CSEE-5590 - Special Topics  
 Python – Lab 3

**Author:**   
Kilaru Kalyan  
Class Id : 22  
Student ID: 16245031

**Configuration:**  
IDE : pycharm Community Edition  
python : version 3.6.4

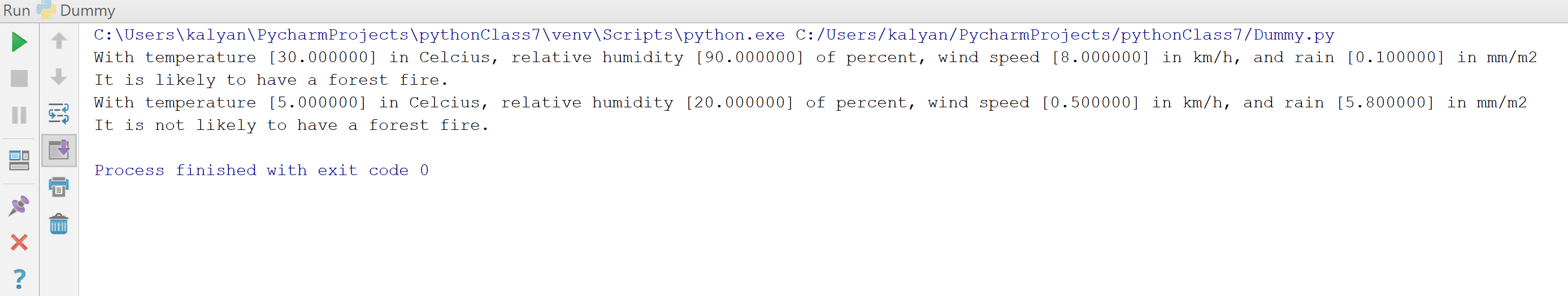
**Objective 1:** To make a prediction using the Linear discriminant analysis and that shows the difference between it and the logistic regression.

**Implementation:**  
I have taken two arrays which represent the data of x-axis and y-axis. I have considered the forest fires data set where y axis represents the burned area in the forest and x axis holds the temperature, humidity and wind. Created the linear discriminant model for the above data. I have considered two predictions one the forest has fire and the other is forest has no fire. When the temperature being high , humidity and wind also being high and the rain is less, the chance for the fire is high. For the second one we don’t have fire when we have all the factors low with high rain.

Both the Logistic and linear discriminant use the same concept of linear model but the difference comes from the parameters, the way they predict them.

**Code:**

*"""  
Data link: http://archive.ics.uci.edu/ml/machine-learning-databases/forest-fires/  
"""***import** csv  
**import** numpy **as** np  
**from** sklearn.discriminant\_analysis **import** LinearDiscriminantAnalysis  
  
x\_axisdata = []  
y\_axisdata = []  
  
**with** open(**"forestfires.csv"**) **as** ff:  
 csvreader = csv.reader(ff, delimiter=**','**)  
   
 next(csvreader)  
 **for** line **in** csvreader:  
 temporary = float(line[8])  
 humidity = float(line[9])  
 wind = float(line[10])  
 rain = float(line[11])  
 area = 1 **if** float(line[12]) > 0 **else** 0  
 x\_axisdata.append([temporary, humidity, wind, rain])  
 y\_axisdata.append(area)  
  
np\_x\_axisdata = np.array(x\_axisdata)  
np\_y\_axisdata = np.array(y\_axisdata)  
  
model = LinearDiscriminantAnalysis()  
model.fit(np\_x\_axisdata, np\_y\_axisdata)  
  
temporary = 30  
humidity = 90  
wind = 8  
rain = 0.1  
print(**" Temperature [%f] , relative humidity [%f] , wind speed [%f] and rain [%f]"** % (temporary, humidity, wind, rain))  
**if** model.predict([[temporary, humidity, wind, rain]])[0]:  
 print(**"Having more chance to have a forest fire"**)  
**else**:  
 print(**"Forest fire may not happen"**)  
  
temporary = 5  
humidity = 20  
wind = 0.5  
rain = 5.8  
print(**"With temperature [%f], relative humidity [%f] wind speed [%f] rain [%f]"** % (temporary, humidity, wind, rain))  
**if** model.predict([[temporary, humidity, wind, rain]])[0]:  
 print(**"Having more chance to have a forest fire"**)  
**else**:  
 print(**"Forest fire may not happen"**)

**Input/Output:**

**Objective 2:** To implement SVM with linear kernel and RBF kernel and to find out the accuracy in both the cases.

**Implementation:**First, we need to implement SVM with both linear and RBF kernals on a data set. The data will be loaded in to the x and y which is the data and the target. Here I have taken 80 percent of training data and the 20 percent of testing data. And I observed that linear kernel is better than RBF when the accuracy is considered.

**Code:**

**from** sklearn **import** svm

**from** sklearn **import** datasets

**from** sklearn **import** metrics

**from** sklearn.model\_selection **import** train\_test\_split  
  
*# Loading the data*iris = datasets.load\_wine()  
*# x\_data and y\_target are loaded with data and the target*x\_data = iris.data  
y\_target = iris.target

*# Divide the data into 20 percent testing the remaining is training data.*x\_training, x\_testing, y\_training, y\_testing = train\_test\_split(x\_data, y\_target, test\_size=0.2)  
  
print(**"SVM with linear kernel:"**)  
*# Linear Model*

linear\_kernel\_model = svm.SVC(kernel=**"linear"**)  
linear\_kernel\_model.fit(x\_training, y\_training)

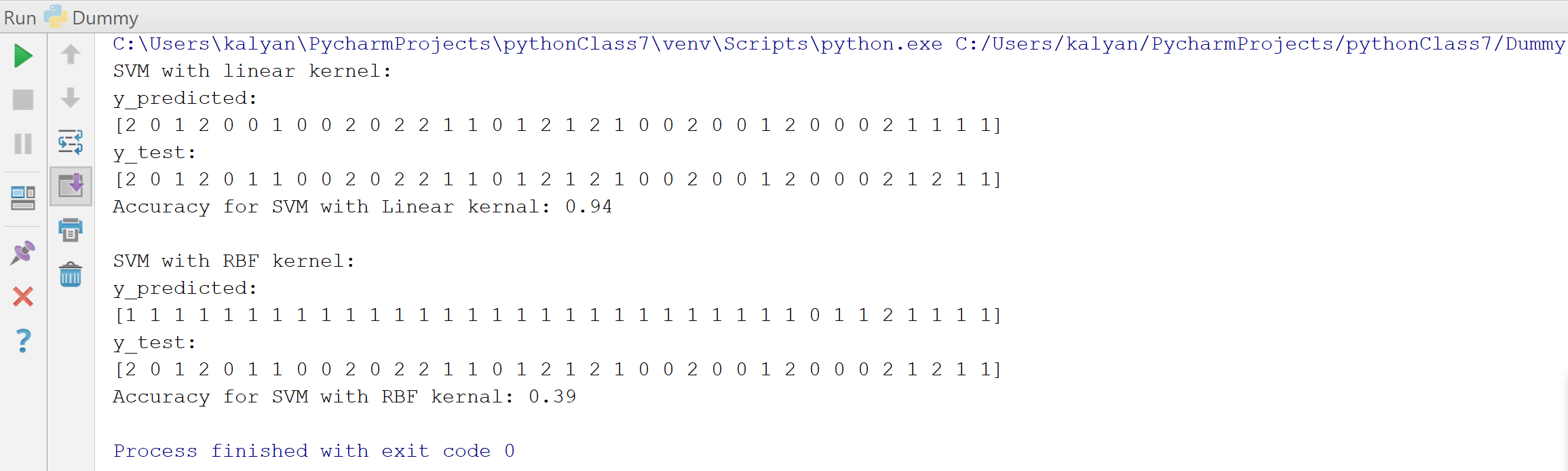
*# Generate y\_predicted from trained model with data from x\_testing*y\_predicted = linear\_kernel\_model.predict(x\_testing)  
*# Print data on both y\_predicted and y\_testing*print(**"y\_predicted:"**)  
print(y\_predicted)

print(**"y\_testing:"**)  
print(y\_testing)  
*# accuracy score by compare y\_testing and y\_predicted*print(**"Accuracy for SVM with Linear kernal: %.2f"** % metrics.accuracy\_score(y\_testing, y\_predicted))  
  
print(**"\nSVM with RBF kernel:"**)  
rbf\_model = svm.SVC(kernel=**"rbf"**)  
rbf\_model.fit(x\_training, y\_training)

*# y\_predicted from trained model with data from x\_testing*y\_predicted = rbf\_model.predict(x\_testing)  
*# Print data on both y\_predicted and y\_testing*

print(**"y\_predicted:"**)  
print(y\_predicted)  
print(**"y\_testing:"**)  
print(y\_testing)  
*# accuracy score by compare y\_testing and y\_predicted*print(**"Accuracy for SVM with RBF kernal: %.2f"** % metrics.accuracy\_score(y\_testing, y\_predicted))

**Input/Output:**



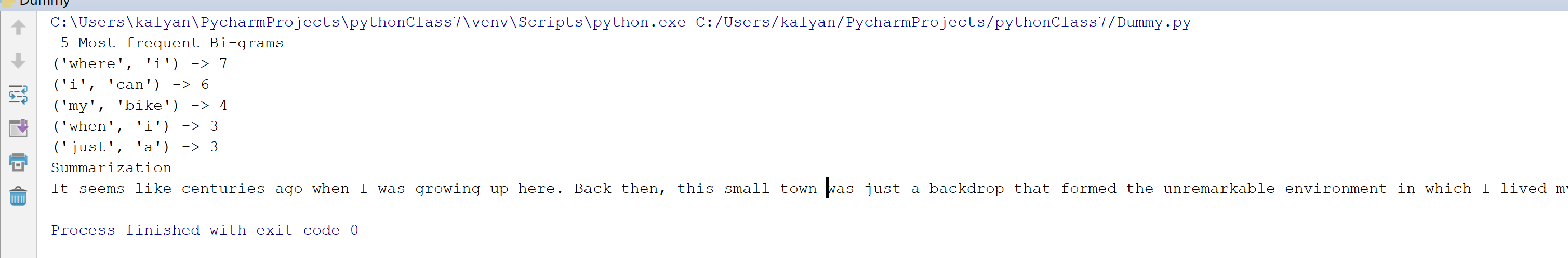
**Objective 3:** To write a program to implement all the natural language processing techniques to perform operation on the data and analyze them.  
  
**Implementation:**First I used the function to get the tokens and then used the lemmatization method to lemmatize the tokens that I got from the data. Secondly I created the function to generate the bi-gram from the tokens that I got previously. Then I calculated the frequency of each bigram. I used the collections to get the top five bi-grams

**Code:**

**

**

**Input/Output:**



**Objective 4:** To compare the accuracy for K nearest neighbor algorithm for various K values.   
  
**Implementation:**The accuracy increases with the increase in the K value. When the small data set is considered we get the below graph which represents that the accuracy will be a almost same from one point of the K value. When we have the low value of K it means it is most flexible fit.

**Code:**  
*import pandas*

*import matplotlib.pyplot as pl*

*from sklearn.cluster import KMeans  
variables = pandas.read\_csv('sample\_stocks.csv')  
Y = variables[['returns']]  
X = variables[['dividendyield']]*

*kmeans = []*

*score = []*

*x=50*

*for i in range(1, x):*

*kmeans.append(KMeans(i))*

*for i in range(len(kmeans)):*

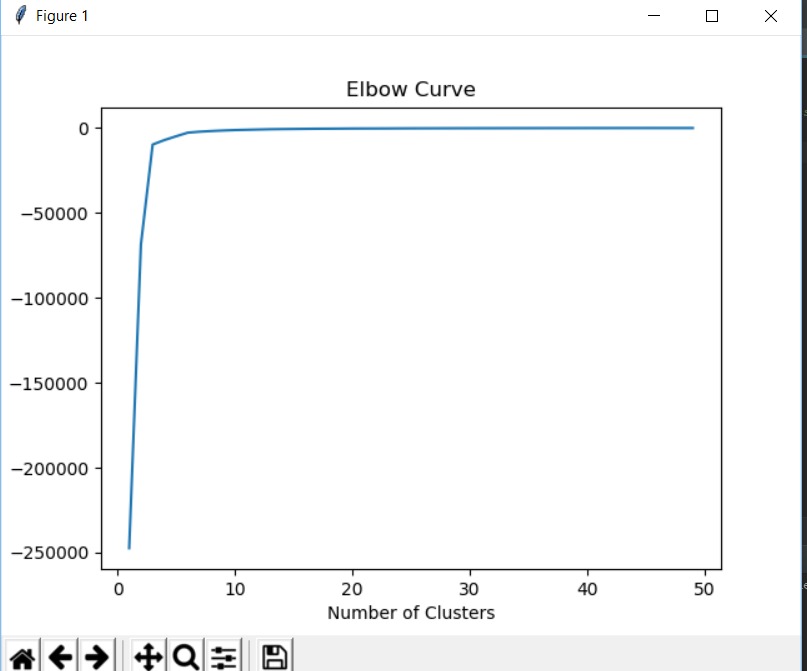
*score.append(kmeans[i].fit(Y).score(Y))*

*pl.plot(range(1,x), score)*

*pl.xlabel('Number of Clusters')*

*pl.ylabel('Score')*

*pl.title('Elbow Curve')*

*pl.show()***Input/Otput:**

**Deployment:**

The code is written in python in pycharm IDE  
Imported the numpy module for the objective-4  
Ran the code in IDE and the outputs are checked in console

**Limitations:**Code is not robust  
Validations are not taken care for inputs in dictionary values.

**References:**https://www.python-courses.eu/dictionsaries.php