**Course: Introduction to Data Science (DS2006) - Laboratory 15**

**Student:**

Download the file[**insurance.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 **insurance.csv** file?

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

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

**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("insurance.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 [insurance.csv](http://insurance.csv.py) 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 **targets** (target values). 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("charges",axis=1)

# Creates a dataframe from just one column:

targets = df["charges"]

Figure 2 - Code Snippet

**Task 6:** Make sure that the **features** and **targets** dataframes 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 regression model 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, targets\_train, targets\_test = train\_test\_split(

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

)

Figure 3 - Code Snippet

**Task 8:** Now that we have the data split into training and test sets, we can run the kNN regressor. 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 Regressor Object.

# Imports the kNN Regressor Implementation from Scikit Learning:

from sklearn.neighbors import KNeighborsRegressor

#Previous code omitted.

# Creates a Knn Regressor Object with k=1

knnr = KNeighborsRegressor(n\_neighbors=1)

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

knnr.fit(features\_train, targets\_train)

Figure 4 - Code Snippet

**Task 9:** Now we will give the kNN regressor 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 Regressor:

predictions = knnr.predict(features\_test)

Figure 5 - Code Snippet

**Task 10:** Now that we have the predictions from the kNN regressor, we need to evaluate it. In Lecture 15 we discussed different evaluation metrics for regression problems. The Scikit learning library has several metrics already implemented. Figure 6 shows an example of how to obtain some metrics for the predictions. Paste the result of your code here:

# Imports the Evaluation Metrics from Scikit Learning:

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

#Previous code omitted.

#Compute the Mean absolute error (MAE):

mae = mean\_absolute\_error(targets\_test, predictions)

print(f"MAE: {mae:.3f}")

#Compute the Mean Squared Error (MSE):

mse = mean\_squared\_error(targets\_test, predictions)

print(f"MSE: {mse:.3f}")

#Compute the Root Mean Squared Error (RMSE):

rmse = np.sqrt(mse)

print(f"RMSE: {rmse:.3f}")

Figure 6 - Code Snippet

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PLEASE DO NOT PROCEED WITHOUT FINISH UP TO TASK 10

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

I mean it :)

PLEASE DO NOT PROCEED WITHOUT FINISH UP TO TASK 10

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

I really mean it :)

PLEASE DO NOT PROCEED WITHOUT FINISH UP TO TASK 10

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If you have done everything in this laboratory so far, you will encounter a

**ValueError: could not convert string to float: 'male'**

This happens because we have non-numeric data in our dataset. As we have discussed in the lecture, we will need to deal with it somehow, and for that we will use one-hot encoding.

For the next set of tasks we will use a new file named [**knnrp.py**](http://knn.py) where we will apply pre-processing techniques to the dataset.

**Task 11:** In order to apply one-hot encoding to our dataset, we need to identify which features are categorical features and use functionalities from ColumnTransformer and OneHotEncoder from Scikit-Learning. Figure 7 shows the functionality needed to use them. Make the necessary changes to your code and run your experiments using the **preprocessed version** of the data. Paste the results here:

# Imports the ColumnTransformer:

from sklearn.compose import ColumnTransformer

# Imports the OneHotEncoder:

from sklearn.preprocessing import OneHotEncoder

#Previous code omitted.

# Define the columns with categorical features:

categorical\_features = ["sex", "smoker", "region"]

# Setups a preprocessor for applying One Hot Encoding to the Categorical features:

preprocessor = ColumnTransformer(

transformers=[("cat", OneHotEncoder(), categorical\_features)]

)

# Apply preprocessing to the training set:

preprocessed\_features\_train = preprocessor.fit\_transform(features\_train)

# Apply preprocessing to the test set:

preprocessed\_features\_test = preprocessor.transform(features\_test)

Figure 7 - Code Snippet

**Task 12:** Now that you have a working version of the regressor try different values for k (at least 5 different values) and paste the results in Table 1.

Table 1 - Experimental results for different values of k with one hot encoding.

| Value used for K | MAE | MSE | RMSE |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Task 13:** Another preprocessing technique that we have seen in the Lecture is the normalization of the numerical attributes. For that reason we will identify the numeric features and use another functionality from the scikit-learning library. Figure 8 shows the functionality needed to use them. Make the necessary changes to your code and run your experiments using the **preprocessed version** of the data with both Normalization and OneHotEncoding. Paste the results here:

**# Imports the MinMaxScaler for Normalization:**

**from sklearn.preprocessing import MinMaxScaler**

**#**Previous code omitted.

**# Define the numeric attributes columns:**

**numeric\_features = ["age", "bmi", "children"]**

**# Updates the Preprocessor to consider the numeric and categorical data columns:**

**preprocessor = ColumnTransformer(**

**transformers=[**

**("num", MinMaxScaler(), numeric\_features),**

**("cat", OneHotEncoder(), categorical\_features)**

**]**

**)**

Figure 8 - Code Snippet

**Task 14:** Now that you have a working version of the regressor with hot encoding and minimax normalization try different values for k (the same values you used in Task 12) and paste the results in Table 2.

Table 2 - Experimental results for different values of k with one hot encoding and normalization.

| Value used for K | MAE | MSE | RMSE |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |