TP4 : K-nearest neighbours & cython

A. Useful iterable objects: *range*, *enumerate* , *zip*, dictionaries...

When using loops, *iterators* can take various forms. When using a simple number, *range()* is fine as in:

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

%matplotlib notebook
```

In [ ]:
```
for i in range(5):
    print(i)
```

range() is **iterable**, not building an explicit list in the memory. You may also use simple lists as in:

In [ ]:
```
words = ['cat', 'window', 'defenestrate']
for w in words:
    print(w, len(w))
```

If you want to use both the element of a list and its index inthis list, **enumerate()** might be useful. Go to https://docs.python.org/3.6/library/functions.html#enumerate (https://docs.python.org/3.6/library/functions.html#enumerate) for more information.

In [ ]:
```
for i, v in enumerate(['tic', 'tac', 'toe']):
    print(i, v)
```

When looping through **dictionaries**, the key and corresponding value can be retrieved at the same time using the items() method.

In [ ]:
```
knights = {'gallahad': 'the pure', 'robin': 'the brave'}
for k, v in knights.items():
    print(k, v)
```

To loop over two or more sequences at the same time, the entries can be paired with the **zip()** function.

In [ ]:
```
questions = ['name', 'quest', 'favorite color']
answers = ['lancelot', 'the holy grail', 'blue']
for q, a in zip(questions, answers):
    print('What is your {0}?  It is {1}.'.format(q, a))
```

Note in passing how the final string has been obtained using *.format(q,a)*.

Go to https://docs.python.org/3.6/tutorial/datastructures.html#tut-loopidioms (https://docs.python.org/3.6/tutorial/datastructures.html#tut-loopidioms) for more details on loops.

B. Classification using the K-nearest neighbours algorithm

This practical is a first introduction to classification using the most intuitive non parametric method : K nearest neighbours. The principle is the following. A set of labelled observations which can belong to 2 classes is given as a learning set. Then new observations must be classified as either of class 1 or 2. The principle of K-NN is to label a new observation from the most frequent label among its K nearest neighbours.

1. Using synthetic data

Load the training and test data sets synth_train.txt and synth_test.txt. Targets belong to {1,2} and entries belong to $\mathbb{R}^2$. We have 100 training data samples and 200 test samples.

```
* the 1st column contains the label of the class the sample,
* columns 2 & 3 contain the coordinates of each sample in 2D.
```

In [ ]:
```
# load the training set
train = np.loadtxt('synth_train.txt')  #...,delimiter=',') if there ar
class_train = train[:,0]
x_train = train[:,1:]
N_train = train.shape[0]
```

In [ ]:
```
# load the test set
test = np.loadtxt('synth_test.txt')
class_test_1 = test[test[:,0]==1]
class_test_2 = test[test[:,0]==2]
x_test = test[:,1:]
N_test = test.shape[0]
```

TO DO:

```
a. Display the training set and distinguish the two classes. T
he function scatter can be useful.
b. Implement the K-nearest neighbours algorithm for classifica
tion.
c. Compute the error rate on the training set and the test set
for K = 1,...,30.
d. Comment on your results. Which value of K seems optimal ?
```

Indication 1: Python functions to sort, norm and replicate (see `tile()`) matrices may be useful to compute and sort distances to a given data point while limiting the number of loops 'for'.

Indication 2: if class_pred contains the predicted labels for N_test samples, one can display classification results by using:

```
In [ ]: a. Display the training set and distinguish the two classes. The funct
```

```
In [ ]: plt.figure()
        plt.plot(train[class_train==1,1],train[class_train==1,2],'ko')  # clas
        plt.plot(train[class_train==2,1],train[class_train==2,2],'mo')  # clas

        plt.show()

        # plt.plot(test[class_pred==1,1],test[class_pred==1,2],'rs')  # red = 
        # plt.plot(test[class_pred==2,1],test[class_pred==1,2],'gs')  # green 
```

where `class_pred` contains the class predicted by your K-NN algorithm (TO DO !).

```
In [ ]: # b. Implement the K-nearest neighbours algorithm for classification.
```

```
In [ ]: # c. Compute the error rate on the training set and the test set for K
```

```
In [ ]: # d. Comment on your results. Which value of K seems optimal ?
```

2. Application to a real data set : Breast cancer Wisconsin.

TO DO: Apply the K-NN classifier to the provided set wdbc12.data.txt.

Information about the data is provided in `wdbc12.names.txt`.

3. Making K-NN faster using `cython`

Cython is a package that permits to interface C code with Python easily. It may be useful to make your code faster for a small coding effort, in particular when using long loops.

See http://docs.cython.org/en/latest/src/tutorial/cython_tutorial.html (http://docs.cython.org/en/latest/src/tutorial/cython_tutorial.html) for more details.

Indication : an example is given in the notebook **`cython_example.ipy`**.

TO DO (subsidiary): Here you may use it to implement a faster version of K-NN.

```
In [ ]:
```

```
In [ ]:
```