Exercice Écrivez un programme `TP1prog2.py` qui ouvre le jeux de données `iris`, entraîne un classifieur des k plus proches voisins (avec $k = 15$) sur ce jeu de données et affiche le score sur l'échantillon d'apprentissage.

2.2 Evaluation de l'erreur du classifieur appris

Lorsque le score du classifieur appris est évalué sur l'ensemble d'apprentissage, il est en général sur-estimé (pourquoi ?) et donc, très peu fiable. La meilleure méthode pour évaluer un classifieur consiste à calculer son score sur un échantillon test, indépendant de l'échantillon d'apprentissage mais généré dans les mêmes conditions. Lorsqu'on dispose d'un seul ensemble d'exemples, il faut donc

- répartir les données en un sous-ensemble d'apprentissage et un sous-ensemble test,
- entrainer un classifieur sur l'ensemble d'apprentissage et l'évaluer sur l'ensemble test,
- puis proposer un classifieur entrainé sur l'ensemble des données.

Si les données sont peu nombreuses, comme c'est le cas pour le jeu de données `iris`, cette évaluation risque d'être pessimiste (pourquoi ?).
`Scikit learn` propose un package `cross_validation` [5] (ou `model_selection` [6] à partir de la version 0.20) qui comprend de nombreuses fonctionnalités, dont la fonction `cross_validation.train_test_split` qui partitionne aléatoirement un jeu de données en un ensemble d'apprentissage et un ensemble de test.

```
from sklearn.cross_validation import train_test_split
# ou from sklearn.model_selection import train_test_split
import random # pour pouvoir utiliser un générateur de nombres aléatoires
X_train,X_test,Y_train,Y_test=\
train_test_split(X,Y,test_size=0.3,random_state=random.seed())
help(train_test_split)
len(X_train)
len(X_test)
```

5. http://scikit-learn.org/stable/modules/cross_validation.html
6. http://scikit-learn.org/stable/model_selection.html

```
len(X_train[Y_train==0])
len(X_train[Y_train==1])
len(X_train[Y_train==2])
```

Exercice Écrivez un programme `TP1prog3.py` qui ouvre le jeux de données `iris`, répartit les données en un ensemble d'apprentissage et un ensemble test (30% pour le test), entraîne un classifieur des 15 plus proches voisins sur l'échantillon d'apprentissage et évalue son erreur sur l'échantillon test.

2.3 Sélection de modèle

Pourquoi prendre $k = 15$? Comment sélectionner une valeur de k optimale ? On utilise souvent une méthode de validation croisée pour sélectionner les hyper-paramètres d'un modèle.
Le package `cross_validation` (ou `model_selection`) propose la fonction `KFold` qui répartit un ensemble de données X en n_folds couples d'ensembles d'apprentissage et de test. Exécutez les commandes suivantes :

```
from sklearn.cross_validation import KFold
kf=KFold(20,n_folds=10,shuffle=True)
for learn,test in kf:
     print("app : ",learn," test ",test)
```

ou, selon la version,

```
from sklearn.model_selection import KFold
kf=KFold(n_splits=10,shuffle=True)
X=[i for i in range(20)]
print(X)
for learn,test in kf.split(X):
     print("app : ",learn," test ",test)
```

Recommencez avec le paramétrage `shuffle=False`.
Étudiez et exécutez le programme ci-dessous (`TP1prog4.py`), en utilisant l'aide de `Scikit learn`.

```
# -*- coding: utf-8 -*-

import random
from sklearn.datasets import load_iris
irisData=load_iris()


X=irisData.data
Y=irisData.target

from sklearn import neighbors

from sklearn.cross_validation import KFold
# from sklearn.model_selection import KFold

kf=KFold(len(X),n_folds=10,shuffle=True)
#kf=KFold(n_splits=10,shuffle=True)

scores=[]
```

```
for k in range(1,30):
    score=0
    clf = neighbors.KNeighborsClassifier(k)
    for learn,test in kf:
    # for learn,test in kf.split(X):
        X_train=X[learn]
        Y_train=Y[learn]
        clf.fit(X_train, Y_train)
        X_test=X[test]
        Y_test=Y[test]
        score = score + clf.score(X_test,Y_test)
    scores.append(score)

print(["{:4.2f}".format(s) for s in scores])

print("meilleure valeur pour k : ",scores.index(max(scores))+1)
```

Que se passe t-il si vous remplacez la 9-ième ligne par `kf=KFold(len(X),n_folds=3,shuffle=False)` (ou `kf=KFold(n_splits=3,shuffle=False)`) ?

3 Application au jeu de données Digits

Le jeu de données `digits` [7] comprend 1797 images de résolution 8x8, représentant un chiffre manuscrit. Le champ `image` est un tableau 8×8 d'entiers représentant des niveaux de gris et variant de 0 (blanc) à 16 (noir). Le champ `data` est un vecteur de dimension 64 comprenant la même information. Le champ `target` est un chiffre compris entre 0 et 9.

```
from sklearn.datasets import load_digits
digits=load_digits()
digits.data[0]
digits.images[0]
digits.data[0].reshape(8,8)
digits.target[0]
```

Il est possible de visualiser chaque image au moyen du code suivant :

```
import pylab as pl
pl.gray()
pl.matshow(digits.images[0])
pl.show()
```

Pour compter les exemples d'une classe

```
print(len(Y[Y==0]))
```

Exercice Écrivez un programme `TP1.prog5.py` qui
- ouvre le jeu de données `digits`,
- répartit les données en un ensemble d'apprentissage et un ensemble test (30% pour le test),

7. voir `http://scikit-learn.org/stable/auto_examples/datasets/plot_digits_last_image.html`.

— choisit par validation croisée sur l'ensemble d'apprentissage un nombre de voisins k_{opt} optimal,
— entraîne le classifieur des k_{opt} plus proches voisins sur l'échantillon d'apprentissage,
— évalue son erreur sur l'échantillon test,
— affiche quelques chiffres parmi ceux qui sont mal classés.