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.

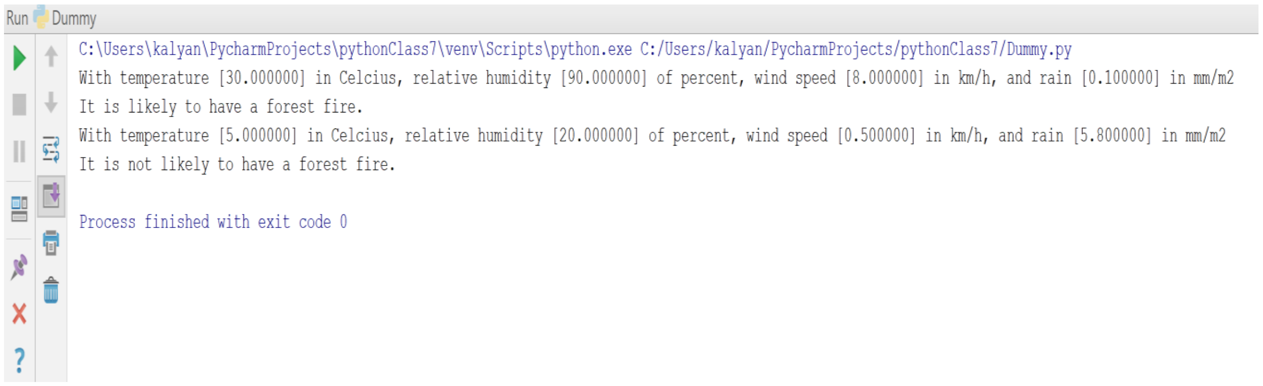
Implementation:
I have taken two arrays which represent the data of x-axis and y-axis. I have considered the forest fires data set where y axis represents the burned area in the forest and x axis holds the temperature, humidity and wind. Created the linear discriminant model for the above data. I have 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:



```
C:\Users\kalyan\PycharmProjects\pythonClass7\venv\Scripts\python.exe C:/Users/kalyan/PycharmProjects/pythonClass7/Dummy.py
With temperature [30.000000] in Celcius, relative humidity [90.000000] of percent, wind speed [8.000000] in km/h, and rain [0.100000] in mm/m2
It is likely to have a forest fire.
With temperature [5.000000] in Celcius, relative humidity [20.000000] of percent, wind speed [0.500000] in km/h, and rain [5.800000] in mm/m2
It is not likely to have a forest fire.

Process finished with exit code 0
```

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

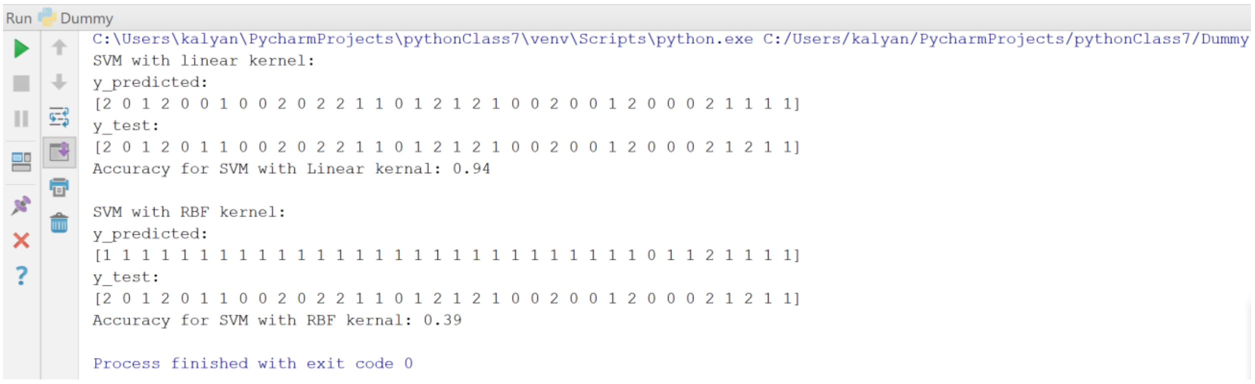
Implementation:

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:



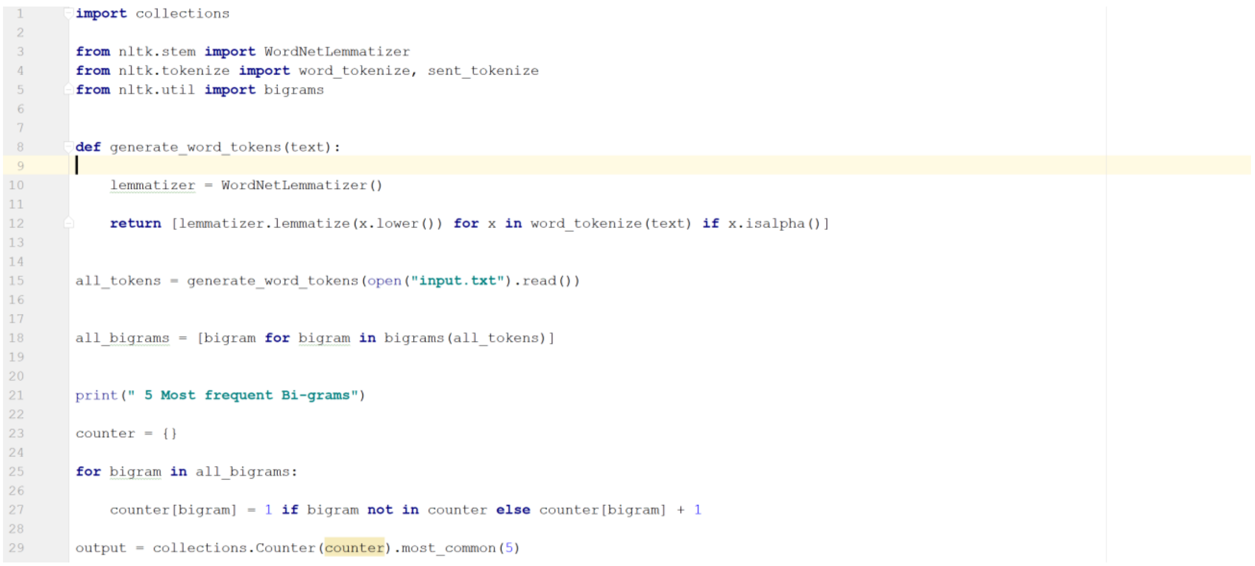
```
C:\Users\kalyan\PycharmProjects\pythonClass7\venv\Scripts\python.exe C:/Users/kalyan/PycharmProjects/pythonClass7/Dummy
SVM with linear kernel:
y_predicted:
[2 0 1 2 0 0 1 0 0 2 0 2 2 1 1 0 1 2 1 2 1 0 0 2 0 0 1 2 0 0 0 2 1 1 1 1]
y_test:
[2 0 1 2 0 1 1 0 0 2 0 2 2 1 1 0 1 2 1 2 1 0 0 2 0 0 1 2 0 0 0 2 1 2 1 1]
Accuracy for SVM with Linear kernal: 0.94

SVM with RBF kernel:
y_predicted:
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 2 1 1 1 1]
y_test:
[2 0 1 2 0 1 1 0 0 2 0 2 2 1 1 0 1 2 1 2 1 0 0 2 0 0 1 2 0 0 0 2 1 2 1 1]
Accuracy for SVM with RBF kernal: 0.39

Process finished with exit code 0
```

Objective 3: To write a program to implement all the natural language processing techniques to perform operation on the data and analyze them.

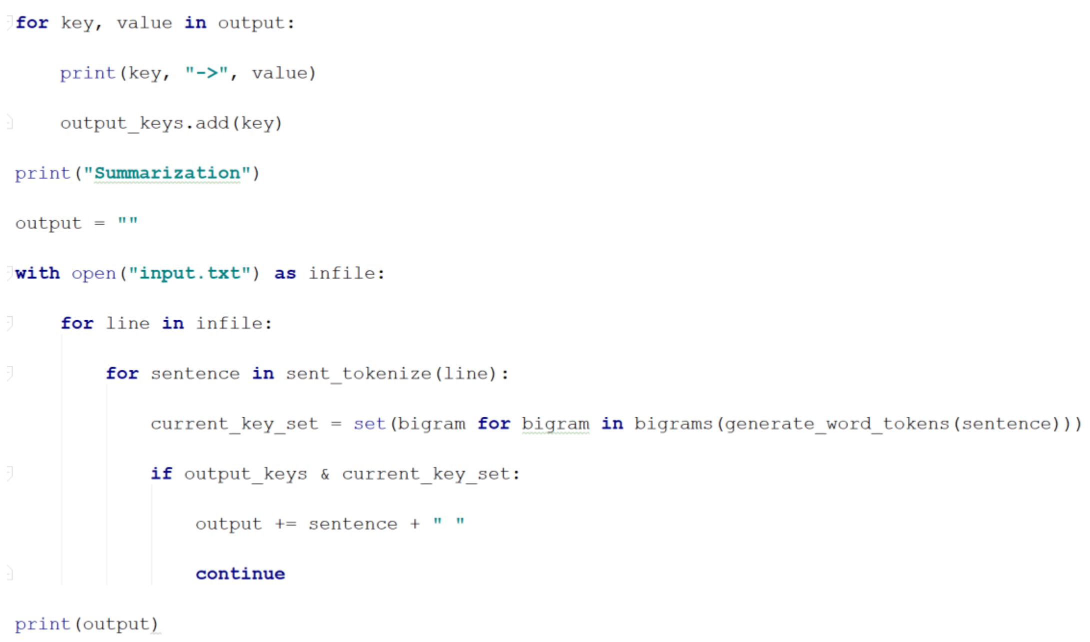
Implementation:

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:

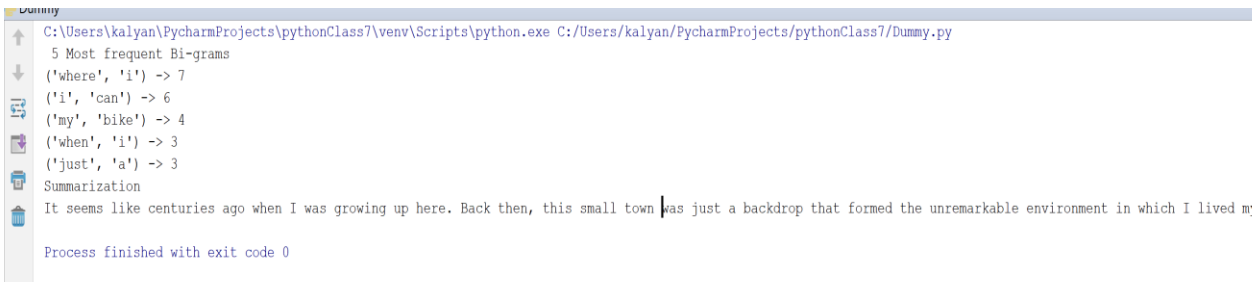


```python
import collections

from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.util import bigrams


def generate_word_tokens(text):

    lemmatizer = WordNetLemmatizer()

    return [lemmatizer.lemmatize(x.lower()) for x in word_tokenize(text) if x.isalpha()]


all_tokens = generate_word_tokens(open("input.txt").read())


all_bigrams = [bigram for bigram in bigrams(all_tokens)]


print(" 5 Most frequent Bi-grams")

counter = {}

for bigram in all_bigrams:

    counter[bigram] = 1 if bigram not in counter else counter[bigram] + 1

output = collections.Counter(counter).most_common(5)
```



```
for key, value in output:
    print(key, "->", value)
    output_keys.add(key)
print("Summarization")
output = ""
with open("input.txt") as infile:
    for line in infile:
        for sentence in sent_tokenize(line):
            current_key_set = set(bigram for bigram in bigrams(generate_word_tokens(sentence)))
            if output_keys & current_key_set:
                output += sentence + " "
                continue
print(output)
```

Input/Output:



```
C:\Users\kalyan\PycharmProjects\pythonClass7\venv\Scripts\python.exe C:/Users/kalyan/PycharmProjects/pythonClass7/Dummy.py
 5 Most frequent Bi-grams
('where', 'i') -> 7
('i', 'can') -> 6
('my', 'bike') -> 4
('when', 'i') -> 3
('just', 'a') -> 3
Summarization
It seems like centuries ago when I was growing up here. Back then, this small town was just a backdrop that formed the unremarkable environment in which I lived m

Process finished with exit code 0
```

Objective 4: To compare the accuracy for K nearest neighbor algorithm for various K values.

Implementation:

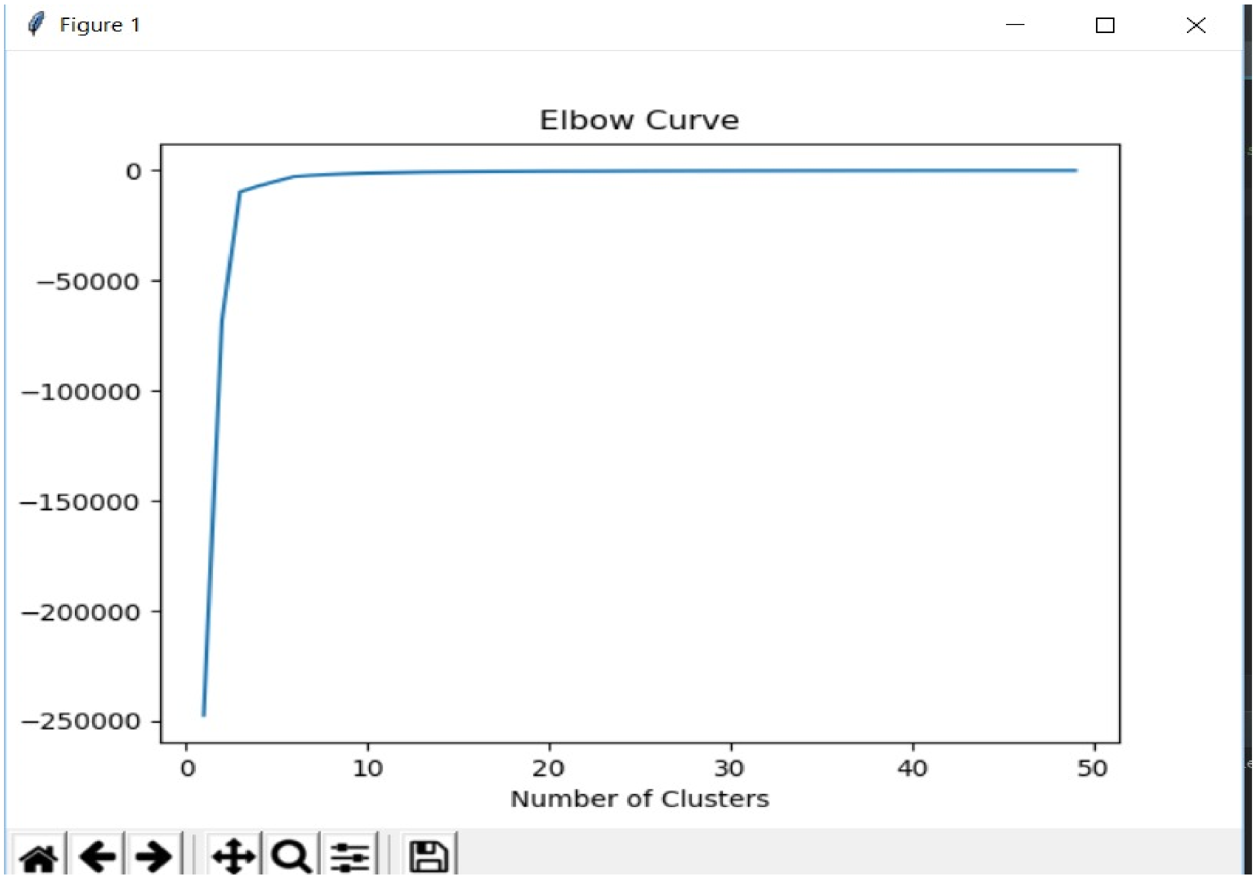
The accuracy increases with the increase in the K value. When the small data set is considered we get the below graph which represents that the accuracy will be a almost same from one point of the K value. When we have the low value of K it means it is most flexible fit.

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