**Course: Introduction to Data Science (DS2006) - Laboratory 14**

**Student:**

Download the file[**iris.csv**](http://iris.csv.py) from blackboard. It is the same Weekly folder as the place where you got this lab from. Take a look at the contents of the file.

**Task 1:** What is the meaning of the information on the first line of the **iris.csv** file?

**Task 2:** What is happening from the second line until the end of the **iris.csv** file?

Now create a new file named [**knn.py**](http://knn.py) where we will proceed to code our first implementation of it.

**Task 3:** The first thing we need to do is to open the dataset, for that we will use pandas built-in dataframe method for opening csv files (read\_csv). An example of how to use it is shown in Figure 1.

import pandas as pd

# Please make sure iris.csv is in the same folder as your code:

df = pd.read\_csv("iris.csv")

Figure 1 - Code Snippet

**Task 4:** The second thing we will do is to check if the DataFrame was loaded with the data from the iris.csv file. Print the result of the dataFrame method head() to do that verification and past the result here:

**Task 5:** Now that we are sure that the data has been loaded successfully we will need to split the data into two groups. One containing all the features and one containing only the classes. To do that we will create two new dataframes, one for the **features** and another one for the **classes**. Figure 2 shows one way we could use to do those data splits:

# Creates a dataframe using the drop method,

# which has two parameters:

# The first parameter tells which labels to remove

# (Columns Name or

# The second parameter tells whether to remove a row index or

# a column name. axis=1 means we want to remove a column.

features = df.drop("class",axis=1)

# Creates a dataframe from just one column:

classes = df["class"]

Figure 2 - Code Snippet

**Task 6:** Make sure that the features and classes dataframe contain the information they should by using the dataframe head() method on both of them. Paste the results here:

**Task 7:** Now that we have the data in this format, we need to use a data partitioning method to split the data into training and test sets to be able to train a classifier and evaluate its results. One way we can do it is to use the train\_test\_split method as shown in Figure 3. The method will take as input the features dataframe, the classes dataframe, the size we want for the test set (in this case a 80/20 split) and a random state which allows us to obtain the exact same split if we run it again with the same data.

# Imports the train\_test\_split functionality from Scikit Learning:

from sklearn.model\_selection import train\_test\_split

#Previous code omitted.

# Split the data into train/test sets

features\_train, features\_test, classes\_train, classes\_test = train\_test\_split(

features, classes, test\_size=0.2, random\_state=10

)

Figure 3 - Code Snippet

**Task 8:** Now that we have the data split into training and test set, we can run the kNN classifier. We will use the implementation that is available in the Scikit-Learning library. Figure 4 shows a code snippet on how to create and train a Knn Object.

# Imports the kNN Implementation from Scikit Learning:

from sklearn.neighbors import KNeighborsClassifier

#Previous code omitted.

# Creates a Knn Classifier Object with k=1

knn = KNeighborsClassifier(n\_neighbors=1)

# Trains this Knn Classifier with the training set obtained previously:

knn.fit(features\_train, classes\_train)

Figure 4 - Code Snippet

**Task 9:** Now we will give the kNN classifier the test set and it will return its predictions. To do that we use the method predict and pass as parameters the features from the test set. Figure 5 illustrates this process.

# Obtains the predictions from the kNN classifier:

predictions = knn.predict(features\_test)

Figure 5 - Code Snippet

**Task 10:** Now that we have the predictions from the kNN classifier, we need to evaluate it. In Lectures 13 and 14 we discussed different evaluation metrics. For starters we could obtain the accuracy of the classifier. The Scikit learning library has several metrics already implemented. Figure 6 shows an example of how to obtain the accuracy for the predictions. Paste the result of your kNN code here:

# Imports the Evaluation Metrics from Scikit Learning:

from sklearn.metrics import accuracy\_score

#Previous code omitted.

#Prints the accuracy:

print("Accuracy:", accuracy\_score(classes\_test, predictions))

Figure 6 - Code Snippet

**Task 11:** Another good to know existing evaluation functionality from Scikit-Learning is the classification\_report.It takes the same input as accuracy\_score but returns more information. An implementation of it is shown in Figure 7. Try it out in your code and paste the result of it here:

# Imports the Classification Report from Scikit Learning:

from sklearn.metrics import classification\_report

#Previous code omitted.

#Prints the classification report:

print(classification\_report(classes\_test, predictions))

Figure 7 - Code Snippet

**Task 12:** If the code from task 11 runs properly, it means that you will get one piece of unknown information in a column called support. In this context this tells you how many instances of that class you have in the test set. However, you will notice that you have an uneven number of examples in the three classes. This is because we are not doing a stratified split (i.e. keeping the same ratio of examples in the test set as in the original dataset). Figure 8 shows an example of this method call using the stratify parameters which receives as an argument the classes.

# Split the data into STRATIFIED train/test sets:

strat\_feat\_train, strat\_feat\_test, strat\_classes\_train, strat\_classes\_test = train\_test\_split(

features, classes, test\_size=0.4, random\_state=10, stratify=classes

)

Figure 8 - Code Snippet

Now create a new file named[**strat-knn.py**](http://knn.py).

**Task 13:** In the [**strat-knn.py**](http://knn.py) file, implement a kNN classifier to perform stratified experiments on the iris dataset. The idea here is that you re-implement the kNN classifier but make all the necessary changes to deal with the function that returns stratified train/test sets from Figure 8.

Now create a new file named[**decision-tree.py**](http://knn.py).

**Task 14:** For this task we will implement the necessary code changes to switch our classifier from a kNN to a **Decision Tree**. Scikit-Learning has an implementation available of this classifier which we will can us by using:

1. The import from the Scikit-Learning Library:

#Imports the Decision Tree Classifier from Scikit Learning:

from sklearn.tree import DecisionTreeClassifier

1. The methods to create and train a decision tree model:

# Create and train Decision Tree model:

dt = DecisionTreeClassifier(random\_state=77)

# Trains this DT Classifier with the training set obtained previously:

dt.fit(features\_train, classes\_train)

1. The method predict to obtain the Predictions from the Decision Tree model:

# Obtains the predictions from the DT classifier:

predictions = dt.predict(features\_test)

Now use this information about the decision tree to implement a decision tree classifier that will perform experiments on the iris dataset, just like the kNN classifier.