AUTHOR

Name: Rohit Abhishek
Student Id: 16158762

OBJECTIVE

The objective is to get familiar with the concepts of regression, support vector machine classification and k nearest neighbor.

FEATURES

The features of each problem are :
Problem 1: Applying Linear Discriminant Analysis in on the dataset and analyzing the difference between logistic regression and linear discriminant analysis.
Problem 2: Applying Support Vector Machine classification to a given dataset. The data split is 80% training and 20% testing. Linear kernel and RBF kernel has been used to test and analyze the accuracy.
Problem 3: Applying lemmatization, bigram to an input file.
Problem 4: Studying K nearest neighbor and analyzing the values of k.

CONFIGURATION

Each program was implement in Spyder which was installed on MAC Os with 2.5 GHz Intel Core i5 processor, 16 GB 1600 MHz DDR3 memory and storage of 500 GB.

INPUT/OUTPUT AND CODE IMPLEMENTATION

This section presents the code implementation.

Question 1)

Pick any dataset from the dataset sheet in class sheet and make one prediction model using your imagination with Linear Discriminant Analysis.

Approach:

In this problem I have used wine dataset to analyze the LDA and logistic regression.
LDA is a supervised dimensionality reduction method. It is used to extract new independent variable which is used to separate the dependent variable of most classes.

In this we will deal with classification problem and will use logistic regression to solve the classification problem.
Since in this dataset we have 13 dimensions. So LDA will be used to reduce the dimensionality and get 2 independent variable that explain the most variance and then logistic regression is applied.
If the predictors are multivariate Gaussian, LDA is optimal whereas logistic regression is semi-parametric as they make weaker assumptions about the predictors. When the dependent variable has more than one category, LDA is applied.
Logistic regression is more robust to gross outliners because it relies on fewer assumptions where as LDA is not. As LDA uses more information about the data, it tends to estimate the parameters more efficiently as compare to logistic regression
On the other hand, LDA is not robust to gross outliers. Because logistic regression relies on fewer assumptions, it seems to be more robust to non-Gaussian type of data.

The libraries used for this problem are

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.linear_model import LogisticRegression
from matplotlib.colors import ListedColormap
```

Here, first we need to import the wine dataset and define the datasets and the features. Then the training and the testing sets are defined. The below code is used for that.

```
dataset=pd.read_csv('wine.csv')
X_set=dataset.iloc[:,0:13].values
y_set=dataset.iloc[:,13].values
X_training, X_testing,y_training,y_testing=train_test_split(X_set,y_set,test_size=0.2,random_state=0)
```

Below is the code to apply Linear Discriminant Analysis with n_component parameter value to be 2, which indicates the components to be retained for the input datasets

```
l=LDA(n_components=2)
X_training=l.fit_transform(X_training,y_training)
X_testing=l.transform(X_testing)
```

Below code is to apply the logistic regression on the training and the testing data fitted by LDA

```
#Applying Logistics Regression
clsf=LogisticRegression(random_state=0)
clsf.fit(X_training,y_training)
```

The code below is used to plot the training set data

```
# Plot the data- Training set

set_X, set_y = X_training, y_training
X1, X2 = np.meshgrid(np.arange(start = set_X[:, 0].min() - 1, stop = set_X[:, 0].max() + 1, step = 0.01),
                     np.arange(start = set_X[:, 1].min() - 1, stop = set_X[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, clsf.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.3, cmap = ListedColormap(('blue', 'yellow')))
print ("Accuracy Score is")
print (clsf.score(set_X, set_y))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for k, l in enumerate(np.unique(set_y)):
    plt.scatter(set_X[set_y == l, 0], set_X[set_y == l, 1],
                c = ListedColormap(('blue', 'yellow'))(k), label = l)
plt.title('Training set')
plt.xlabel('Linear Discriminant 1')
plt.ylabel('Linear Discriminant 2')
plt.legend()
plt.show()
```

The below code is used to plot the testing set data

```
# Plot the data- Testing Set

set_X, set_y = X_testing, y_testing
X1, X2 = np.meshgrid(np.arange(start = set_X[:, 0].min() - 1, stop = set_X[:, 0].max() + 1, step = 0.01),
                     np.arange(start = set_X[:, 1].min() - 1, stop = set_X[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, clsf.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.3, cmap = ListedColormap(('blue', 'yellow')))
print ("Accuracy Score is")
print (clsf.score(set_X, set_y))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for k, l in enumerate(np.unique(set_y)):
    plt.scatter(set_X[set_y == l, 0], set_X[set_y == l, 1],
                c = ListedColormap(('blue', 'yellow'))(k), label = l)
plt.title('Testing set')
plt.xlabel('Linear Discriminant 1')
plt.ylabel('Linear Discriminant 2')
plt.legend()
plt.show()
```

The full code is shown below

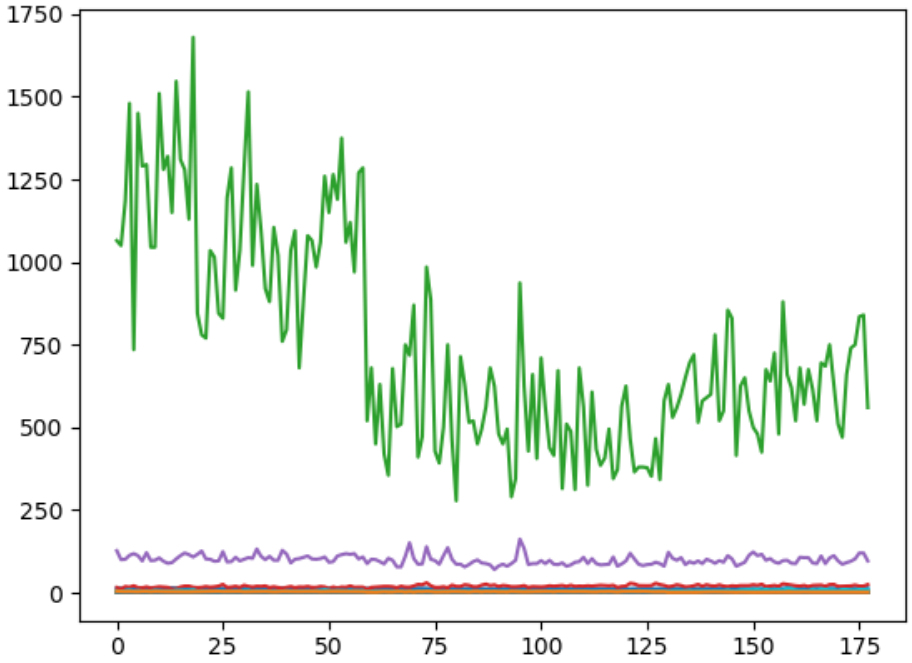
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Mar  4 15:33:04 2018

@author: rohitabhishek
"""
#LDA
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.linear_model import LogisticRegression
from matplotlib.colors import ListedColormap


dataset=pd.read_csv('wine.csv')
X_set=dataset.iloc[:,0:13].values
y_set=dataset.iloc[:,13].values
X_training,
X_testing,y_training,y_testing=train_test_split(X_set,y_set,test_size=0.2)



#Applying Linear Discriminant Analysis

l=LDA(n_components=2)
X_training=l.fit_transform(X_training,y_training)
X_testing=l.transform(X_testing)

#Applying Logistics Regression
clsf=LogisticRegression()
clsf.fit(X_training,y_training)
```

```
#Test Result prediction
y_predication=clsf.predict(X_testing)


# Plot the data- Training set

set_X, set_y = X_training, y_training
X1, X2 = np.meshgrid(np.arange(start = set_X[:, 0].min() - 1, stop = set_X[:,
0].max() + 1, step = 0.01),
                     np.arange(start = set_X[:, 1].min() - 1, stop = set_X[:, 1].max() + 1,
step = 0.01))
plt.contourf(X1, X2, clsf.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.3, cmap = ListedColormap(('blue', 'yellow')))
print ("Accuracy Score is")
print (clsf.score(set_X, set_y))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for k, l in enumerate(np.unique(set_y)):
    plt.scatter(set_X[set_y == l, 0], set_X[set_y == l, 1],
                c = ListedColormap(('blue', 'yellow'))(k), label = l)
plt.title('Training set')
plt.xlabel('Linear Discriminant 1')
plt.ylabel('Linear Discriminant 2')
#plt.legend()
plt.show()



# Plot the data- Testing Set

set_X, set_y = X_testing, y_testing
X1, X2 = np.meshgrid(np.arange(start = set_X[:, 0].min() - 1, stop = set_X[:,
0].max() + 1, step = 0.01),
                     np.arange(start = set_X[:, 1].min() - 1, stop = set_X[:, 1].max() + 1,
step = 0.01))
plt.contourf(X1, X2, clsf.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.3, cmap = ListedColormap(('blue', 'yellow')))
print ("Accuracy Score is")
```

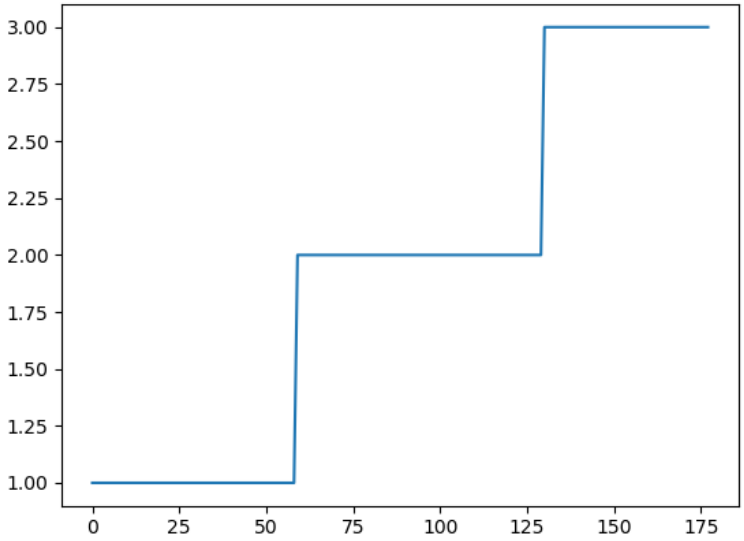
```
print (clsf.score(set_X, set_y))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for k, l in enumerate(np.unique(set_y)):
    plt.scatter(set_X[set_y == l, 0], set_X[set_y == l, 1],
                c = ListedColormap(('blue', 'yellow'))(k), label = l)
plt.title('Testing set')
plt.xlabel('Linear Discriminant 1')
plt.ylabel('Linear Discriminant 2')
#plt.legend()
plt.show()
```

Simulations:
Accuracy Score of training data **0.9929577464788732**
Accuracy Score is of testing data **1.0**

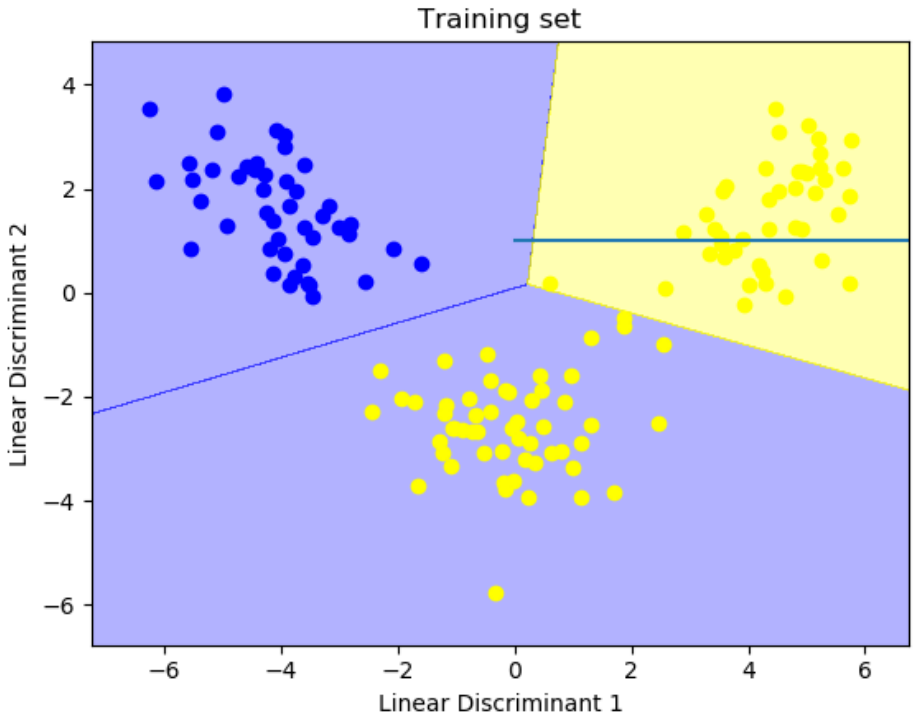
Plotting X_set



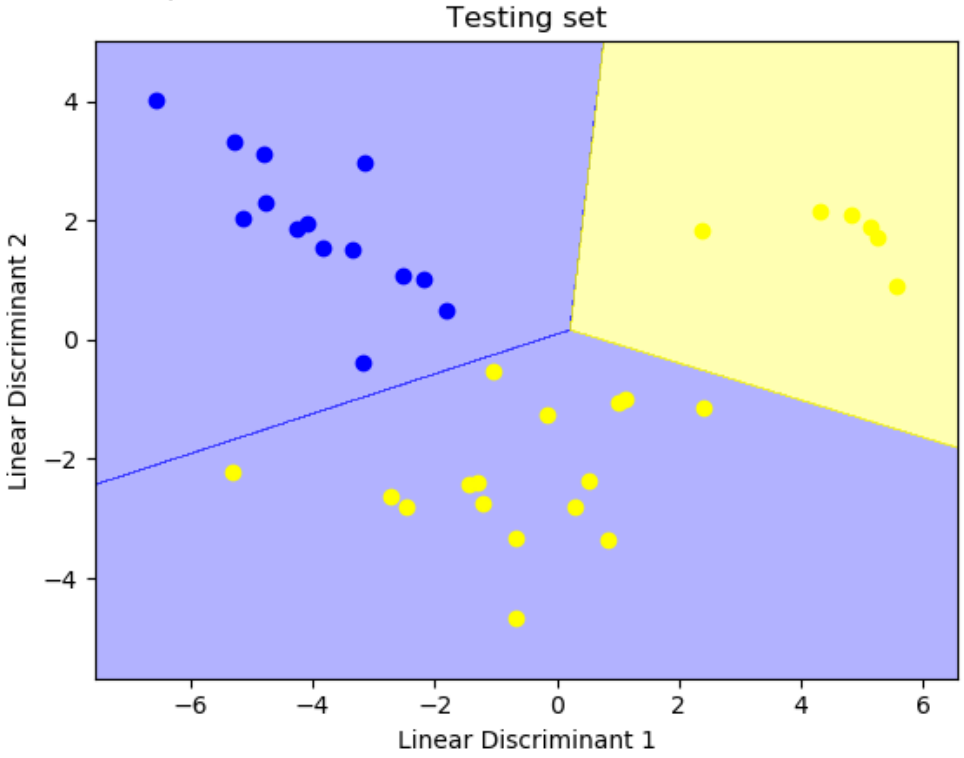
Plotting Y_set



Plotting the training set



Plotting the testing set



Question 2)

Implement Support Vector Machine classification,
1) Choose one of the dataset using the datasets features in the scikit -learn
2) Load the dataset
3) According to your dataset, split the data to 20% testing data, 80% training data(you can also use any other number)
4) Apply SVC with Linear kernel
5) Apply SVC with RBF kernel
6) Report the accuracy of the model on both models separately and report their differences if there is
7) Report your view how can you increase the accuracy and which kernel is the best for your dataset and why

Approach:

For this problem I have used Social_Network_Ads dataset which comprises of User ID, Gender, Age, Estimated Salary and Purchased fields.
So here we need to import the datasets, then split the data into testing and training. Make 2 classifiers for linear kernel and rbf kernel. And finally calculate the accuracy.

Importing the libraries

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cross_validation import train_test_
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from matplotlib.colors import ListedColormap
```

Importing the social network ads datasets and setting the data and target values.

```
# Import the datasets for social network ads
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values #Column 2,3 for age and estimated salary
y = dataset.iloc[:, 4].values #feature dataset
```

For this we need to perform feature scaling so that smaller numeric values are not dominated by the greater ones. The code for the same is below

```
#Feature Scaling so that smaller numeric values are not dominted by the greater ones
scaling = StandardScaler()
X_training = scaling.fit_transform(X_training) #Scaling the training data
X_testing = scaling.transform(X_testing) #Scaling the testing data
```

The below code is used to apply support vector to the training set where clf1 is used for linear kernel and clf2 is used for rbf kernel

```
# Applying Support Vector to the Training set

clf1 = SVC(kernel = 'linear', random_state = 0) #Applying linear kernel
clf2 = SVC(kernel = 'rbf', random_state = 0) #Applying RBF kernel
clf1.fit(X_training, y_training)
clf2.fit(X_training, y_training)
```

The below code is used for plotting the training and the testing set for linear kenel

```
# Plot the Training set for linear kernel

set_X, set_y = X_training, y_training
X1, X2 = np.meshgrid(np.arange(start = set_X[:, 0].min() - 1, stop = set_X[:, 0].max() + 1, step = 0.01),
                     np.arange(start = set_X[:, 1].min() - 1, stop = set_X[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, clf1.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.3, cmap = ListedColormap(('blue', 'yellow')))
print ("Accuracy Score is")
print (clf1.score(set_X, set_y))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for k, l in enumerate(np.unique(set_y)):
    plt.scatter(set_X[set_y == l, 0], set_X[set_y == l, 1],
                c = ListedColormap(('blue', 'yellow'))(k), label = l)
plt.title('Linear Kernel:Training set')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Plot the Testing set for linear kernel

set_X, set_y = X_testing, y_testing
X1, X2 = np.meshgrid(np.arange(start = set_X[:, 0].min() - 1, stop = set_X[:, 0].max() + 1, step = 0.01),
                     np.arange(start = set_X[:, 1].min() - 1, stop = set_X[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, clf1.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.3, cmap = ListedColormap(('blue', 'yellow')))
print ("Accuracy Score is")
print (clf1.score(set_X, set_y))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for k, l in enumerate(np.unique(set_y)):
    plt.scatter(set_X[set_y == l, 0], set_X[set_y == l, 1],
                c = ListedColormap(('blue', 'yellow'))(k), label = l)
plt.title('Linear Kernel:Testing set')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

The below code is used for plotting the training and the testing set for rbf kenel

```
# Plot the Training set for RBF kernel

set_X, set_y = X_training, y_training
X1, X2 = np.meshgrid(np.arange(start = set_X[:, 0].min() - 1, stop = set_X[:, 0].max() + 1, step = 0.01),
                     np.arange(start = set_X[:, 1].min() - 1, stop = set_X[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, clf2.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.3, cmap = ListedColormap(('blue', 'yellow')))
print ("Accuracy Score is")
print (clf2.score(set_X, set_y))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for k, l in enumerate(np.unique(set_y)):
    plt.scatter(set_X[set_y == l, 0], set_X[set_y == l, 1],
                c = ListedColormap(('blue', 'yellow'))(k), label = l)
plt.title('RBF Kernel:Training set')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Plot the Testing set for RBF kernel

set_X, set_y = X_testing, y_testing
X1, X2 = np.meshgrid(np.arange(start = set_X[:, 0].min() - 1, stop = set_X[:, 0].max() + 1, step = 0.01),
                     np.arange(start = set_X[:, 1].min() - 1, stop = set_X[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, clf2.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.3, cmap = ListedColormap(('blue', 'yellow')))
print ("Accuracy Score is")
print (clf2.score(set_X, set_y))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for k, l in enumerate(np.unique(set_y)):
    plt.scatter(set_X[set_y == l, 0], set_X[set_y == l, 1],
                c = ListedColormap(('blue', 'yellow'))(k), label = l)
plt.title('RBF Kernel:Testing set')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

The full code is shown below:

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  5 15:05:34 2018

@author: rabhishe
"""

# Support Vector Machine (SVM)

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cross_validation import train_test_split #importing the library for
training and testing of the data
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from matplotlib.colors import ListedColormap

# Import the datasets for social network ads
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values #Column 2,3 for age and estimated salary
y = dataset.iloc[:, 4].values #feature dataset



X_training, X_testing, y_training, y_testing = train_test_split(X, y, test_size = 0.2,
random_state = 0) #Traiining set and Testing set

#Feature Scaling so that smaller numeric values are not dominted by the greater
ones
scaling = StandardScaler()
X_training = scaling.fit_transform(X_training) #Scaling the training data
X_testing = scaling.transform(X_testing) #Scaling the testing data

# Applying Support Vector to the Training set

clf1 = SVC(kernel = 'linear', random_state = 0) #Applying linear kernel
```

```
clf2 = SVC(kernel = 'rbf', random_state = 0) #Applying RBF kernel
clf1.fit(X_training, y_training)
clf2.fit(X_training, y_training)

# Test Result prediction
y_predict1 = clf1.predict(X_testing)
y_predict2 = clf2.predict(X_testing)



# Plot the Training set for linear kernel

set_X, set_y = X_training, y_training
X1, X2 = np.meshgrid(np.arange(start = set_X[:, 0].min() - 1, stop = set_X[:,
0].max() + 1, step = 0.01),
                     np.arange(start = set_X[:, 1].min() - 1, stop = set_X[:, 1].max() + 1,
step = 0.01))
plt.contourf(X1, X2, clf1.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.3, cmap = ListedColormap(('blue', 'yellow')))
print ("Accuracy Score for Training set for Linear Kernel is")
print (clf1.score(set_X, set_y))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for k, l in enumerate(np.unique(set_y)):
    plt.scatter(set_X[set_y == l, 0], set_X[set_y == l, 1],
                c = ListedColormap(('blue', 'yellow'))(k), label = l)
plt.title('Linear Kernel:Training set')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Plot the Testing set for linear kernel

set_X, set_y = X_testing, y_testing
X1, X2 = np.meshgrid(np.arange(start = set_X[:, 0].min() - 1, stop = set_X[:,
0].max() + 1, step = 0.01),
                     np.arange(start = set_X[:, 1].min() - 1, stop = set_X[:, 1].max() + 1,
step = 0.01))
```

```
plt.contourf(X1, X2, clf1.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.3, cmap = ListedColormap(('blue', 'yellow')))
print ("Accuracy Score for Testing set for Linear Kernel is")
print (clf1.score(set_X, set_y))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for k, l in enumerate(np.unique(set_y)):
    plt.scatter(set_X[set_y == l, 0], set_X[set_y == l, 1],
                c = ListedColormap(('blue', 'yellow'))(k), label = l)
plt.title('Linear Kernel:Testing set')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
# Plot the Training set for RBF kernel

set_X, set_y = X_training, y_training
X1, X2 = np.meshgrid(np.arange(start = set_X[:, 0].min() - 1, stop = set_X[:,
0].max() + 1, step = 0.01),
                     np.arange(start = set_X[:, 1].min() - 1, stop = set_X[:, 1].max() + 1,
step = 0.01))
plt.contourf(X1, X2, clf2.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.3, cmap = ListedColormap(('blue', 'yellow')))
print ("Accuracy Score for Training set for RBF Kernel is")
print (clf2.score(set_X, set_y))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for k, l in enumerate(np.unique(set_y)):
    plt.scatter(set_X[set_y == l, 0], set_X[set_y == l, 1],
                c = ListedColormap(('blue', 'yellow'))(k), label = l)
plt.title('RBF Kernel:Training set')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
```

```
plt.legend()
plt.show()

# Plot the Testing set for RBF kernel

set_X, set_y = X_testing, y_testing
X1, X2 = np.meshgrid(np.arange(start = set_X[:, 0].min() - 1, stop = set_X[:,
0].max() + 1, step = 0.01),
                     np.arange(start = set_X[:, 1].min() - 1, stop = set_X[:, 1].max() + 1,
step = 0.01))
plt.contourf(X1, X2, clf2.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.3, cmap = ListedColormap(('blue', 'yellow')))
print ("Accuracy Score for Testing set for RBF Kernel is")
print (clf2.score(set_X, set_y))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for k, l in enumerate(np.unique(set_y)):
    plt.scatter(set_X[set_y == l, 0], set_X[set_y == l, 1],
                c = ListedColormap(('blue', 'yellow'))(k), label = l)
plt.title('RBF Kernel:Testing set')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

Accuracy Score
Accuracy Score for Training set for Linear Kernel is **0.821875**
Accuracy Score for Testing set for Linear Kernel is **0.9125**

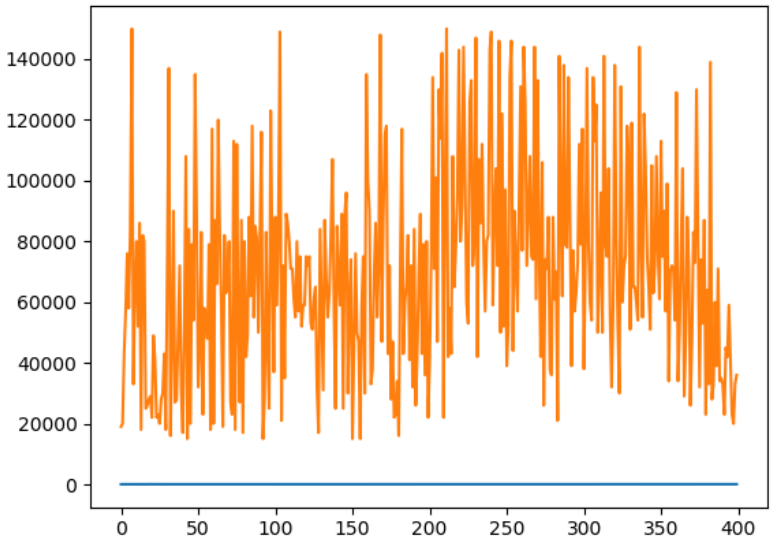
Accuracy Score for Training set for RBF Kernel is **0.903125**
Accuracy Score for Testing set for RBF Kernel is **0.95**

As we can see from the above values that the accuracy of RBF is more as compared to Linear Kernel.
When the number of features is larger than number of observations, linear kernel should be used whereas when the observations are larger that the number of features, RBF kernel should be used.

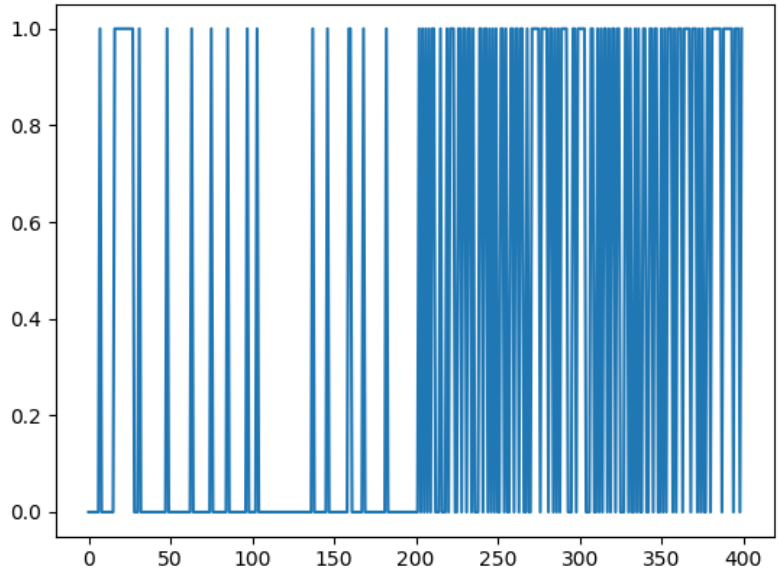
So, for my dataset RBF is the best fit. For this dataset my testing size is 0.2. However, if the testing size is 0.1, the accuracy score of testing data for linear kernel becomes more than that of RBF. One way to increase the accuracy is reduce the difference in error rate between the test data and training data i.e. reducing overfitting. So, using dimensionality reducing techniques, the accuracy can be increased.

Simulations

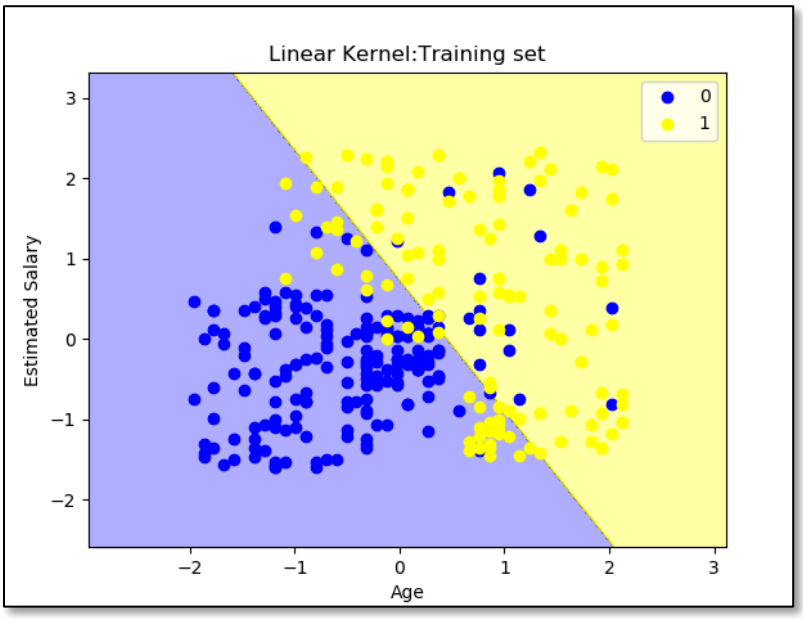
Plotting X



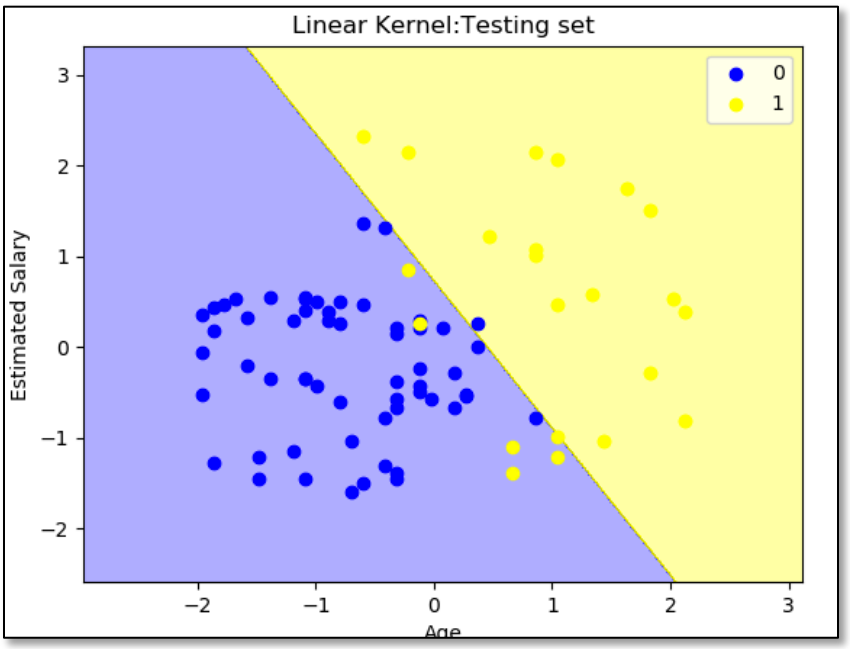
Plotting Y



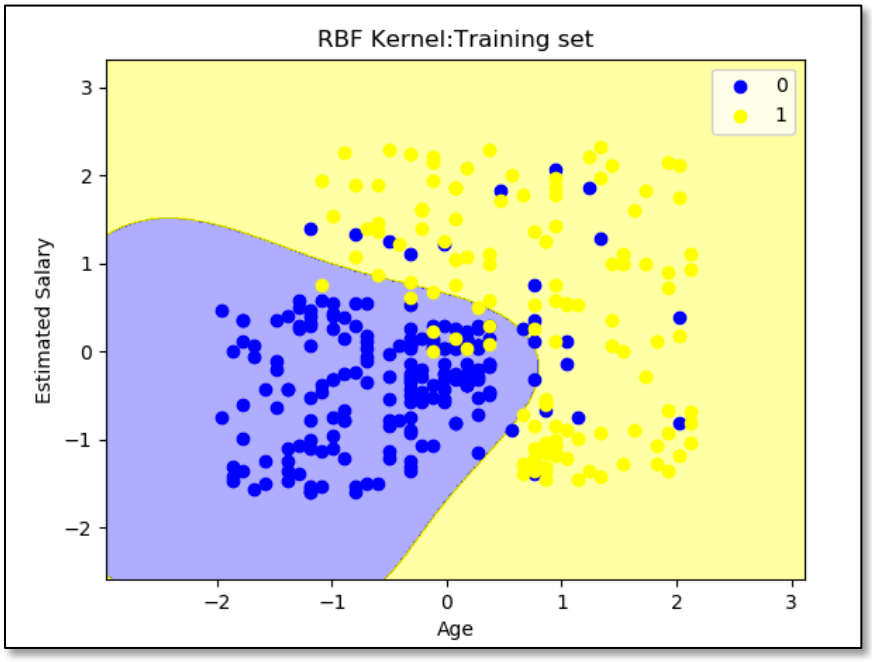
Plotting the Training set for Linear Kernel



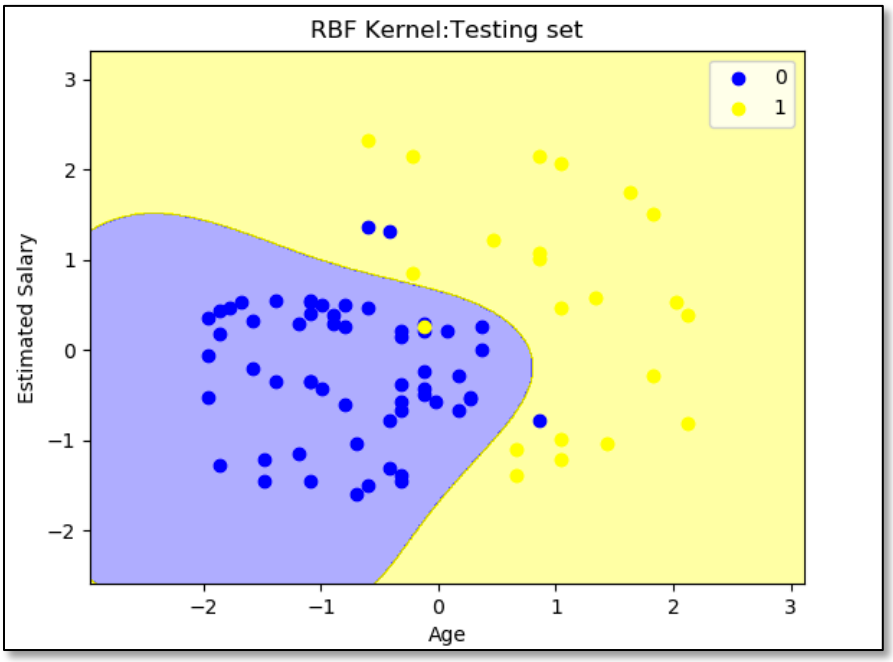
Plotting the Testing set for linear Kernel



Plotting the Training set for RBF Kernel



Plotting the Testing set for RBF Kernel



Question 3)

Write a program to take an Input file.
Use the simple approach below to summarize a text file:
-Read the file
-Using Lemmatization, apply lemmatization on the words
-Apply the bigram on the text
-Calculate the word frequency (bi-gram frequency) of the words (bi-grams)
-Choose top five bi-grams that has been repeated most
-Go through the original text that you had in the file
-Find all the sentences with those most repeated bi-grams
-Extract those sentences and concatenate

Approach:

First, we need to open the input file and read it. Then lemmatization will be done followed by bigram. Then we will use counter to find the frequency of bigrams. After finding the top 5 bigrams, we will look for the most frequent bigram in the input file. Finally, those sentences would be concatenated and printed.

Code for importing the libraries

```
#Importing the libraries
from nltk.corpus import wordnet as w

from nltk.stem import LancasterStemmer , WordNetLemmatizer
from nltk import pos_tag,ne_chunk, ngrams
from collections import Counter
import urllib.request
from bs4 import BeautifulSoup
from urllib import request
import requests
import pandas as pd
import os
import nltk
import collections
import matplotlib.pyplot as plt

from nltk.book import *

from nltk.tokenize import word_tokenize,sent_tokenize,wordpunct_tokenize
```

Reading the input file

```
#Input file
raw = open('ebook.txt')
f=raw.read()# Reading the file
```

Applying Lemmatization on the words

```
#Applying Lemmatization on the words
words=[]
for line in raw:
    words.extend(line.strip().split())
l=WordNetLemmatizer()
lemmatize=map(l.lemmatize,words)
for i in lemmatize:
    print (i,)
```

Code for Applying bigram to text

```
token = nltk.word_tokenize(f)
```

Code to calculate the word frequency (bi-gram frequency) of the words(bi-grams)

```
bigrams = ngrams(token,2)
print (Counter(bigrams))
```

Code to choose top five bi-grams that has been repeated most

```
a=Counter(ngrams(token,2)).most_common(5)
bigrams.plot(10)
print (a)
```

The below code is to find all the sentences in the file which contains the most repeated bigrams and then concatenating them and printing.

```
b=[]
for i in range(len(a)):
    b.append(a[i][0])
print (b)
c=[]
for j in range(len(b)):
    #print(b[j][0]+" "+b[j][1])
    c.append(b[j][0]+" "+b[j][1])
#print (c)
new_line=""
for line in raw:
    #print (line)
    for bgrm in c:
        if bgrm in line:
            print (line) #printing all the lines which has the most repeated bi-grams
            new_line=new_line+line #Concatenating the lines which has the most repeated bi-grams
print(new_line) #printing the concatenated lines with most repeated bi-grams
```

Below is the full code

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  5 18:19:59 2018

@author: rabhishe
"""
#Importing the libraries
from nltk.corpus import wordnet as w

from nltk.stem import LancasterStemmer , WordNetLemmatizer
from nltk import pos_tag,ne_chunk, ngrams
from collections import Counter
import urllib.request
from bs4 import BeautifulSoup
from urllib import request
import requests
import pandas as pd
import os
import nltk
import collections
import matplotlib.pyplot as plt

from nltk.book import *




#Input file
raw = open('ebook.txt')
f=raw.read()# Reading the file

#Applying Lemmatization on the words
from nltk.tokenize import word_tokenize,sent_tokenize,wordpunct_tokenize
words=[]
for line in raw:
    words.extend(line.strip().split())
l=WordNetLemmatizer()
lemmatize=map(l.lemmatize,words)
```

```
for i in lemmatize:
    print (i,)

#Applying bigram to text
token = nltk.word_tokenize(f)

#Calculating the word frequency (bi-gram frequency) of the words(bi-grams)
bigrams = ngrams(token,2)
print (Counter(bigrams))

#Choosing top five bi-gramsthat has been repeated most
a=Counter(ngrams(token,2)).most_common(5)
bigrams.plot(10)
print (a)

b=[]
for i in range(len(a)):
    b.append(a[i][0])
print (b)
c=[]
for j in range(len(b)):
    #print(b[j][0]+" "+b[j][1])
    c.append(b[j][0]+" "+b[j][1])
#print (c)
new_line=""
for line in raw:
    #print (line)
    for bgrm in c:
        if bgrm in line:
            print (line) #printing all the lines which has the most repeated bi-grams
            new_line=new_line+line #Concatenating the lines which has the most
repeated bi-grams
print(new_line) #printing the concatenated lines with most repeated bi-grams
```

Simulation

```
 ['3', 'May', '.', 'Bistritz', '.', '_', '--', 'Left', 'Munich', 'at',
'8', ':', '35', 'P', '.', 'M', '.,', 'on', '1st', 'May', ',',
'arriving', 'at', 'Vienna', 'early', 'next', 'morning', ';', 'should',
'have', 'arrived', 'at', '6', ':', '46', ',', 'but', 'train', 'wa',
'an', 'hour', 'late', '.', 'Buda', '-', 'Pesth', 'seems', 'a',
'wonderful', 'place', ',', 'from', 'the', 'glimpse', 'which', 'I',
'got', 'of', 'it', 'from', 'the', 'train', 'and', 'the', 'little', 'I',
'could', 'walk', 'through', 'the', 'street', '.', 'I', 'feared', 'to',
'go', 'very', 'far', 'from', 'the', 'station', ',', 'a', 'we', 'had',
'arrived', 'late', 'and', 'would', 'start', 'a', 'near', 'the',
'correct', 'time', 'a', 'possible', '.', 'The', 'impression', 'I',
'had', 'wa', 'that', 'we', 'were', 'leaving', 'the', 'West', 'and',
'entering', 'the', 'East', ';', 'the', 'most', 'western', 'of',
'splendid', 'bridge', 'over', 'the', 'Danube', ',', 'which', 'is',
'here', 'of', 'noble', 'width', 'and', 'depth', ',', 'took', 'u',
'among', 'the', 'tradition', 'of', 'Turkish', 'rule', '.']
```

Bigrams

```
 [('3', 'May'), ('May', '.'), ('.', 'Bistritz'), ('Bistritz', '.'),
('.', '_'), ('_', '--'), ('--', 'Left'), ('Left', 'Munich'), ('Munich',
'at'), ('at', '8'), ('8', ':'), (':', '35'), ('35', 'P'), ('P', '.'),
('.', 'M'), ('M', '.,'), ('.,', 'on'), ('on', '1st'), ('1st', 'May'),
('May', ','), (',', 'arriving'), ('arriving', 'at'), ('at', 'Vienna'),
('Vienna', 'early'), ('early', 'next'), ('next', 'morning'),
('morning', ';'), (';', 'should'), ('should', 'have'), ('have',
'arrived'), ('arrived', 'at'), ('at', '6'), ('6', ':'), (':', '46'),
('46', ','), (',', 'but'), ('but', 'train'), ('train', 'was'), ('was',
'an'), ('an', 'hour'), ('hour', 'late'), ('late', '.'), ('.', 'Buda'),
('Buda', '-'), ('-', 'Pesth'), ('Pesth', 'seems'), ('seems', 'a'),
('a', 'wonderful'), ('wonderful', 'place'), ('place', ','), (',',
'from'), ('from', 'the'), ('the', 'glimpse'), ('glimpse', 'which'),
('which', 'I'), ('I', 'got'), ('got', 'of'), ('of', 'it'), ('it',
'from'), ('from', 'the'), ('the', 'train'), ('train', 'and'), ('and',
'the'), ('the', 'little'), ('little', 'I'), ('I', 'could'), ('could',
'walk'), ('walk', 'through'), ('through', 'the'), ('the', 'streets'),
('streets', '.'), ('.', 'I'), ('I', 'feared'), ('feared', 'to'), ('to',
'go'), ('go', 'very'), ('very', 'far'), ('far', 'from'), ('from',
'the'), ('the', 'station'), ('station', ','), (',', 'as'), ('as',
'we'), ('we', 'had'), ('had', 'arrived'), ('arrived', 'late'), ('late',
'and'), ('and', 'would'), ('would', 'start'), ('start', 'as'), ('as',
'near'), ('near', 'the'), ('the', 'correct'), ('correct', 'time'),
('time', 'as'), ('as', 'possible'), ('possible', '.'), ('.', 'The'),
('The', 'impression'), ('impression', 'I'), ('I', 'had'), ('had',
'was'), ('was', 'that'), ('that', 'we'), ('we', 'were'), ('were',
'leaving'), ('leaving', 'the'), ('the', 'West'), ('West', 'and'),
('and', 'entering'), ('entering', 'the'), ('the', 'East'), ('East',
';'), (';', 'the'), ('the', 'most'), ('most', 'western'), ('western',
'of'), ('of', 'splendid'), ('splendid', 'bridges'), ('bridges',
'over'), ('over', 'the'), ('the', 'Danube'), ('Danube', ','), (',',
'which'), ('which', 'is'), ('is', 'here'), ('here', 'of'), ('of',
'noble'), ('noble', 'width'), ('width', 'and'), ('and', 'depth'),
('depth', ','), (',', 'took'), ('took', 'us'), ('us', 'among'),
('among', 'the'), ('the', 'traditions'), ('traditions', 'of'), ('of',
'Turkish'), ('Turkish', 'rule'), ('rule', '.')]
```

Calculating bi-frequency of the bi-grams

```
 Counter({('from', 'the'): 3, ('is', 'here'): 1, ('bridges', 'over'):
1, ('at', 'Vienna'): 1, ('depth', ','): 1, ('but', 'train'): 1, ('and',
'the'): 1, ('May', ','): 1, ('a', 'wonderful'): 1, ('the', 'streets'):
1, ('over', 'the'): 1, ('as', 'possible'): 1, (',', 'but'): 1, ('of',
'Turkish'): 1, ('to', 'go'): 1, ('the', 'correct'): 1, (';', 'should'):
1, ('walk', 'through'): 1, ('1st', 'May'): 1, ('--', 'Left'): 1,
('correct', 'time'): 1, ('the', 'train'): 1, ('the', 'West'): 1, ('.',
'M'): 1, (';', 'the'): 1, ('start', 'as'): 1, ('would', 'start'): 1,
('through', 'the'): 1, ('most', 'western'): 1, ('arrived', 'late'): 1,
('at', '8'): 1, ('among', 'the'): 1, ('.', 'I'): 1, ('little', 'I'): 1,
('splendid', 'bridges'): 1, ('streets', '.'): 1, ('Bistritz', '.'): 1,
('possible', '.'): 1, ('was', 'that'): 1, ('.', 'Buda'): 1,
('arriving', 'at'): 1, ('3', 'May'): 1, ('Buda', '-'): 1,
('traditions', 'of'): 1, ('46', ','): 1, ('had', 'was'): 1, ('the',
'most'): 1, ('it', 'from'): 1, ('the', 'traditions'): 1, (':', '46'):
1, ('next', 'morning'): 1, ('which', 'I'): 1, ('Pesth', 'seems'): 1,
('could', 'walk'): 1, ('.,', 'on'): 1, ('that', 'we'): 1, ('us',
'among'): 1, ('place', ','): 1, ('6', ':'): 1, ('I', 'feared'): 1,
('was', 'an'): 1, ('which', 'is'): 1, ('arrived', 'at'): 1, ('here',
'of'): 1, ('.', 'The'): 1, ('and', 'depth'): 1, ('late', 'and'): 1,
('.', '_'): 1, ('entering', 'the'): 1, ('very', 'far'): 1, ('the',
'Danube'): 1, ('wonderful', 'place'): 1, (',', 'which'): 1, ('rule',
'.'): 1, ('as', 'near'): 1, ('hour', 'late'): 1, ('glimpse', 'which'):
1, ('Munich', 'at'): 1, ('of', 'it'): 1, ('the', 'station'): 1, (',',
'from'): 1, ('West', 'and'): 1, ('the', 'East'): 1, ('time', 'as'): 1,
('.', 'Bistritz'): 1, ('-', 'Pesth'): 1, ('had', 'arrived'): 1,
('noble', 'width'): 1, (':', '35'): 1, ('we', 'were'): 1, ('took',
'us'): 1, ('we', 'had'): 1, ('leaving', 'the'): 1, ('should', 'have'):
1, ('of', 'noble'): 1, ('were', 'leaving'): 1, ('feared', 'to'): 1,
('and', 'entering'): 1, ('on', '1st'): 1, ('impression', 'I'): 1,
('Left', 'Munich'): 1, ('Vienna', 'early'): 1, ('I', 'could'): 1,
('train', 'and'): 1, ('late', '.'): 1, ('station', ','): 1, ('the',
'little'): 1, (',', 'arriving'): 1, ('near', 'the'): 1, ('an', 'hour'):
1, ('early', 'next'): 1, ('have', 'arrived'): 1, ('go', 'very'): 1,
('train', 'was'): 1, ('P', '.'): 1, ('width', 'and'): 1, ('at', '6'):
1, ('got', 'of'): 1, (',', 'as'): 1, ('as', 'we'): 1, ('Turkish',
'rule'): 1, ('the', 'glimpse'): 1, ('western', 'of'): 1, ('M', '.,'):
1, ('Danube', ','): 1, ('and', 'would'): 1, (',', 'took'): 1, ('far',
'from'): 1, ('8', ':'): 1, ('_', '--'): 1, ('35', 'P'): 1, ('I',
'had'): 1, ('East', ';'): 1, ('of', 'splendid'): 1, ('I', 'got'): 1,
('May', '.'): 1, ('The', 'impression'): 1, ('morning', ';'): 1,
('seems', 'a'): 1})
```

Most frequent bi-grams

```
[(('from', 'the'), 3), (('is', 'here'), 1), (('bridges', 'over'), 1),
(('at', 'Vienna'), 1), (('depth', ','), 1)]
```

Concatenated most frequent bigrams

```
Buda-Pesth seems a wonderful place, from the glimpse which I
got of it from the train and the little I could walk through the
streets.
```

Question 4)

Report your views on the k nearest neighbor algorithm when we change the K how it will affect the accuracy. Provide a good justification about the changes of the accuracy when we change the amount of K.
For example: compare the accuracy when K=1 and K is a big number like 50, why the accuracy will change

Approach:

For this question we first need to write a code for k nearest neighbor. Then we need to test the accuracy by changing the values of k. I have tested the accuracy for both without and with cross validation. For without cross validation the k range is from 1 to 150 whereas for cross validation it ranges from 1 to 120. Each part of the code is explained below.

Code to import all libraries

```
#Importing the libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
from sklearn.cross_validation import train_test_split
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn import datasets, metrics
```

For this we need to use iris data set. Below code is used to import the iris dataset

```
#Using Iris dataset
irisdataset=datasets.load_iris()
X_set=irisdataset.data
y_set=irisdataset.target
```

Below code is used to find the accuracy score for k neighbors without cross validation with k ranging from 1 to 150

```
#K Neighbors without Cross Validation
k_neighbor_range=range(1,151)
score=[]
for K in k_neighbor_range:
    K_model_without_cross_validation=KNeighborsClassifier(n_neighbors=K)
    K_model_without_cross_validation.fit(X_set,y_set)
    print("Accuracy for",K,"=",K_model_without_cross_validation.score(X_set,y_set))
    score.append(K_model_without_cross_validation.score(X_set,y_set))

import matplotlib.pyplot as plt

plt.plot(k_neighbor_range,score)
plt.xlabel("K Value (Without cross Validation)")
plt.ylabel("Accuracy")
plt.show()
```

Below code is used to find the accuracy score for k neighbors without cross validation with k ranging from 1 to 120

```
#K Neighbors with Cross Validation
X_training,X_testing,y_training,y_testing=train_test_split(X_set,y_set,test_size=0.2)

#testing and training data
model= KNeighborsClassifier(n_neighbors=5)
model.fit(X_training,y_training)

y_pred=model.predict(X_testing)


k_neighbor_range=range(1,121)
score=[]

for K in k_neighbor_range:
    k_nearest_n=KNeighborsClassifier(n_neighbors=K)
    k_nearest_n.fit(X_training,y_training)
    y_prediction=k_nearest_n.predict(X_testing)
    score.append(metrics.accuracy_score(y_testing,y_prediction))

import matplotlib.pyplot as plt

plt.plot(k_neighbor_range,score)
plt.xlabel("K Value (Cross Validation)")
plt.ylabel("Accuracy")
plt.show()
```

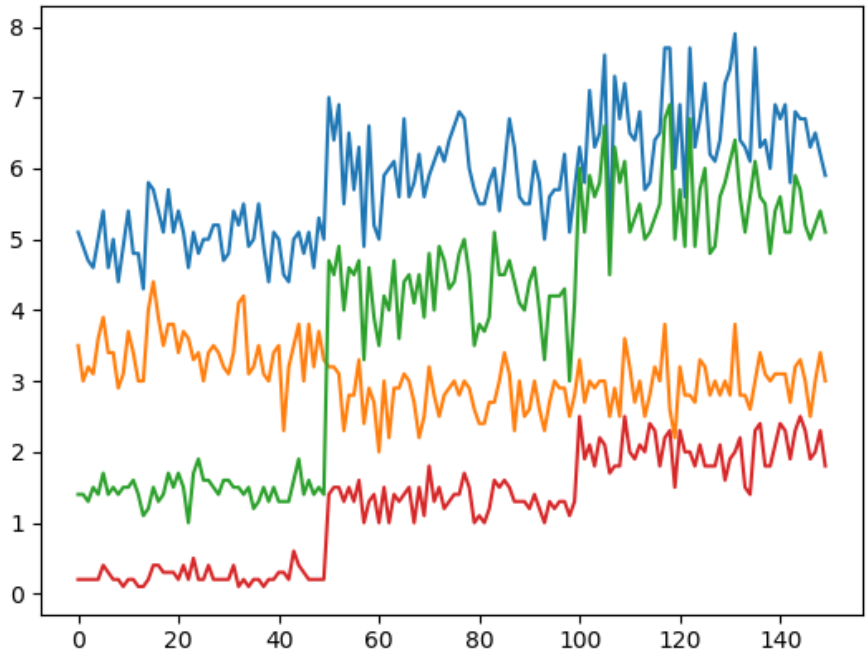
Below is the full code

```
# -*- coding: utf-8 -*-
"""
Created on Tue Mar  6 16:35:32 2018

@author: rabhishe
"""
#Importing the libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
from sklearn.cross_validation import train_test_split
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn import datasets, metrics



#Using Iris dataset
irisdataset=datasets.load_iris()
X_set=irisdataset.data
y_set=irisdataset.target



#K Neighbors without Cross Validation
k_neighbor_range=range(1,151)
score=[]
for K in k_neighbor_range:
    K_model_without_cross_validation=KNeighborsClassifier(n_neighbors=K)
    K_model_without_cross_validation.fit(X_set,y_set)
    print("Accuracy
for",K,"=",K_model_without_cross_validation.score(X_set,y_set))
    score.append(K_model_without_cross_validation.score(X_set,y_set))

import matplotlib.pyplot as plt

plt.plot(k_neighbor_range,score)
plt.xlabel("K Value (Without cross Validation)")
plt.ylabel("Accuracy")
plt.show()

#K Neighbors with Cross Validation
```

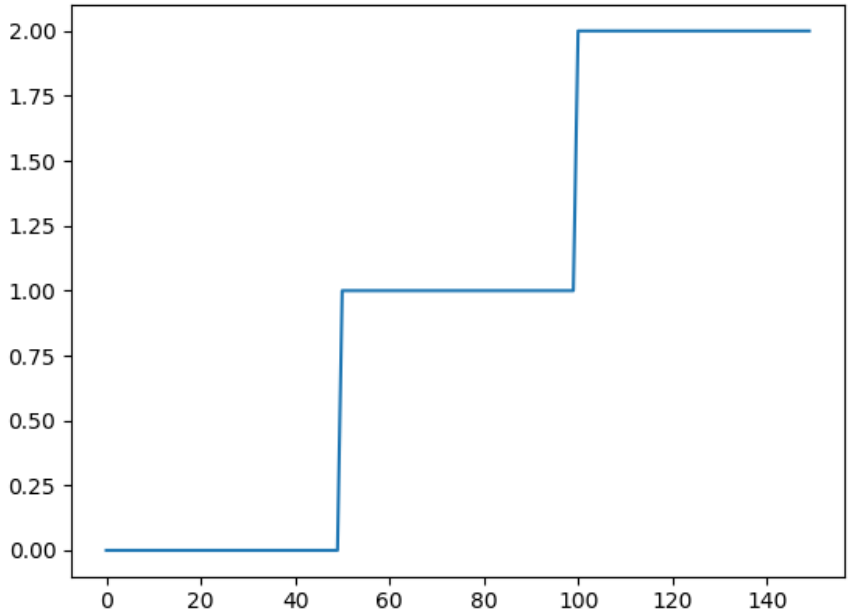
```
X_training,X_testing,y_training,y_testing=train_test_split(X_set,y_set,test_size=0.
2)

#testing and training data
model= KNeighborsClassifier(n_neighbors=5)
model.fit(X_training,y_training)

y_pred=model.predict(X_testing)


k_neighbor_range=range(1,121)
score=[]

for K in k_neighbor_range:
    k_nearest_n=KNeighborsClassifier(n_neighbors=K)
    k_nearest_n.fit(X_training,y_training)
    y_prediction=k_nearest_n.predict(X_testing)
    score.append(metrics.accuracy_score(y_testing,y_prediction))

import matplotlib.pyplot as plt

plt.plot(k_neighbor_range,score)
plt.xlabel("K Value (Cross Validation)")
plt.ylabel("Accuracy")
plt.show()
```

Simulation

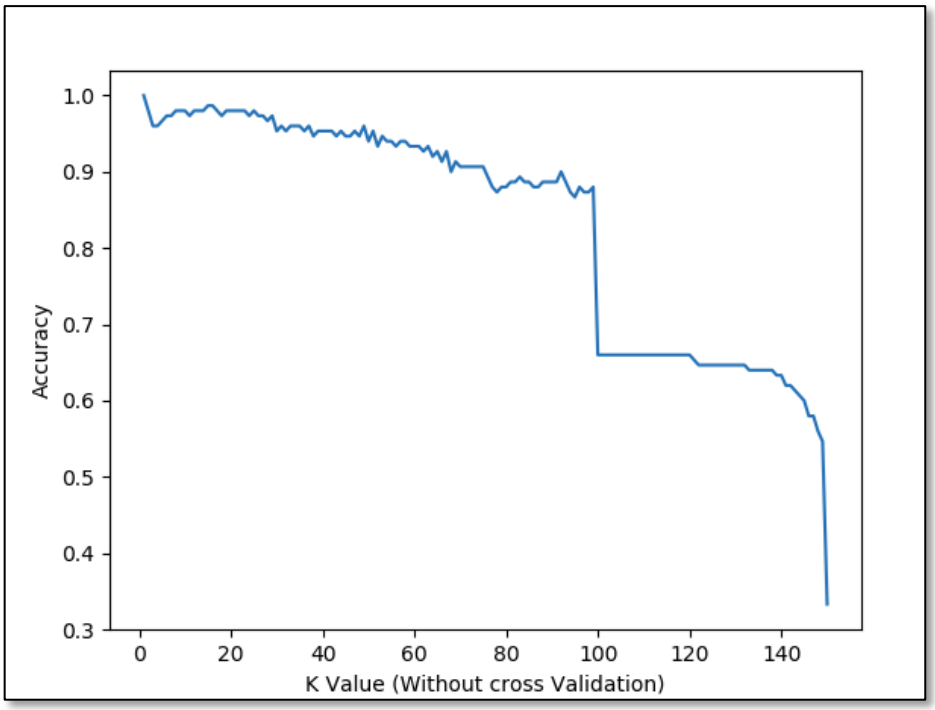
Plotting the X_set



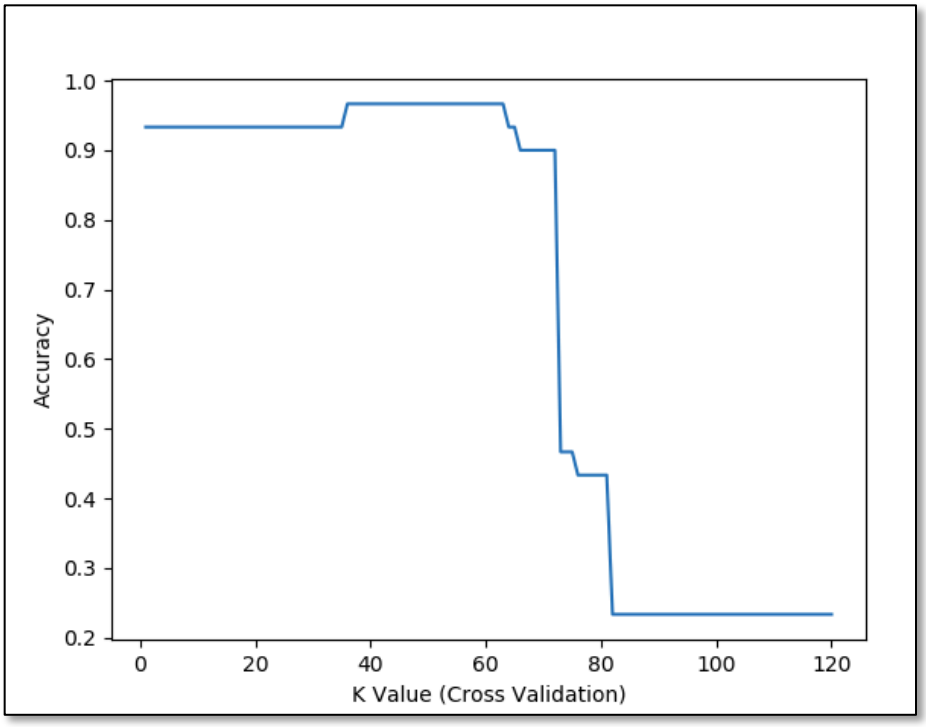
Plotting the Y_set



Plotting the accuracy for K (without cross validation)



Plotting the accuracy for K (cross validation)



As we can see from the above results as the value of K increases the accuracy seems to lower. This is because with lower value of K, the region of prediction is

restrained which makes the classifier look at the overall distribution blindly. Lower the value of K, the more flexible the fit, hence low bias but high variance resulting in more jagged boundary.
As the value of K is increased, more voters are involved for each prediction resulting in more resiliency to the outliers. So, the larger the value of k, smoother would be the decision boundaries resulting in lower variance and increased bias.

DEPLOYMENT

For deployment of the code, the following process should be followed:

- Install python on your system. It can be download from https://www.python.org/downloads/
- Download the code from my git. (https://github.com/ra5xc/CSEE5590/tree/master/Assignment%203)
- Import libraries
- Execute the code using python
- I would highly suggest reading the report before executing the code to better understand the functionality of the codes. The report can be downloaded from my git as well.

LIMITATION

Problem 1: The disadvantage of using LDA is that it can give predicted probabilities lying outside the range of 0-1. Which is why I used logistic regression in problem 1.
Problem 2: Using SVM slows down the speed and puts a limitation on both the testing and training data. For large scale tasks the algorithmic complexity and memory requirement of svm increases drastically.
Problem 3: Lemmatization may sometimes give false positive hence causing ambiguity. Further it might not be applicable to all languages.
Problem 4: K nearest neighbor has limitations on the problem size and memory. The computation cost is also high.

REFERENCES

https://www.superdatascience.com (code templates have been referred from here)
https://github.com