Sravani Konujula

Class ID-23

16230172

Lab Assignment-3

**Objective:**

The purpose of this assignment is

* Making a prediction on the dataset using LDA
* Applying Linear and RBF kernel on SVM
* Applying Lemmatization and bigram on text
* Finding the results of K nearest neighbors with K values.

**Features:**

* Graph has been displayed for the considered dataset using LDA implementation
* Using SVM implementation with the scikit-learn and applying SVC with linear kernel and RBF kernel
* By using Lemmatization and bigram frequency on words to display top 5 words

And concatenate the sentences with high frequency words

* To display the results of k nearest neighbor algorithm for k=1 and k=50 values.

**Configuration:**

Pycharm

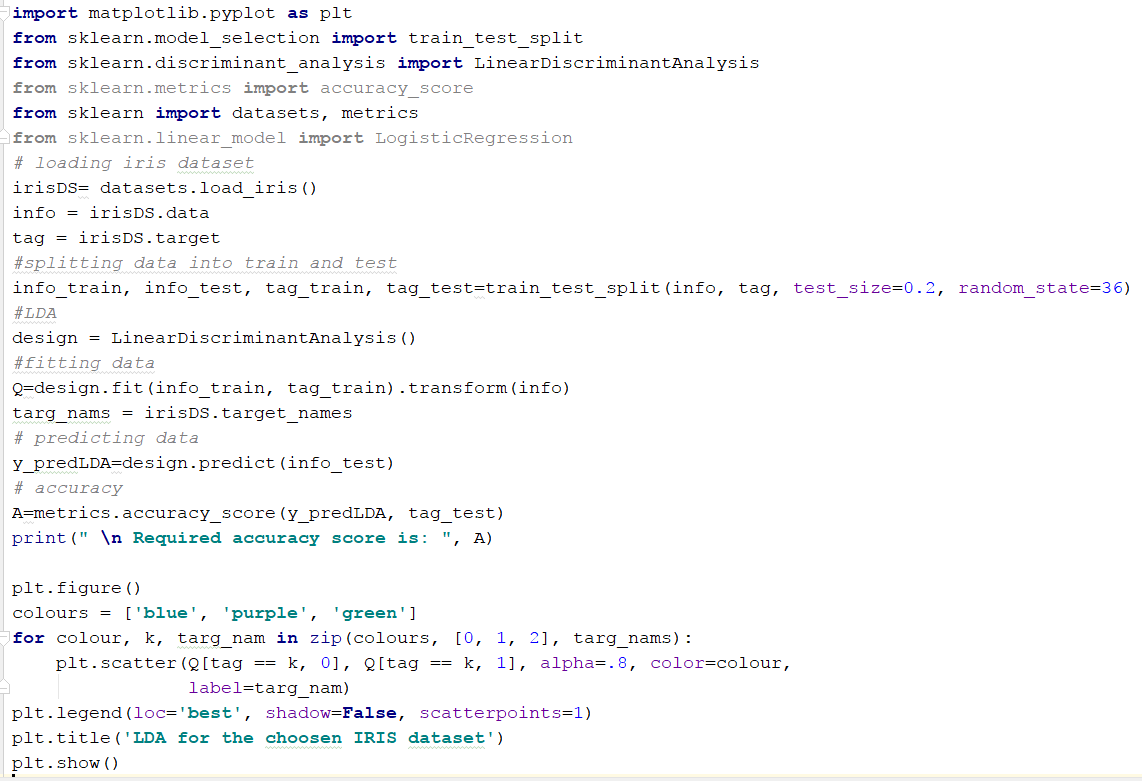
Python: 2.7.13

**Output Screens**

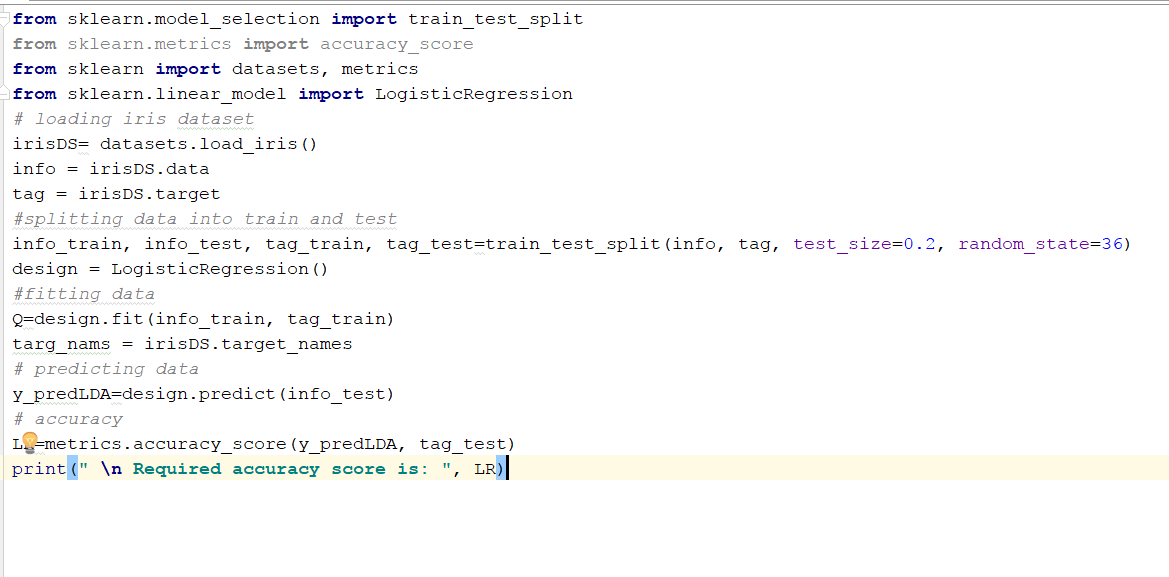
**Question1:**

**Source Code:**

**LDA:**

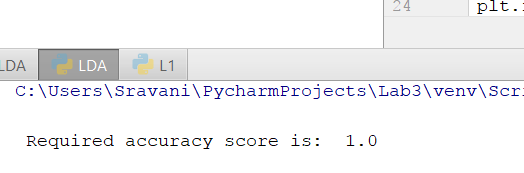


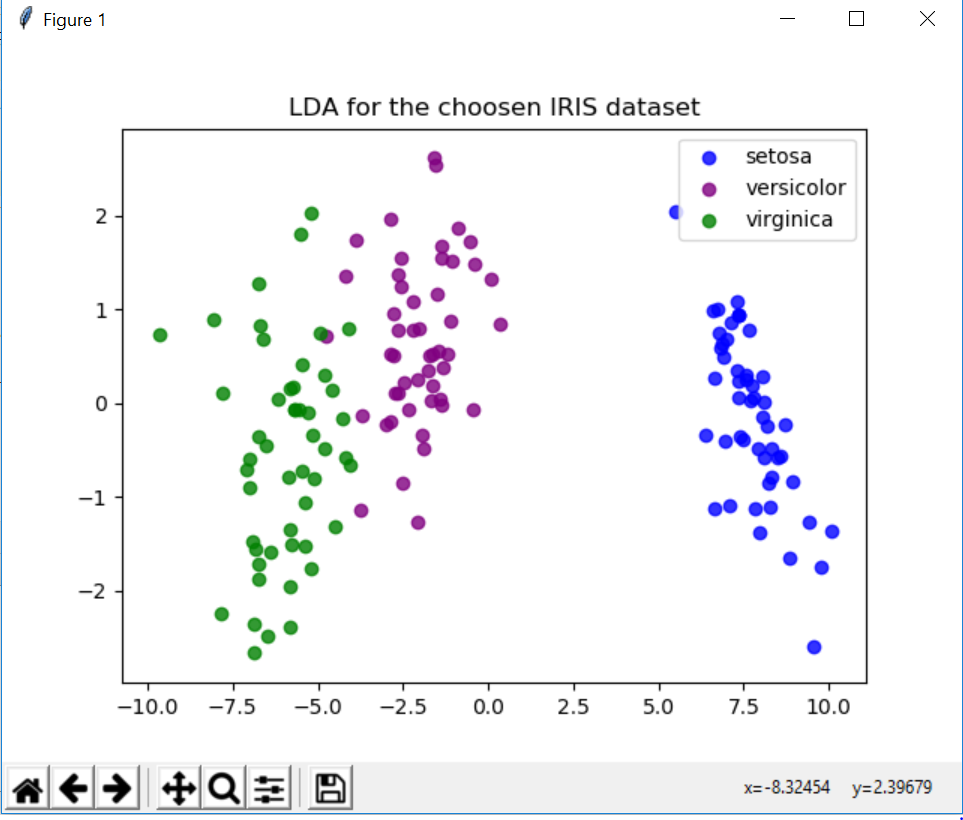
**LR:**



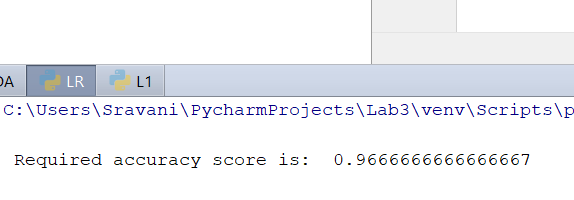
**Output Screen:**

**LDA output:**



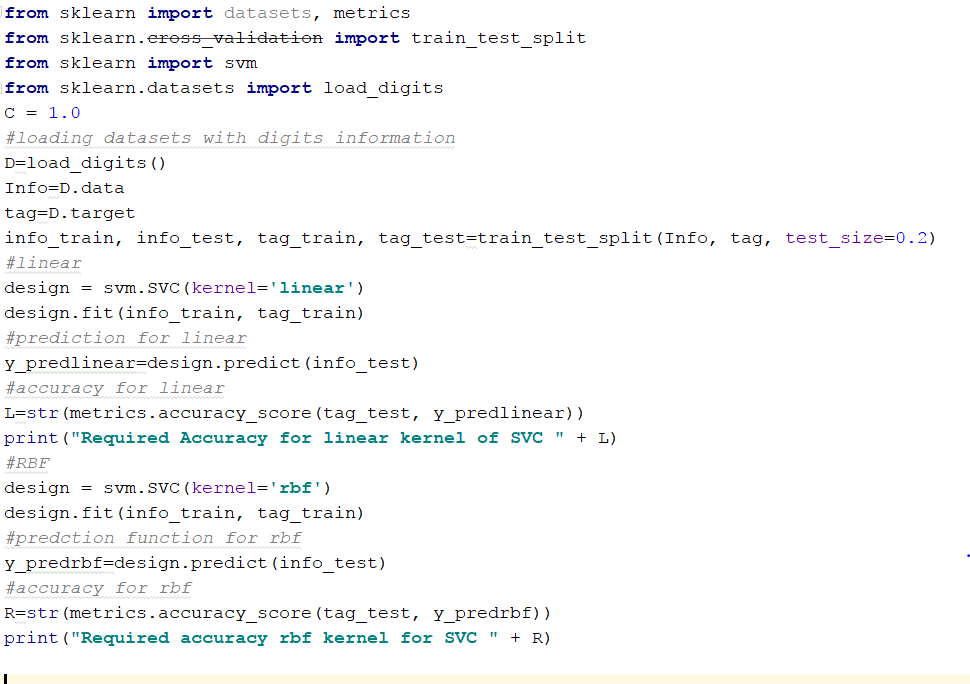


**LR output:**

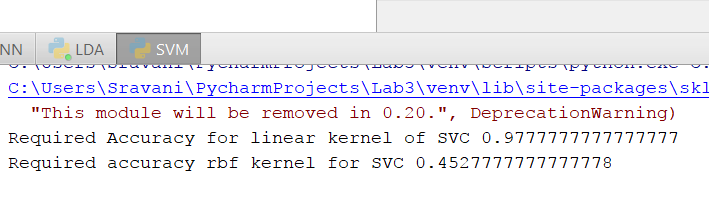


**Question2:**

**Source Code2:**



**Output Screen2:**

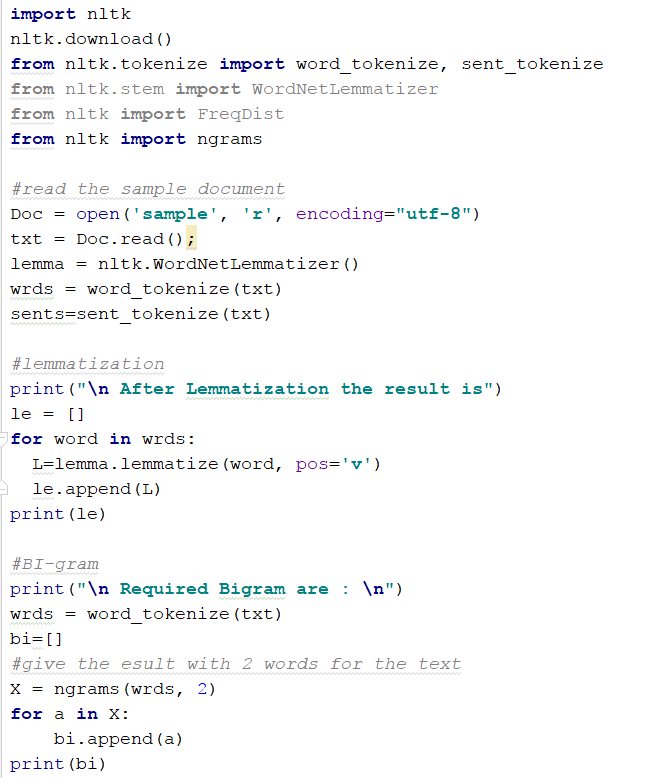


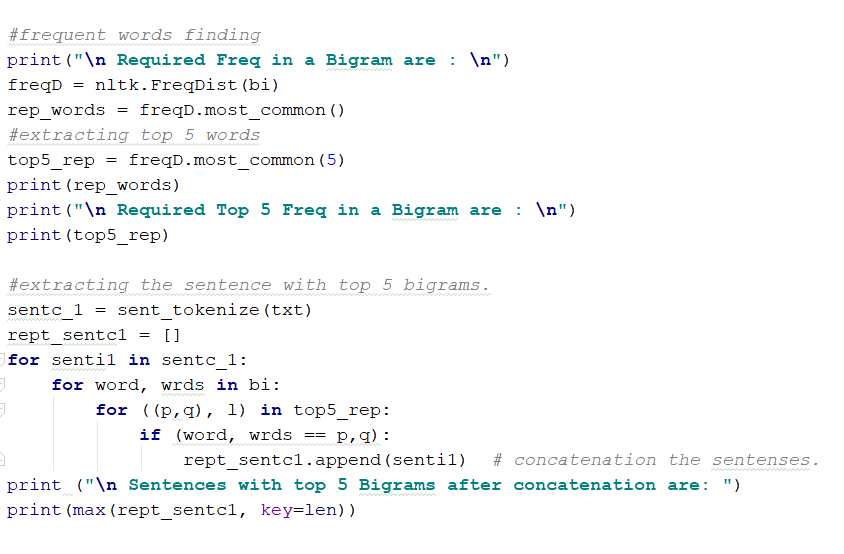
**Report:**

Obtained accuracies for both linear and RBF(non linear) kernels is 0.977777777 and 0.45277777778 respectively. It is mostly based on the features if linear is having high then it is preferred comparative to RBF and vice versa. In my view Linear is having high so considered linear is best.

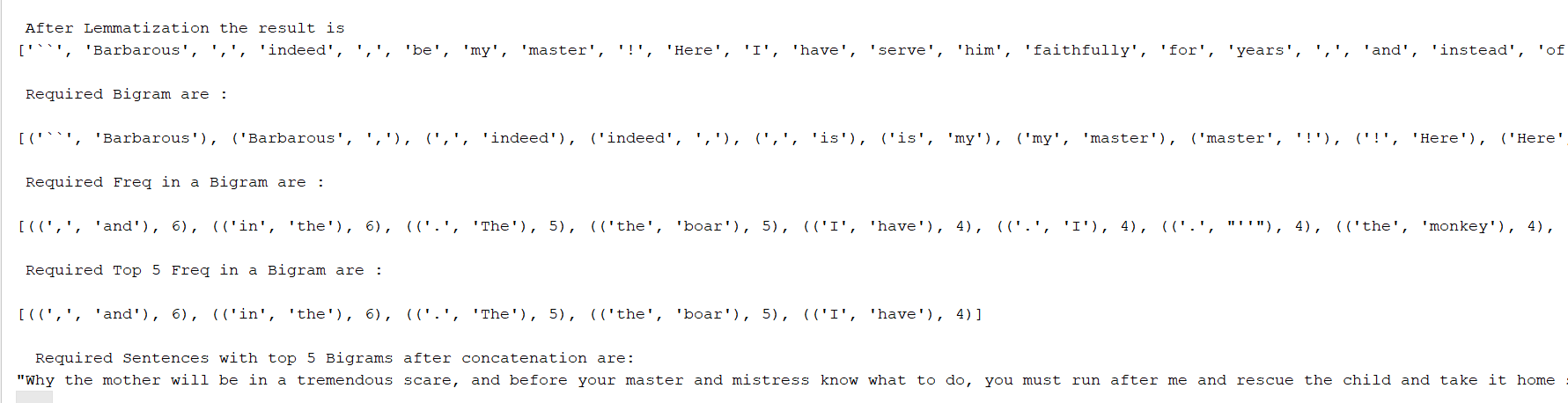
**Question3:**

**Source Code3:**



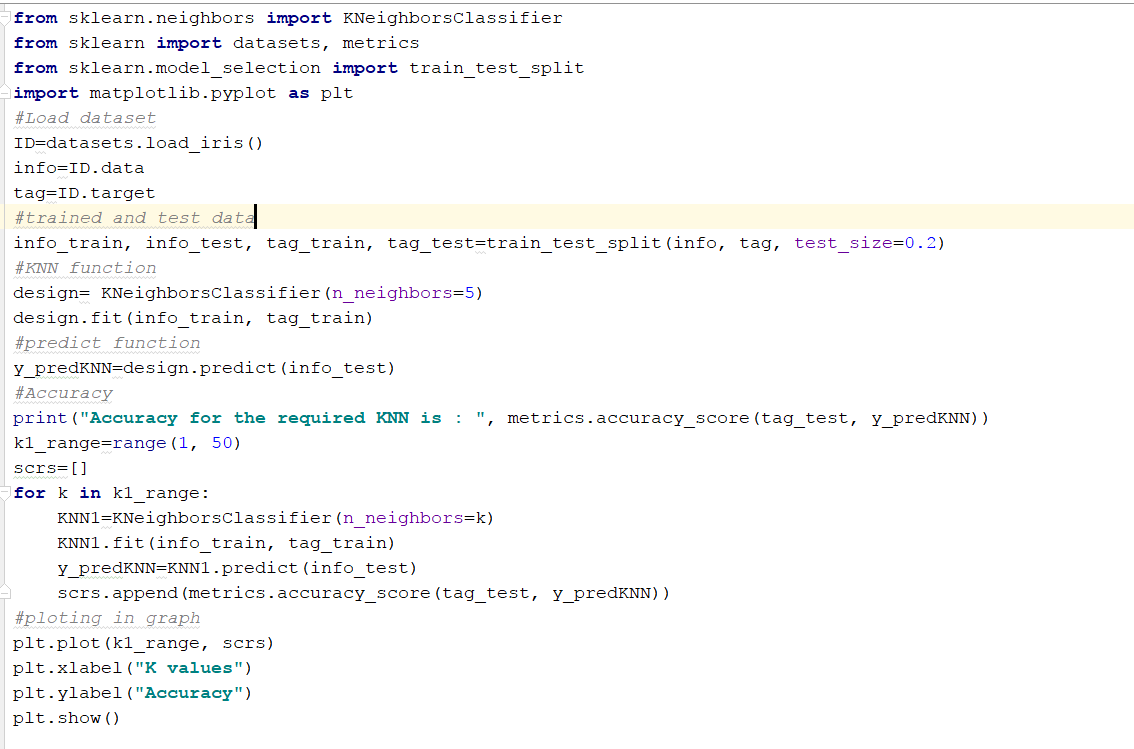


**Output Screen:**



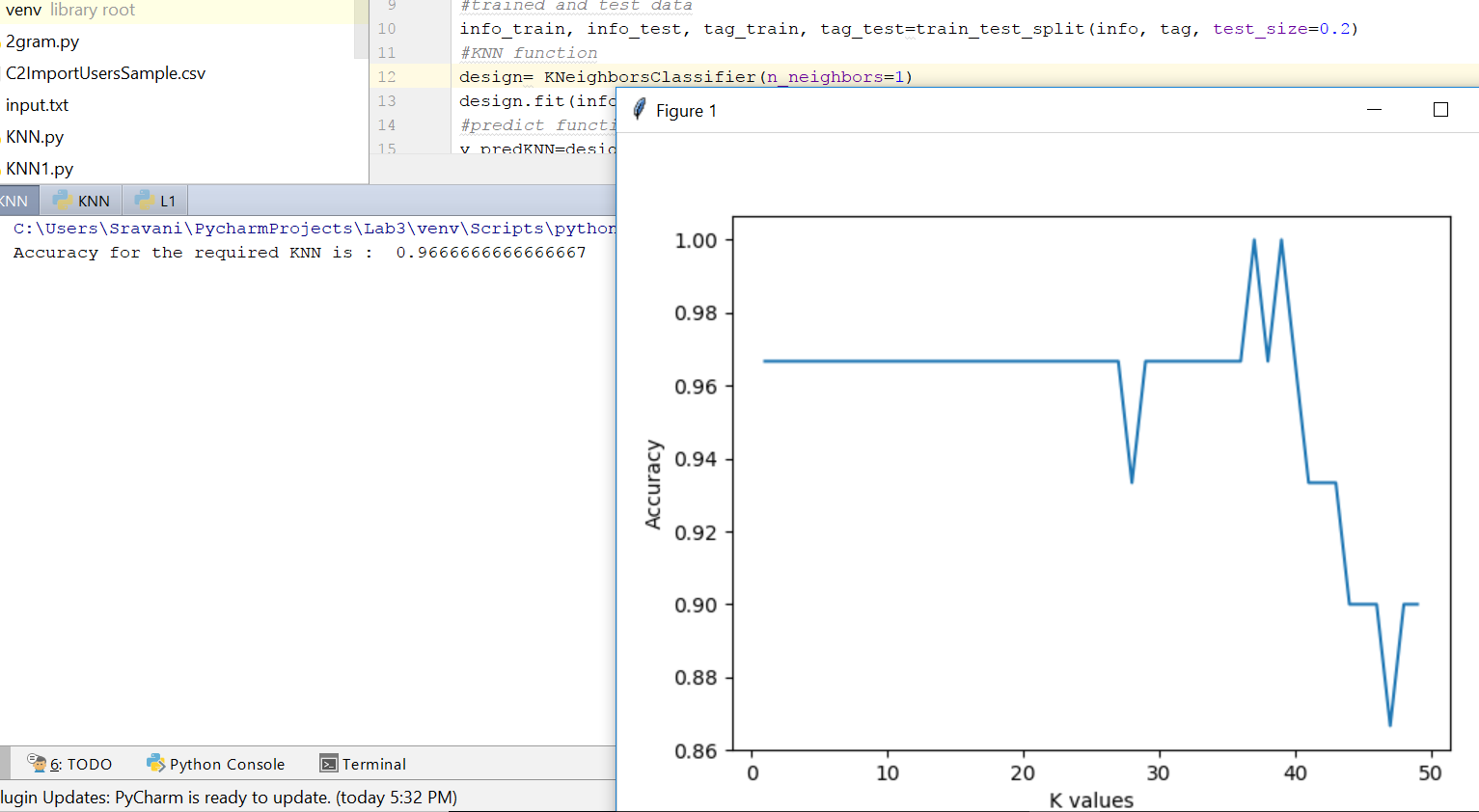
**Question4:**

**Source Code4:**

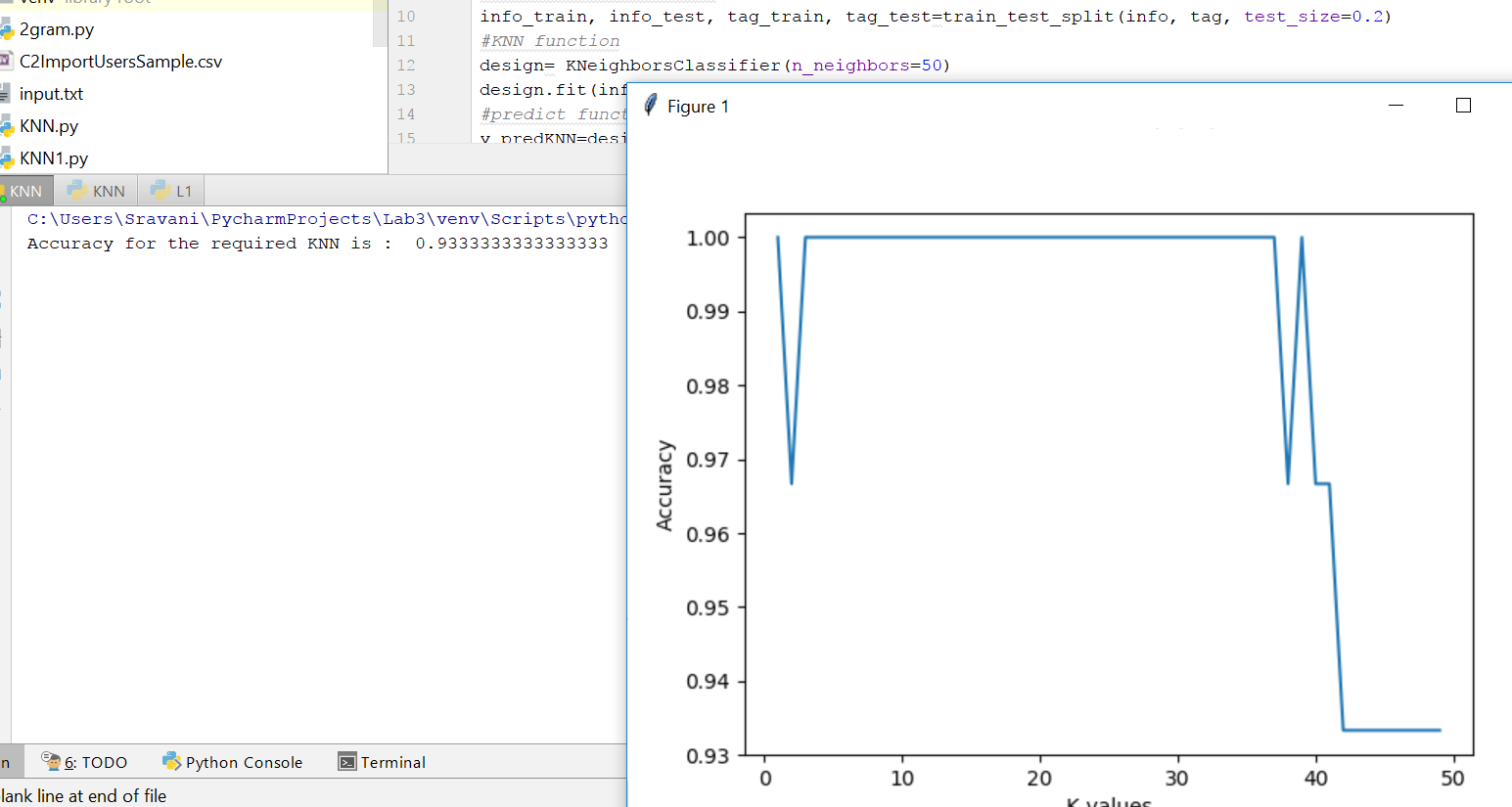


**Output Screen:**

For K=1



**For K=50**



**Code Snippet1:**

**Linear Discriminant Analysis and its Accuracy score with plotting it in graph.**

**import** matplotlib.pyplot **as** plt  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.discriminant\_analysis **import** LinearDiscriminantAnalysis  
**from** sklearn.metrics **import** accuracy\_score  
**from** sklearn **import** datasets, metrics  
**from** sklearn.linear\_model **import** LogisticRegression  
*# load dataset*irisDS= datasets.load\_iris()  
info = irisDS.data  
tag = irisDS.target  
info\_train, info\_test, tag\_train, tag\_test=train\_test\_split(info, tag, test\_size=0.2, random\_state=36)  
*#LDA*design = LinearDiscriminantAnalysis()  
*#fitting data*Q=design.fit(info\_train, tag\_train).transform(info)  
targ\_nams = irisDS.target\_names  
*# predicting data*y\_predLDA=design.predict(info\_test)  
*# accuracy*A=metrics.accuracy\_score(y\_predLDA, tag\_test)  
print(**" \n Required accuracy score is: "**, A)  
  
plt.figure()  
colours = [**'orange'**, **'pink'**, **'violet'**]  
**for** colour, k, targ\_nam **in** zip(colours, [0, 1, 2], targ\_nams):  
 plt.scatter(Q[tag == k, 0], Q[tag == k, 1], alpha=.8, color=colour,  
 label=targ\_nam)  
plt.legend(loc=**'best'**, shadow=**False**, scatterpoints=1)  
plt.title(**'LDA for the choosen IRIS dataset'**)  
plt.show()

**For Logistic Regression:**

**Logistic regression and its accuracy score.**

**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.metrics **import** accuracy\_score  
**from** sklearn **import** datasets, metrics  
**from** sklearn.linear\_model **import** LogisticRegression  
*# loading dataset*irisDS= datasets.load\_iris()  
info = irisDS.data  
tag = irisDS.target  
info\_train, info\_test, tag\_train, tag\_test=train\_test\_split(info, tag, test\_size=0.2, random\_state=36)  
design = LogisticRegression()  
*#fitting data*Q=design.fit(info\_train, tag\_train)  
targ\_nams = irisDS.target\_names  
*# predicting data*y\_predLDA=design.predict(info\_test)  
*# accuracy*LR=metrics.accuracy\_score(y\_predLDA, tag\_test)  
print(**" \n Required accuracy score is: "**, LR)

**Report:**

LDA in which more than 2 classes are taken where accuracy value is more compared to LR in which only 2 classes is taken.

**Snippet2:**

**SVM kernal (Linear and RBF)**

**from** sklearn **import** datasets, metrics  
**from** sklearn.cross\_validation **import** train\_test\_split  
**from** sklearn **import** svm  
**from** sklearn.datasets **import** load\_digits  
C = 1.0  
*#loading datasets with digits information*D=load\_digits()  
Info=D.data  
tag=D.target  
info\_train, info\_test, tag\_train, tag\_test=train\_test\_split(Info, tag, test\_size=0.2)  
*#linear*design = svm.SVC(kernel=**'linear'**)  
design.fit(info\_train, tag\_train)  
*#prediction for linear*tag\_predlinear=design.predict(info\_test)  
*#linear kernal with accuracy*L=str(metrics.accuracy\_score(tag\_test, tag\_predlinear))  
print(**"Required Accuracy for linear kernel of SVC "** + L)  
*#RBF*design = svm.SVC(kernel=**'rbf'**)  
design.fit(info\_train, tag\_train)  
*#predction function for rbf*tag\_predrbf=design.predict(info\_test)  
*#accuracy for rbf*R=str(metrics.accuracy\_score(tag\_test, tag\_predrbf))  
print(**"Required accuracy rbf kernel for SVC "** + R)

**Report:**

**Code Snippet3:**

Using Natural Language toolkit worked out on the tasks lemmatization, bigram, frequency for the bigram words and top five bigram words from them and also extracting sentences in which top five words are and concatenating them.

**import** nltk  
nltk.download()  
**from** nltk.tokenize **import** word\_tokenize, sent\_tokenize  
**from** nltk.stem **import** WordNetLemmatizer  
**from** nltk **import** FreqDist  
**from** nltk **import** ngrams  
  
*#read the sample document*Doc = open(**'sample'**, **'r'**, encoding=**"utf-8"**)  
txt = Doc.read();  
lemma = nltk.WordNetLemmatizer()  
wrds = word\_tokenize(txt)  
sents=sent\_tokenize(txt)  
  
*#lemmatization*print(**"\n After Lemmatization the result is"**)  
le = []  
**for** word **in** wrds:  
 L=lemma.lemmatize(word, pos=**'v'**)  
 le.append(L)  
print(le)  
  
*#BI-gram*print(**"\n Required Bigram are : \n"**)  
wrds = word\_tokenize(txt)  
bi=[]  
*#give the esult with 2 words for the text*X = ngrams(wrds, 2)  
**for** a **in** X:  
 bi.append(a)  
print(bi)  
  
*#frequent words finding*print(**"\n Required Freq in a Bigram are : \n"**)  
freqD = nltk.FreqDist(bi)  
rep\_words = freqD.most\_common()  
*#extracting top 5 words*top5\_rep = freqD.most\_common(5)  
print(rep\_words)  
print(**"\n Required Top 5 Freq in a Bigram are : \n"**)  
print(top5\_rep)  
  
*#extracting the sentence with top 5 bigrams.*sentc\_1 = sent\_tokenize(txt)  
rept\_sentc1 = []  
**for** senti1 **in** sentc\_1:  
 **for** word, wrds **in** bi:  
 **for** ((p,q), l) **in** top5\_rep:  
 **if** (word, wrds == p,q):  
 rept\_sentc1.append(senti1) *# concatenation the sentenses.*print (**"\n Sentences with top 5 Bigrams after concatenation are: "**)  
print(max(rept\_sentc1, key=len))

**Code Snippet4:**

**KNN with accuracy score and variation between K values for 1 and 50.**

**from** sklearn.neighbors **import** KNeighborsClassifier  
**from** sklearn **import** datasets, metrics  
**from** sklearn.model\_selection **import** train\_test\_split  
**import** matplotlib.pyplot **as** plt  
*#Load dataset*ID=datasets.load\_iris()  
info=ID.data  
tag=ID.target  
*#trained and test data*info\_train, info\_test, tag\_train, tag\_test=train\_test\_split(info, tag, test\_size=0.2)  
*#KNN function*design= KNeighborsClassifier(n\_neighbors=1)  
design.fit(info\_train, tag\_train)  
*#predict function*y\_predKNN=design.predict(info\_test)  
*#Accuracy*print(**"Accuracy for the required KNN is : "**, metrics.accuracy\_score(tag\_test, y\_predKNN))  
kn1\_range1=range(1, 50)  
ss1=[]  
**for** q **in** kn1\_range1:  
 KNN1=KNeighborsClassifier(n\_neighbors=q)  
 KNN1.fit(info\_train, tag\_train)  
 y\_predKNN=KNN1.predict(info\_test)  
 ss1.append(metrics.accuracy\_score(tag\_test, y\_predKNN))  
*#ploting in graph*plt.plot(kn1\_range1, ss1)  
plt.xlabel(**"K values"**)  
plt.ylabel(**"Accuracy"**)  
plt.show()

**Summary for KNN:**

* When we taken n\_neighbors=1 the accuracy score value is more comparing to n\_neighbors=50
* From the result my observation is accuracy score score are inversely proportional as shown in above output screens

**Deployment:**

Code is written in python and we used pycharm to run this and printed result in the python console.

**Limitations:**

No limitations for code snippets.