UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

Machine Learning: Decision Trees and Neural Nets (~4h)

The goal of this practical session is to get a practical understanding of decision trees, neural networks, and their learning algorithms. The practical session will be using Scikit-learn and, optionally in the last exercises, PyTorch or Tensor Flow.

Scikit-learn is a free software machine learning library for the Python programming language. It is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. A nice textbook on Python for Data Science is available on Github. PyTorch and TensorFlow are open source software library for numerical computation using data flow graphs. They also use the Python programming language.

This session is inspired by the one from E. Fromont from previous years which is itself insipride by a practical sessions given in French at Aix-Marseille-University.

To **install** Scikit-learn and Tensor Flow on your own computer, I suggest to start by installing ANACONDA (version 3.*) which includes NumPy, SciPy and Scikit-learn. To visualize some models (in particular decision trees) in sklearn you will also need to install Graphviz. You can install PyTorch (or TensorFlow) by following their website's instructions. Installation under Windows might be more difficult (try this with conda).

You do not need a very strong background in Python to do this practical session. However, if you want to learn the basics, you can follow this tutorial or this one.
Note that if you do not want to use the standard Python interpreter from your terminal, you can use `ptpython` (install it using `pip install ptpython`) that will provide better completion, colors, edition help, … Another interesting alternative is the "Jupyter Notebook". It is a web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text. Jupyter is already installed in Anaconda, just type `jupyter notebook` in your dev directory to use it.

To be graded (part of the practical grade), you are expected to upload on Claroline connect (course "ML and PR M1") no more than two days after the session, an archive (`.zip` format) with a report (4-5 pages) that records for each exercise your results for the lines that are numbered. It is more important to write down what the command do than what the actual result is (in particular, do not paste answers that are very long, such as entire datasets). This archive should also contain a number of Python programs explicitly asked in the text of the session.

The original format and the description of most of the datasets that are used during this lab session can be found on the UCI web page. We will use in particular, the ones provided in Scikit-learn dataset respository: **iris, boston, diabetes, digits, linnerud, sample images, 20newsgroups.** The datasets use a common set of attributes (there are not all allways defined): **data, target, target names, feature names, DESCR.**

- `data` is n*m dimensional array where n is the number of instances and m the number of attributes;
- `target` stores the class label of each instance (in a supervised setting)
- `target_names` stores the name of the classes
- `feature_names` stores the name of the attributes
- `DESCR` is a complete description of the the dataset in text format.

On some computers, the datasets (in `.csv`) are available in "~/ananconda/….../lib/python3.6/site-packages/sklearn/datasets/". If you are curious about how the dataset attributes (data, target, target_names, feature_names, DESCR) mentioned before are loaded, you can look at the file "base.py" in this directory.

Exercise 1: Warm up, visualizing IRIS (Fisher, 1936): (40 minutes)

This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. The data set contains 3 classes (`Iris-Setosa, Iris-Versicolour, Iris-Virginica`) of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other. The attributes are the `sepal_length` (in cm), the `sepal_width` (in cm), the `petal_length` (in cm) and the `petal_width` (in cm).

Start `python` (or `ptpython` or `jupyter notebook`) in a terminal. The following commands help you to load the Iris dataset.

```
from sklearn.datasets import load_iris #data are loaded
irisData = load_iris()
```

To print and understand the data attributes, you can type:

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

Execute the following commands and understand what they do (write down the answer in your report for the lines that are numbered); **you can copy and paste the commands in your terminal**):

```
print(len(irisData.data))                                    # [1]
help(len) # to quit the "help" press 'q'
print(irisData.target_names[0])                              # [2]
print(irisData.target_names[2])
print(irisData.target_names[-1])
print(irisData.target_names[-1+len(irisData.target_names)])  # [3]
print(irisData.data.shape)                                   # [4]
print(irisData.data.shape[0])                                # [5]
print(irisData.data[0])                                      # [6]
print(irisData.data[0,1]) # same as irisData.data[0][1] but preferred in numpy
print(irisData.data[1:3,1])
print(irisData.data[:,1])
```

The `matplotlib` library and in particular its module `matplotlib.pyplot` can be used to visualize your data. "Pyplot provides the state-machine interface to the underlying plotting library in matplotlib. This means that figures and axes are implicitly and automatically created to achieve the desired plot. For example, calling plot from pyplot will automatically create the necessary figure and axes to achieve the desired plot. Setting a title will then automatically set that title to the current axes object".

Execute the following commands and understand (and write in your report) what they do:

```
from matplotlib import pyplot as plt # replace the name "pyplot" by "plt"
X = irisData.data
Y = irisData.target
x = 0
y = 1

plt.scatter(X[:,x], X[:,y], c=Y)                             # [7]
plt.show()                                                   # [8]
help(plt.scatter)
plt.xlabel(irisData.feature_names[x])
plt.ylabel(irisData.feature_names[y])                        # [9]
plt.scatter(X[:, x], X[:, y], c=Y)
plt.show()
```

The following sequence of commands allows you to obtain a slightly more precise answer. Execute the commands and understand (and write in your report for the ones numbered) what they do:

```
print(Y==0)                                                    # [10]
print(X[Y==0])                                                 # [11]
print(X[Y==0][:, x])
plt.scatter(X[Y==0][:, x], X[Y==0][:, y],                      # [12]
            c="red", label=irisData.target_names[0])
plt.scatter(X[Y==1][:, x], X[Y==1][:, y],
            c="green", label=irisData.target_names[1])
plt.scatter(X[Y==2][:, x], X[Y==2][:, y],
            c="blue", label=irisData.target_names[2])
plt.legend()                                                   # [13]
plt.show()
```

And finally, create a standalone file `pract1prog1.py` which contain the following program:

```
import sys
...
# ^ fill-in with imports and data loading...

colors = ["red", "green", "blue"]
for i in range(3):
    plt.scatter(X[Y==i][:, x], X[Y==i][:,y], c=colors[i], label=irisData.target_names[i])
plt.legend()
plt.xlabel(irisData.feature_names[x])
plt.ylabel(irisData.feature_names[y])
plt.title("Iris Data - size of the sepals only")
if len(sys.argv) > 1:
    plt.savefig(sys.argv[1])
else:
    plt.show()
```

In a terminal, launch the program by typing `python3 pract1prog1.py`

Exercise 2: Create your own dataset (30 minutes)

Here, you will create your own dataset. We propose three very different *alternative* ways to do that. Select the one you want. The dataset that you have created will be used in the last exercise.

You can **load** any text file using Numpy and the command `numpy.loadtxt` described in this page or you can load a `.csv` file using:

```
from numpy import genfromtxt
my_data = genfromtxt('my_file.csv', delimiter=',')
```

Alternative 1: draw your dataset

You can use a tool to draw points in a 2D plane and export and save them in a `.csv` file. https://www.librec.net/datagen.html

Alternative 2: a synthetic dataset

You could create your own n-class classification problems in sklearn using `datasets.make_classification( n_samples=25, n_features=4, n_informative=2, n_redundant=2, n_classes=2)` (from `sklearn.datasets`) (the function has many more possible parameters). The command returns an array X of shape [n_samples, n_features] which contains the generated samples and an array Y of shape [n_samples] which contains the integer labels for the class membership of each sample.

Alternative 3: a custom dataset

1. Choose the kind of preference you would like to explain, e.g.: actress/actors, food, cars, music bands, or anything you want.
2. Create a data file in a format understandable by python containing about 25 entries, each described by about 4 attributes (e.g.: calories numeric, taste {sweet, sour, bitter, salty}, vegetarian {yes, no}) and the last attribute containing your target class (e.g.: like_it {yes, no}). Provide your dataset in your archive.

Exercise 3: KNN classifier on IRIS (40 minutes)

You can find more information about the KNN (K-nearest-neighbors) classifier in scikit-learn in the corresponding page. The KNN algorithm is implemented in the `neighbors` package. In the following, the `clf = neighbors.KNeighborsClassifier(n neighbors)` creates an object "KNN classifier", `clf.fit(X, Y)` uses the data to define the classifier, the command `clf.predict` can be used to classify new examples whereas `clf.predict_proba` estimates the probability of the given classification. `clf.score` computes the global score of the classifier on a given dataset.

Execute the following commands and understand (and write in your report when they are numbered) what they do:

```
from sklearn import neighbors
nb_neighb = 15
help(neighbors.KNeighborsClassifier)
clf = neighbors.KNeighborsClassifier(nb_neighb)
help(clf.fit)

clf.fit(X, Y)                                                    # [1]
# ^ this obviously does not work if X and Y were not defined before
help(clf.predict)
print(clf.predict([[ 5.4, 3.2, 1.6, 0.4]]))                      # [2]
print(clf.predict_proba([[ 5.4, 3.2, 1.6, 0.4]]))                # [3]
print(clf.score(X,Y))                                            # [4]
Z = clf.predict(X)                                               # [5]
print(X[Z!=Y])                                                   # [6]
```

You have learned that the empirical score (ex: accuracy) given on the training set overestimates the true score of your classifier on unseen data. We thus need a set of data independent from your training data but generated in the same conditions to evaluate the classifier. We thus need to separate our data into 2 sets: the training and the test sets, train the classifier with the training set and evaluate it on the test set. However, if you have few data (as for Iris), this evaluation might be pessimistic (do you know why?). Scikit learn proposes a model selection package which allows you to split a data set into training/test (using `model_selection.train_test_split`). The metrics module provides a good number of evaluation criteria for your classifier.

```
from sklearn.model_selection import train_test_split
import random # to generate random numbers
```

Execute the commands and understand (and write in your report) what they do:

```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,            # [7]
                    test_size=0.3,random_state=random.seed())
print(X_train.shape)                                             # [8]
print(X_test.shape)
print(X_train[Y_train==0].shape)                                 # [9]
print(X_train[Y_train==1].shape)
print(X_train[Y_train==2].shape)
clf = clf.fit(X_train, Y_train)
Y_pred =clf.predict(X_test)
from sklearn.metrics import confusion_matrix                     # [10]
```

```
cm = confusion_matrix(Y_test, Y_pred)
print(cm)
```

[11] Explain in a few lines what you can see (in general) in a confusion matrix.

Why did we chose k=15? How do we chose the best value for k? In general those hyper parameters are tuned using cross validation. The model_selection package offers the function `KFold` which splits the original dataset X into n folds which are pairs of (training set, test set).

Execute the following commands:

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

[12] Try again using the parameter `shuffle=False`. What happened?

Study and execute the following program (`pract1prog2.py`):

```
...
# ^ fill-in with imports and data loading...
X = irisData.data
Y = irisData.target
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.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(scores)
print("best k:", scores.index(max(scores))+1)
```

[13] What happen if you replace the line `kf=KFold(n_splits=10, shuffle=True)` by `kf=KFold(n_splits=3, shuffle=`

Exercise 4: Decision Trees on IRIS (40 minutes)

You can find more information about Decision Tree classifiers in scikit-learn in the corresponding documentation page. The algorithm is implemented in the `tree` package. Execute the commands and understand (and write in your report) what they do:

```
from sklearn.datasets import load_iris
from sklearn import tree
iris = load_iris()

clf = tree.DecisionTreeClassifier()                    # [1]
clf = clf.fit(iris.data, iris.target)                  # [2]
print(clf.predict([iris.data[50,:]]))                  # [3]
print(clf.score(iris.data, iris.target))               # [4]
tree.export_graphviz(clf, out_file='tree.dot')         # [5]
```

If you try the command `help(tree.DecisionTreeClassifier)` you will see all the parameters of this algorithm among which:

- `max_depth`: (default none) The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

- `min_samples_split`:(default=2) The minimum number of samples required to split an internal node.
- `max_leaf_nodes`: grow a tree with "max_leaf_nodes" in best-first fashion. Best nodes are defined as relative reduction in impurity. If "None" then the algorithm allows an unlimited number of leaf nodes.

For example, you can try the command `clf = tree.DecisionTreeClassifier( criterion="entropy", max_depth=3, max_leaf_nodes=5)` Note that, in linux, you can transform the `.dot` file into a `.pdf` using the command `dot -Tpdf tree.dot -o iris.pdf`. In Windows, you can start the `graphviz` application and open the `.dot` file from there. It is also possible to integrate the graph directly into a jupyter notebook (with `graphviz.Source(dot_data)`)

- `[6]` Write a program `pract1prog3.py` which opens the IRIS dataset and trains a decision tree using the default parameters. Visualize this tree. How many leaves does it contain? Start the training phase again by gradually decreasing the number of leaves from 9 to 3 using the command `clf = tree.DecisionTreeClassifier(max_leaf_nodes=xx)` and observe (and describe) the resulting trees.
- `[7]` Write a program `pract1prog4.py` which opens the IRIS dataset and trains a decision tree using the **Gini** criterion (in `gini-iris.dot`) and a second one using the **entropy** (in `entropy-iris.dot`). Compare the two trees.
- `[8]` Write a program `pract1prog5.py` which creates a dataset using the command `X,Y = make_classification(n_samples=100000,n_features=20,n_informative=15,n_classes=3)`, split the generated data into a learning set and a test set (30% for test). Learn a decision tree on the learning set using the command `clf = tree.DecisionTreeClassifier(max_leaf_nodes=500*i)` (where "i" varies from 1 to 20) then print the score of the classifier (note that `print("%6.4f" %x)` prints a floating point number "x", at least six characters wide, with four characters after the decimal point) on the learning AND on the test set. What do you notice? What is the name of the observed phenomenon?
- `[9]` Same question (`pract1prog6.py`) but with trees of different **depth** using the command `clf = tree.DecisionTreeClassifier(max_depth=i)` (for i from 1 to 40).

You can deduce from the two last questions that the depth of the tree and the max number of leaf are important parameters for the model. They are linked but different: if you control one you also control the other but it is difficult to know in advance which one you should work with. You must tune these parameters using cross validation to find the ones that are best suited to your problem.

Exercise 5: Neural Netwoks on DIGITS (30 minutes)

NB: For this exercise, you can either follow exactly what is asked, using scikit learn, or you can explore the equivalent using PyTorch or TensorFlow. If you use use PyTorch or TensorFlow, you will handle more of the details about the optimization process.

The digits dataset contains 5620 instances that are described by 64 attributes (corresponding to the 8*8 images of integer pixels in the range 0(white)..16(black). The target is a digit between 0 and 9. The MNIST dataset also contains digits (but they are 28*28 images and are more numerous). If you use MNIST, take a subset so your experiments run faster.

Execute the following program:

```
from sklearn.datasets import load_digits
digits = load_digits()
digits.data[0]
digits.images[0]
digits.data[0].reshape(8,8)
digits.target[0]
# you can see the pictures using this piece of code:
from matplotlib import pyplot as plt
plt.matshow(digits.images[0])
plt.gray()
plt.show()
```

```
#to count the number of examples of a particular class, you can use:
Y=digits.target
print(len(Y[Y==0]))
```

Although PyTorch and Tensor Flow can be more efficient (and can use a GPU), Scikit-learn also contains algorithms for learning deep neural networks, with the class MLPClassifier which implements a multi-layer perceptron (MLP) algorithm that is trained using Backpropagation.

```
from sklearn.neural_network import MLPClassifier
X = digits.data
```

Execute the following commands and understand (and write in your report) what they do:

```
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,              # [1]
         hidden_layer_sizes=(5, 2),random_state=1)
clf.fit(X, Y)
```

[2] Write a program `pract1prog7.py` which opens the DIGITS dataset and train a MLP using the default parameters and a small number of hidden layers. Split the data into a learning set and a test set (30% for test) and report the score obtained by your classifier on the test set. Run your program multiple times with the same parameters. Change at least 3 parameters of your classifier (ex: number of neurons, number of layers, learning rate) and report the score your obtain on your test set for each version. What can you conclude?

Exercise 6: Play with your own data (50 minutes)

All the programs created in this exercise should be part of the archive that you will upload on Claroline to be graded.

1. Write a program `pract1prog8.py` which open your dataset, train a KNN classifier (with k= 3) on the data and print the score on the training set.
2. Extend the previous program to split your dataset into training and test (30% for test), train a classifier using a KNN algorithm (with k = 3) on the training set and evaluate it on the test set. Print a number of examples that are not well classified.
3. Extend the previous program to evaluate its accuracy using a leave-one-out cross validation.
4. Do the same process as in the previous three questions in a program `pract1prog9.py` with a Decision tree classifier (report the parameters that you are using).
5. Do the same process as in the previous three questions in a program `pract1prog10.py` with a Neural Network classifier.
6. Which algorithm can explain your concept best? Print the generated models when it is possible. Do they tell you anything interesting?

As a bonus, you can try to visualize the results of the different classifiers by drawing the decision boundary in the input space.