**Classification with Python**

*Jean Auriol, jean.auriol@ucalgary.ca*

**Introduction:**

In general a learning problem considers a set of n samples of data and tries to predict properties of a second set of unknown data. We only consider here **supervised learning problems** in which the data comes with additional attributes that we want to predict. In particular, we focus in this course on **classification problems**. Samples belong to two or more classes and we want to learn from already labeled data how to predict the class of unlabeled data.

Machine learning is about learning some properties of a data set and then testing those properties against another data set. In this document we will use the following terminology

* **Original data set.**  This set corresponds to the data which are available for learning. It will be split into two data subsets: the **training set** (in which we learn some properties) and the **testing set**, on which we test the learned properties and adjust some tuning parameters.
* **Unknown data set.**  This set of data that we want to classify.

In the following exercises, you will need some specific libraries. These libraries must be installed first. In particular, we will need the sklearn library. As a reminder, to install libraries, open *AnacondaPrompt* and type:

**pip install sklearn**

Next, open up Jupyter Notebook and navigate to this folder, **before you make a new notebook**. Now, create a new Python 3 notebook. First, we need to load the necessary libraries:

**import numpy as np  
import matplotlib.pyplot as plt**

**import matplotlib.markers**

**from sklearn import datasets, linear\_model, neighbors**

**from sklearn.metrics import mean\_squared\_error, r2\_score, explained\_variance\_score**

**from sklearn.neighbors import NearestNeighbors**

The library **scikit-learn** comes with a few standard datasets (for instance the iris and digits datasets) that we will use later. We import them with the command **from sklearn import datasets**. A dataset is a dictionary-like object that holds all the data and some metadata about the data. This data is stored in the .data member, which is a (n\_samples, n\_features) array. One or more response variables are stored in the .target member. Let us consider for instance the **digits dataset.**

digits = datasets.load\_digits()

Each data point of this dataset is a 8x8 image of a digit. As there exist 10 different digits, there are 10 classes in this dataset. More precisely, the characteristics of this dataset are given by the following table.

|  |  |
| --- | --- |
| Classes | 10 |
| Samples per class | ~180 |
| Samples total | 1797 |
| Dimensionality | 64 |
| Features | integers 0-16 |

In the case of this dataset, digits.data gives access to the features that can be used to classify the digits samples:

print(digits.data)

[[ 0. 0. 5. ... 0. 0. 0.]

[ 0. 0. 0. ... 10. 0. 0.]

[ 0. 0. 0. ... 16. 9. 0.]

...

[ 0. 0. 1. ... 6. 0. 0.]

[ 0. 0. 2. ... 12. 0. 0.]

[ 0. 0. 10. ... 12. 1. 0.]]

and digits.target gives the ground truth for the digit dataset, that is the number corresponding to each digit image that we are trying to learn:

digits.target

array([0, 1, 2, ..., 8, 9, 8])

The data is **always**  a 2D array shape (n\_samples, n\_features), although the original data may have had a different shape. In the case of the digits, each original sample is an image of shape (8,8) and can be accessed using

digits.images[0]

array([[ 0., 0., 5., 13., 9., 1., 0., 0.],

[ 0., 0., 13., 15., 10., 15., 5., 0.],

[ 0., 3., 15., 2., 0., 11., 8., 0.],

[ 0., 4., 12., 0., 0., 8., 8., 0.],

[ 0., 5., 8., 0., 0., 9., 8., 0.],

[ 0., 4., 11., 0., 1., 12., 7., 0.],

[ 0., 2., 14., 5., 10., 12., 0., 0.],

[ 0., 0., 6., 13., 10., 0., 0., 0.]])

If you plot the corresponding image, a zero will appear. This can be seen writing

plt.gray()

plt.matshow(digits.images[0])

plt.show()

You can check it reading the corresponding target (digits.target[0]).

**Iris dataset:** Using the command iris=datasets.load\_iris(), load the iris data set. There are three classes in this dataset which are *setosa, versicolor* and *virginica.*

Print these dataset using the command **list(iris.target\_names).**

Each component of the data set has four characteristics (in the form of real numbers). What are the characteristics of the 10th element of the dataset?

Which plant is the 10th element of the dataset?

Now, you know how to play with a dataset!

**Exercise 1: Nearest Neighbours on a temperature problem**

The purpose of this exercise is to learn how to use the Nearest Neighbours algorithm on a toy problem.

We consider a data set that corresponds to the mean temperatures (in °C) and the total precipitations (in mm) for different days of May 2018 and May 2019 in Calgary.

More precisely, we have the following tables

May 2018

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Day | 05/01 | 05/02 | 05/03 | 05/04 | 05/05 | 05/06 | 05/07 | 05/08 | 05/09 | 05/10 |
| Temp | 9.4 | 9.9 | 13.1 | 14.5 | 13.6 | 12.8 | 16.7 | 17.7 | 9.4 | 6.3 |
| Precip | 0.4 | 0 | 0 | 0 | 0 | 0 | 0.4 | 1 | 1.7 | 2.6 |

May 2019

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Day | 05/01 | 05/02 | 05/03 | 05/04 | 05/05 | 05/06 | 05/07 | 05/08 | 05/09 | 05/10 |
| Temp | 2.3 | 5.2 | 4.1 | -1.4 | -0.3 | 4.4 | 5.8 | 7.3 | 9.6 | 13.2 |
| Precip | 0 | 4.6 | 0 | 5.5 | 1.4 | 0 | 6 | 2.2 | 0.3 | 0.6 |

1. Using the command np.array, create the data array X\_data and the target array X\_target that correspond to the given data set. The data array should have 2 columns (one for the temperature, and one for the precipitations) and 20 lines. The components of the target array X\_target should be either 0 (May 2018) or 1 (May 2019). The target array should have 20 lines.
2. Create the 10x1 arrays *temp\_2018; temp\_2019, precip\_2018* and *precip\_2019* that respectively correspond to the temperature for the 10 considered days of 2018, of 2019 and for the precipitations in 2018 and 2019.

Using the following command lines, plot the different points

plt.figure(figsize=(9,5))

plt.grid(linewidth=0.25)

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),np.arange(y\_min, y\_max, h))

plt.scatter(temp\_2018,precip\_2018,color='red',marker='v', label='May 2018')

plt.scatter(temp\_2019,precip\_2019,color='blue',marker='s', label='May 2019')

plt.xlabel("Temperature (°C)")

plt.ylabel("Precipitation (mm)")

plt.title("")

plt.legend(loc='upper right')

plt.show()

1. We consider a day for which the temperature was of 15°C and for which the precipitations were of 3mm. Just by graphic reading, do you think this day belongs to 2019 or 2018?

What about a second day for which the temperature was of 6.5°C and the precipitations were of 1mm?

To solve the classification problem, we want to plot the decision boundaries for each class.

To do so, copy and paste the following code

n\_neighbors =5 # Nb of neighbours

h = .02 # step size in the mesh

temp=X\_data[0:20:1,0]

precip=X\_data[0:20:1,1]

# Create color maps

cmap\_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])

cmap\_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

# We create an instance of Neighbours Classifier and fit the data.

clf = neighbors.KNeighborsClassifier(n\_neighbors, weights='uniform')

clf.fit(X\_data, X\_target)

# Plot the decision boundary. For that, we will assign a color to each

# point in the mesh [x\_min, x\_max]x[y\_min, y\_max].

x\_min, x\_max = temp.min() - 1, temp.max() + 1

y\_min, y\_max = precip.min() - 1, precip.max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),

np.arange(y\_min, y\_max, h))

Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])

# Put the result into a color plot

Z = Z.reshape(xx.shape)

plt.figure()

plt.pcolormesh(xx, yy, Z, cmap=cmap\_light)

# Plot also the training points

plt.scatter(temp, precip, c=X\_target, cmap=cmap\_bold,

edgecolor='k', s=20)

plt.xlim(xx.min(), xx.max())

plt.ylim(yy.min(), yy.max())

plt.title("3-Class classification (k = %i, weights = '%s')"

% (n\_neighbors, weights))

plt.show()

clf.kneighbors([[6.5,1]])

The most important lines are the three lines

clf = neighbors.KNeighborsClassifier(n\_neighbors, weights='uniform')

clf.fit(X\_data, X\_target)

clf.predict(np.c\_[xx.ravel(), yy.ravel()])

**The method .fit(X\_data, X\_target)** is used fit the model using X\_data as training data and X\_target as target values.

**The method .predict** predicts the class labels for the provided data.

**The method .kneighbors** finds the K-neighbors of a point

1. Plot the decision boundaries using the distance weight.
2. For each weight change the number of neighbours used. What do you think about it? What would be for you the optimal number? Do you think we have enough training points?
3. We now consider a new data set that corresponds to the mean temperatures (in °C) and the total precipitations (in mm) for some other days of May 2018 and May 2019 in Calgary. Using the function score, returns the mean accuracy on the given testing set for k=3. Try again for k=5. Plot the accuracy as a function of k. Does the choice of the weights make a difference? Comment the obtained results.

May 2018

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Day | 05/11 | 05/12 | 05/13 | 05/14 | 05/15 | 05/16 | 05/17 | 05/18 | 05/19 | 05/20 |
| Temp | 9.9 | 12.8 | 15.4 | 18.4 | 19.9 | 19 | 9.3 | 10.1 | 9.9 | 14.3 |
| Precip | 0.2 | 0 | 0 | 0 | 0 | 0 | 10.8 | 1.8 | 0 | 0 |

May 2019

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Day | 05/11 | 05/12 | 05/13 | 05/14 | 05/15 | 05/16 | 05/17 | 05/18 | 05/19 | 05/20 |
| Temp | 12.2 | 16.5 | 12.4 | 14.3 | 12.8 | 7 | 5.1 | 4.4 | 6.7 | 4.1 |
| Precip | 0 | 0 | 0.1 | 0.4 | 0.4 | 1.6 | 10 | 2.8 | 0 | 0 |

**Exercise 2:** **Nearest Neighbours for the digit dataset**

In this exercise, we consider the digit dataset. This dataset has 1797 samples. We will use 500 samples as the training dataset, 500 samples as the testing dataset and 797 samples as the unknown dataset.

1-We choose 50 neighbour. What is the accuracy of the classification? We will consider the different weights (uniform and distance).

2-Compare the performance of the different algorithms: “ball\_tree”, kd\_tree”, “brute” for 50 and 150 neighbors (accuracy and time of execution). What do you think of the obtained results? To print the time of execution of a script, you can use the following code.

import time

start = time.time()

#Your code

end = time.time()

print(end - start)

3-We choose either k=50, k=100 or k=150. Which value of k seems to be the best? Test your classifier on the unknown dataset for each value of k. Comment.

4-Using your training set, plot the accuracy as a function of k. Comment.

**Exercise 3:** **Nearest Neighbours for the iris dataset**

In this exercise, we consider the iris dataset. This dataset has 150 samples. The first 50 samples correspond to *setosa,* the next 50 to *versicolor* and the last 50 to *virginica.* We will use 50 samples as the training dataset, 50 samples as the testing dataset and 50 samples as the unknown dataset. Each sample of the dataset has four components. However, we will consider in this exercise that **only the two first components are relevant**.

1-Create the training dataset that contains at least 16 samples that correspond to *setosa,* 16 samples that correspond to *versicolor* and 16 samples that correspond to *virginica.* Create the testing and unknown datasets mixing the types of iris.

2- For each type of weights, plot the decision domain using the given training set.

3-Using you testing dataset and the “distance weights”, what is the best value for k? For this value of k, test your classifier against the unknown dataset. Now test it with k=10 and k=20. Comment.

4-We now consider that your unknown dataset is your new training set, that your former testing set is your new unknown set and that your former training set is your new testing set. Answer again to question 2 and 3. Comment.

5-Repeat question 2 and 3 but now considering the three first components of your dataset. What are the effects on your classifier?

**Exercise 4:** **Decision trees for the iris dataset**

For this exercise you will need to import the following libraries

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier

from sklearn import datasets

from IPython.display import Image

from sklearn import tree

import graphviz

import pydotplus

The library graphviz is required if you want to draw the tree. However to use it you need to check that the path of the graphviz executable is known. If this is not the case, go to Control Panel\System and Security\System, Advanced system settings, environment variables, path and add the corresponding path. In my case the path was

C:\Users\jean.auriol\AppData\Local\Continuum\anaconda3\Library\bin\graphviz

Once the classification has been done, you can plot the tree using the following code

# Create DOT data

dot\_data = tree.export\_graphviz(clf, out\_file=None,

feature\_names=**dataset\_name**.feature\_names,

class\_names= **dataset\_name**.target\_names)

# Draw graph

graph = pydotplus.graph\_from\_dot\_data(dot\_data)

# Show graph

Image(graph.create\_png()

1. For the Iris data set, create an original data set of 75 elements and a training set of 75 elements (no unknown set here). Only consider the two first components of the data. Make sure that both sets contains similar proportions of *setosa, versicolor* and *virginica*. Plot the decision tree corresponding to the original dataset. Comment. Test this tree on the unknown set (command score). What is its accuracy? Comment. Compare with the K-means algorithm. The function to create a decision tree classifier is clf = DecisionTreeClassifier ()
2. Play with the option of the DecisionTreeClassifier. Criterion is “Gini” by default but you can change it for entropy. The maximum depth of the tree can be chosen by the command “max\_depth”. The command “min\_samples\_split” allows you to select the number of samples required to split an internal node (have a look to the list of possible options as min\_impurity\_decrease)... Compute the corresponding accuracy in each situation. Comment. Compute the execution time when testing against the training data.
3. Repeat the analysis but now considering three sets of 50 elements. The pruning should be done on the testing data. Also consider all the four components of the data.
4. If you have time, try to plot the decision tree for the digit dataset.

**Exercise 5:** **Large Dataset**

For this last exercise, we will use one the dataset available in the folder Dataset. You can choose the one you want and if you have our own dataset that you would like to analyse, feel free to use it.

These datasets take the form of .xslx files. To use them with Python, you will need the following lines of code.

import pandas as pd

df = pd.read\_excel(r'*your\_file\_path*')

#print the column names

print(df.columns)

#get the values for a given column

values = df['class'].values

print(values)

You should be aware that the K-Neighbours approach requires to compute distances. This notion isn’t well-defined for strings. Thus, you should convert strings into double before analyzing your data. Moreover, that as we are dealing with classification algorithms, the target should be a label. For instance if your target is the grade obtained at the final (which is between 0 and 100), you may have to regroup the grades into larger labels (as pass/fail or A/B/C/D/E).

For the set of data you have, divide it into a training set, a testing set and an unknown set. Test different classification algorithms and compute the accuracy for each algorithm.