CS5590/490-Python/DeepLearning

LAB ASSIGNMENT-3

By: Syed Jawad Hussain Shah

Student ID: 16117985

Class ID: 46

**Author:** Syed Jawad Hussain Shah

**Objective:**

The objective of this lab task is to get familiar with Python language and its statistical features, like Linear discriminant model, Support vector machine model, K nearest neighbors(KNN) and Natural language processing.

**Features:**

The features of this lab include to write a program to implement a Linear Discriminant Analysis model on a certain data for classification. The second feature of the program is to Implement Support Vector Machine classification on a certain data from datasets with linear and RBF kernel and report on their accuracy. The third task is to get familiar with Natural language processing in python and use this feature to read an input file, lemmatize it, create bigrams, find the 5 most common bigrams, and concatenate the sentences within that text file that contains those common bigrams. The fourth feature is to report on the k nearest neighbor algorithm and discuss how the changes in K will affect the accuracy of the model and provide justification to reasoning.

**Configuration:**

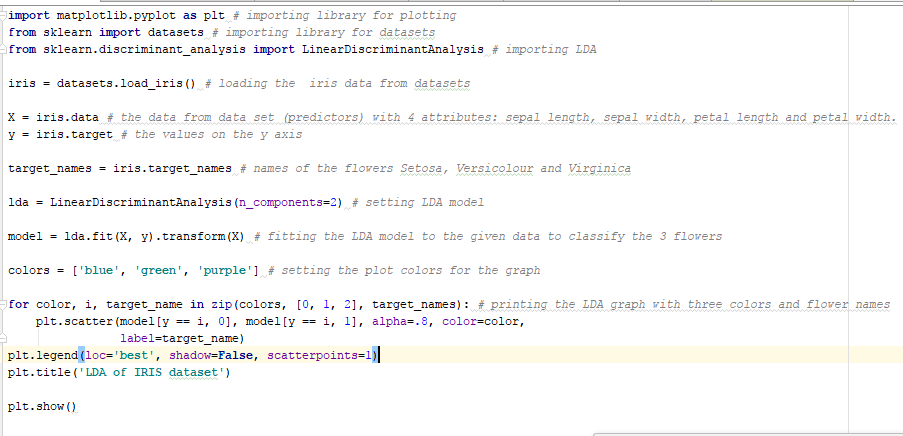
Python 3.6.4

IDE: JetBrains PyCharm community Edition 2017.3.3

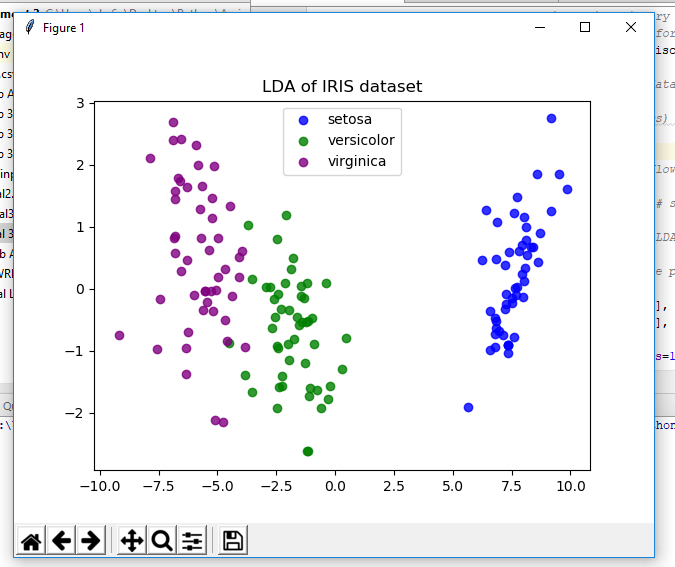
**Input/output (screenshots):**

Q1) Linear Discriminant Analysis:

1. Code:

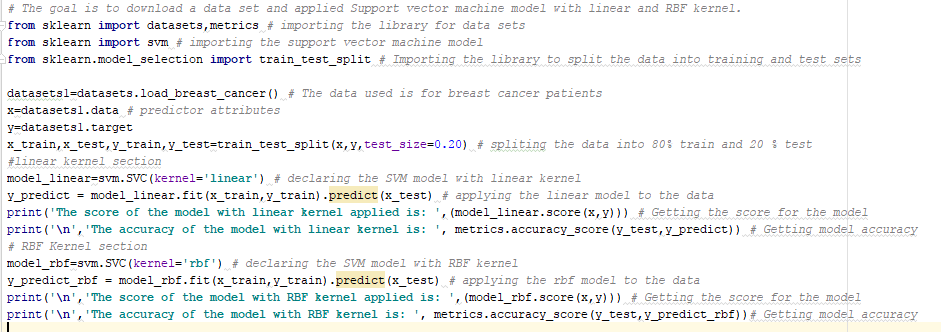


1. Results:

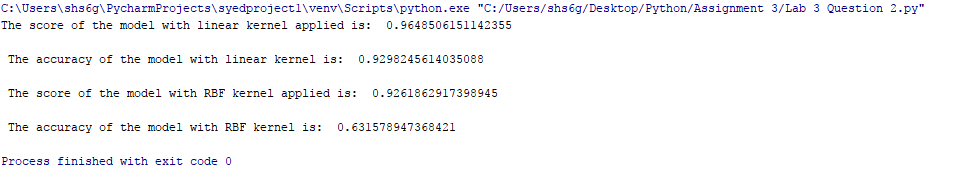


Q2) SVM model:

1. Code:

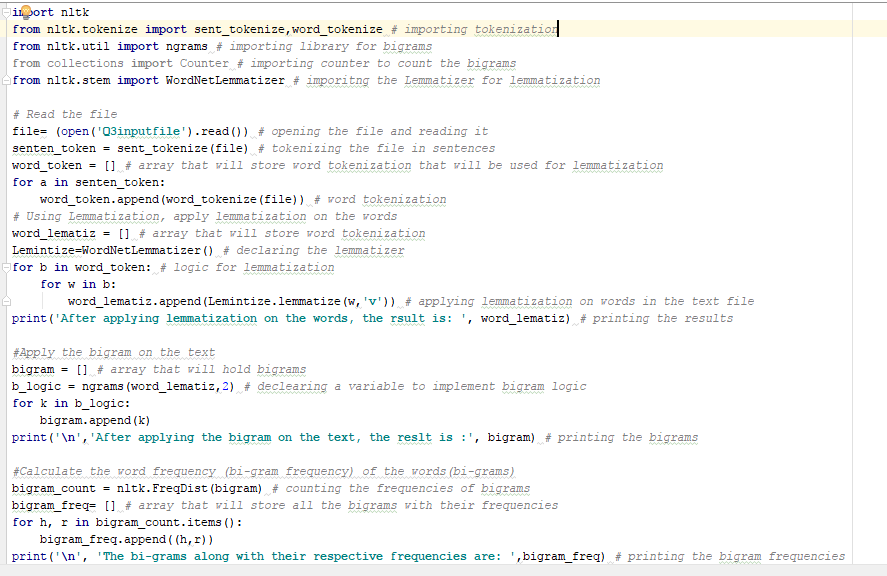


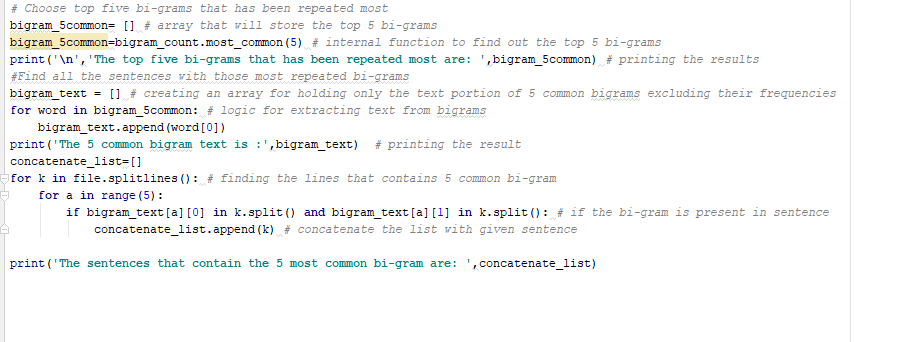
Results:



Q3) Natural language processing:

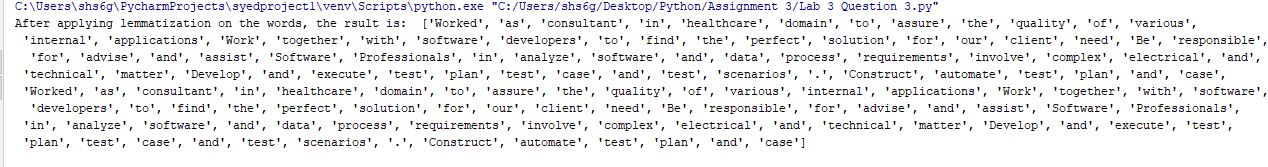
Code:





Results:

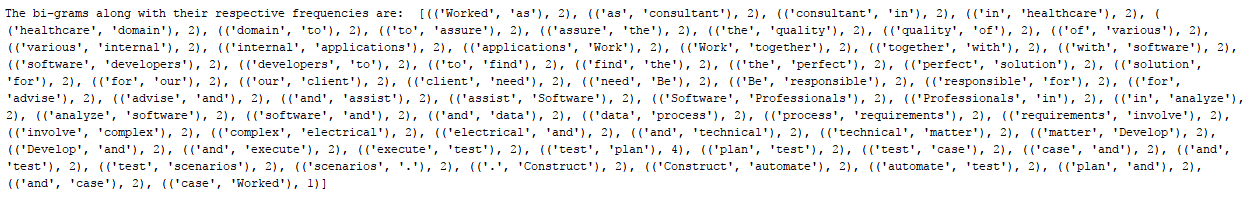
1. Applying lemmatization on the words:



1. Applying the bigram on the text:

|  |
| --- |
|  |

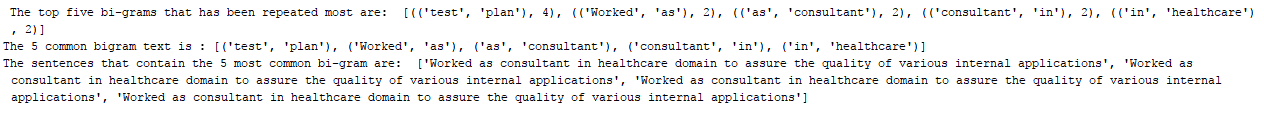
1. Calculate the word frequency (bi-gram frequency) of the words (bi-grams):



1. Choose top five bi-grams that has been repeated most:

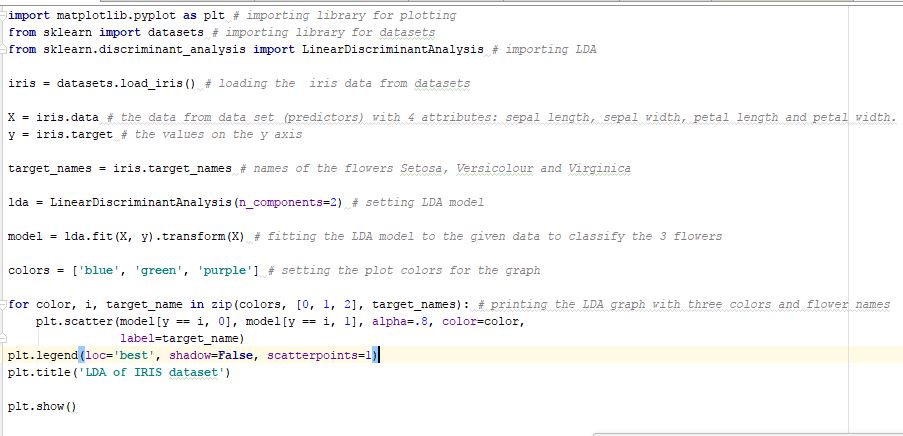


1. Find all the sentences with those most repeated bi-grams, extract those sentences and concatenate:



**Explanation of the implementation including code snippet**

Q1) : Linear Discriminant Analysis:



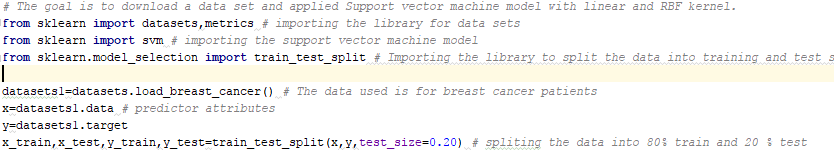
The program starts by importing all the libraries necessary that are required, for plotting, for loading data, and for implementing Linear Discriminant Analyses. Iris data is used, which contains 4 attributes of the flowers, sepal length, sepal width, petal length and petal width. The goal of the program is to apply Linear Discriminant Analyses to this data and classify the flowers into three categories, setosa, versicolor, and verginicia. The color used for plotting are blue, green and purple. At the end the program classifies the data into either of the 3 flowers. Since there were three classes in the response, LDA was a better choice for model.

**Explanations about the difference of logistic regression and Linear Discriminant Analysis:**

Logistic regression is generally used for two class classification. LDA is used for multi-class classification (more than two classes). Logistic regression is based on Maximum likelihood estimation. LDA is Based on Least squares estimation. Logistic regression estimates probability immediately and conditionally. LDA estimates probability mediately and uses both conditional and marginal information. Logistic regression is not very sensitive to outliers, whereas LDA is quite sensitive to outliers. LDA creates discriminant function(s) in order to maximize the difference between the groups on the function. Logistic regression works like ordinary least squares regression but on the logit of the dependent variable.

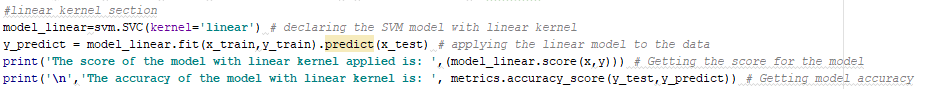
Q2) SVM model:

1. Dataset selection, loading, splitting the data to 20% testing data, 80% training data:



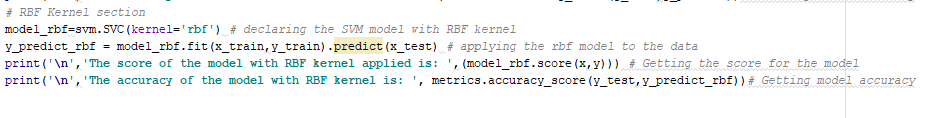
The program starts by importing all the libraries for datasets, SVM model and splitting the data into training data and test data. The data set used is from datasets library, related to the breast cancer patients. The data is split into training data (80%) and test data (20%).

1. SVC with Linear kernel:



The variable model\_linear implements a SVC model with linear kernel. After that this model is applied to the data and the model score and accuracy is obtained by .score and .accuracy\_score functions of python svc.

1. SVC with Rbf kernel:



The variable model\_rbf implements a SVC model with Rbf kernel. After that this model is applied to the data and the model score and accuracy is obtained by .score and .accuracy\_score functions of python svc.

Accuracy report of linear kernel and Rbf kernel:

The accuracy of linear kernel is 92% and score of 0.96.

The accuracy of Rbf kernel is 63% and score of 0.92.

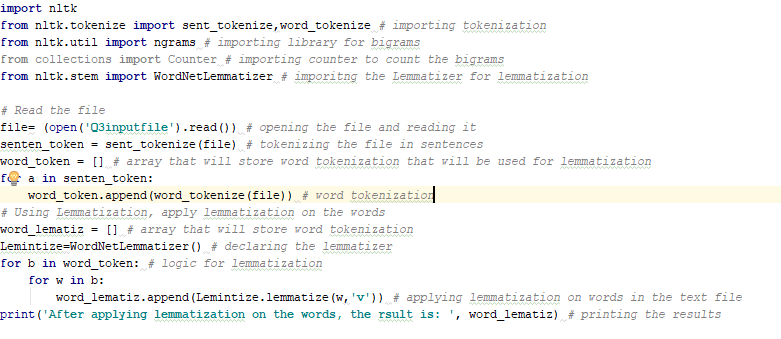
The linear kernel has higher accuracy and score than the Rbf kernel which makes it more accurate.

**Views on how to increase the accuracy and which kernel is the best for dataset:**

Linear kernel is easy and fast to train the model as compared to Rbf kernel. Since in my example I have used 80 % training data (more training data), it will be a wise decision to use linear kernel. That could be the reason behind the high accuracy and score of the linear kernel model. Also, linear kernel model is also less prone to overfitting than Rbf kernel. Since the data I used for this exercise is linear (not multidimensional) and with the above explanation, the accuracy of the model will increase when linear kernel is used with higher % of training data (may be 90 %), since this higher percentage will not be prone to overfitting. Also the number of features were higher than number of observation, linear kernel would be a better choice.

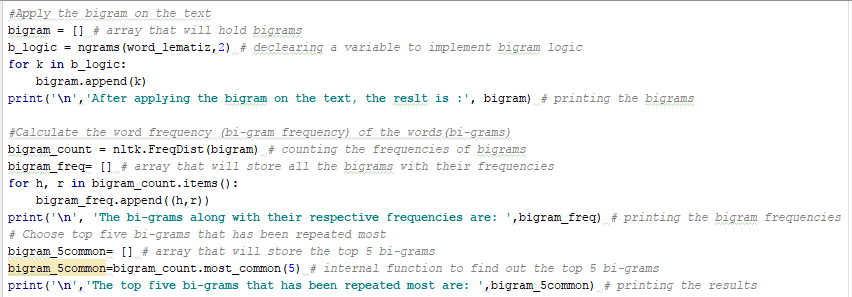
Q3) Natural Language processing:

1. Using Lemmatization, apply lemmatization on the words



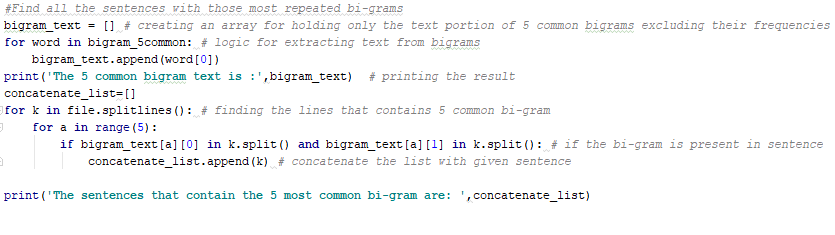
The program starts by importing all the libraries needed for natural language processing. First, it reads a input file “Q3inputfile”. Then it tokenizes the sentences with in that text file. After that the word in the text files are tokenized so that it can be used for lemmatization. For loops are used for word tokenization and lemmatization. At the end of this section of the code the lemmatization is printed as a output.

1. Applying the bigram on the text, calculating their frequency, choosing top 5 bigrams:



So, in this part the ngram feature of python is used to create the bigrams. The bigrams frequency was then counted using nltk.FreqDist functionality of the python. After that a for loop was used to store the bi-grams along with their frequencies in a list called bigram\_freq and the list was printed. After that most\_common() function of the python was used to extract the 5 most common bigrams from bigram\_freq and stored it in bigram\_5common list. This list was also printed.

C) Finding all the sentences with those most repeated bi-grams, extracting those sentences and concatenating:



Once the 5 most common bigrams were found along with their respective frequencies, the text from those bigrams was extracted and stored in a list called bigram\_text and the list was printed. The text was compared with all the lines in the input file and the sentences that contained those bigrams were concatenated in a list called concatenate\_list. A for loop in range 5 was used because there were only 5 bigrams that we need to find the in the sentences. At the end of the program the list that contains all the sentences that has the 5 most common bigrams, concatenate\_list, was printed.

**Q4)** 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

The change in accuracy with respect to change in K, can be either high or low. It really depends on the value of k. Typically; a smaller value of K is preferred over a larger one. A small value of k means that noise will have a higher influence on the result and a large value make it computationally expensive. The general rule of thumb is to set K to the square root of the number of training patterns/samples and this leads to better results.

So now, If k = 1, then the object is simply assigned to the class of that single nearest neighbor. This is not a good assignment of K since noise in the data will impact the model and the overall accuracy of the model will be reduced. This can also suffer from underfitting. On the other hand, if K = 50, the decision boundary becomes smoother and model become simpler, but it can have impact on the computational expenses and can also suffer from overfitting. I will go with a K value that is in between the 1 and 50, for example k = 15. So in this way if computational expenses is not an issue, the slightly higher value of K will ensure the model accuracy and will reduce the error rate.

**Code:**

**Q1)**

*# The goal is to apply LDA to a data set***import** matplotlib.pyplot **as** plt *# importing library for plotting***from** sklearn **import** datasets *# importing library for datasets***from** sklearn.discriminant\_analysis **import** LinearDiscriminantAnalysis *# importing LDA*iris = datasets.load\_iris() *# loading the iris data from datasets*X = iris.data *# the data from data set (predictors) with 4 attributes: sepal length, sepal width, petal length and petal width.*y = iris.target *# the values on the y axis*target\_names = iris.target\_names *# names of the flowers Setosa, Versicolour and Virginica*lda = LinearDiscriminantAnalysis(n\_components=2) *# setting LDA model*model = lda.fit(X, y).transform(X) *# fitting the LDA model to the given data to classify the 3 flowers*colors = [**'blue'**, **'green'**, **'purple'**] *# setting the plot colors for the graph***for** color, i, target\_name **in** zip(colors, [0, 1, 2], target\_names): *# printing the LDA graph with three colors and flower names* plt.scatter(model[y == i, 0], model[y == i, 1], alpha=.8, color=color,  
 label=target\_name)  
plt.legend(loc=**'best'**, shadow=**False**, scatterpoints=1)  
plt.title(**'LDA of IRIS dataset'**)  
  
plt.show()

**Q2)**

*# The goal is to download a data set and applied Support vector machine model with linear and RBF kernel.***from** sklearn **import** datasets,metrics *# importing the library for data sets***from** sklearn **import** svm *# importing the support vector machine model***from** sklearn.model\_selection **import** train\_test\_split *# Importing the library to split the data into training and test sets*datasets1=datasets.load\_breast\_cancer() *# The data used is for breast cancer patients*x=datasets1.data *# predictor attributes*y=datasets1.target  
x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.20) *# spliting the data into 80% train and 20 % test  
#linear kernel section*model\_linear=svm.SVC(kernel=**'linear'**) *# declaring the SVM model with linear kernel*y\_predict = model\_linear.fit(x\_train,y\_train).predict(x\_test) *# applying the linear model to the data*print(**'The score of the model with linear kernel applied is: '**,(model\_linear.score(x,y))) *# Getting the score for the model*print(**'\n'**,**'The accuracy of the model with linear kernel is: '**, metrics.accuracy\_score(y\_test,y\_predict)) *# Getting model accuracy  
# RBF Kernel section*model\_rbf=svm.SVC(kernel=**'rbf'**) *# declaring the SVM model with RBF kernel*y\_predict\_rbf = model\_rbf.fit(x\_train,y\_train).predict(x\_test) *# applying the rbf model to the data*print(**'\n'**,**'The score of the model with RBF kernel applied is: '**,(model\_rbf.score(x,y))) *# Getting the score for the model*print(**'\n'**,**'The accuracy of the model with RBF kernel is: '**, metrics.accuracy\_score(y\_test,y\_predict\_rbf))*# Getting model accuracy*

**Q3)**

**import** nltk  
**from** nltk.tokenize **import** sent\_tokenize,word\_tokenize *# importing tokenization***from** nltk.util **import** ngrams *# importing library for bigrams***from** collections **import** Counter *# importing counter to count the bigrams***from** nltk.stem **import** WordNetLemmatizer *# imporitng the Lemmatizer for lemmatization  
  
# Read the file*file= (open(**'Q3inputfile'**).read()) *# opening the file and reading it*senten\_token = sent\_tokenize(file) *# tokenizing the file in sentences*word\_token = [] *# array that will store word tokenization that will be used for lemmatization***for** a **in** senten\_token:  
 word\_token.append(word\_tokenize(file)) *# word tokenization  
# Using Lemmatization, apply lemmatization on the words*word\_lematiz = [] *# array that will store word tokenization*Lemintize=WordNetLemmatizer() *# declaring the lemmatizer***for** b **in** word\_token: *# logic for lemmatization* **for** w **in** b:  
 word\_lematiz.append(Lemintize.lemmatize(w,**'v'**)) *# applying lemmatization on words in the text file*print(**'After applying lemmatization on the words, the rsult is: '**, word\_lematiz) *# printing the results  
  
#Apply the bigram on the text*bigram = [] *# array that will hold bigrams*b\_logic = ngrams(word\_lematiz,2) *# declearing a variable to implement bigram logic***for** k **in** b\_logic:  
 bigram.append(k)  
print(**'\n'**,**'After applying the bigram on the text, the reslt is :'**, bigram) *# printing the bigrams  
  
#Calculate the word frequency (bi-gram frequency) of the words(bi-grams)*bigram\_count = nltk.FreqDist(bigram) *# counting the frequencies of bigrams*bigram\_freq= [] *# array that will store all the bigrams with their frequencies***for** h, r **in** bigram\_count.items():  
 bigram\_freq.append((h,r))  
print(**'\n'**, **'The bi-grams along with their respective frequencies are: '**,bigram\_freq) *# printing the bigram frequencies  
# Choose top five bi-grams that has been repeated most*bigram\_5common= [] *# array that will store the top 5 bi-grams*bigram\_5common=bigram\_count.most\_common(5) *# internal function to find out the top 5 bi-grams*print(**'\n'**,**'The top five bi-grams that has been repeated most are: '**,bigram\_5common) *# printing the results  
#Find all the sentences with those most repeated bi-grams*bigram\_text = [] *# creating an array for holding only the text portion of 5 common bigrams excluding their frequencies***for** word **in** bigram\_5common: *# logic for extracting text from bigrams* bigram\_text.append(word[0])  
print(**'The 5 common bigram text is :'**,bigram\_text) *# printing the result*concatenate\_list=[]  
**for** k **in** file.splitlines(): *# finding the lines that contains 5 common bi-gram* **for** a **in** range(5):  
 **if** bigram\_text[a][0] **in** k.split() **and** bigram\_text[a][1] **in** k.split(): *# if the bi-gram is present in sentence* concatenate\_list.append(k) *# concatenate the list with given sentence*print(**'The sentences that contain the 5 most common bi-gram are: '**,concatenate\_list)

**Deployment:**

1. Save the folder in your local machine.
2. Install Python 3.6.4 and PyCharm IDE on your machine.
3. Run PyCharm, click on file->open->files location of the saved folder.
4. Select the desired code file with .py extension to execute.
5. Right click on the code screen and then click on run "filename".
6. Give the input and validate the output.

**limitation**

1. In Q1, the program uses only LDA and doesn’t apply the logistic regression model to same data for comparison.
2. The program in Q2 is more bias towards linear kernel.
3. The Program in Q2 apply the linear and Rbf kernel to only linear data and doesn’t use a multidimensional data to don’t apply these kernels for accuracy comparison.
4. The program in Q3 is only used on a smaller text file and is not implemented on a bigger data sets.
5. The text file for Q3 is made in such way to remove commas and full stops, because the code doesn’t handle commas, hyphens, breaks and full stops.
6. Only a verbal explanation is given for Q4 and the practical implementation of the model with different K values is missing.

**References**

<https://stats.stackexchange.com/questions/95247/logistic-regression-vs-lda-as-two-class-classifiers>

<https://www.quora.com/What-is-the-difference-between-logistic-regression-and-discriminant-analysis>

<https://docs.scipy.org/doc/numpy/reference/generated/numpy.concatenate.html>

<https://codereview.stackexchange.com/questions/57046/identifying-matching-bigrams-in-large-text-collection>

<https://inzaniak.github.io/pybistuffblog/posts/2017/04/20/python-count-frequencies-with-nltk.html>

<https://stats.stackexchange.com/questions/73032/linear-kernel-and-non-linear-kernel-for-support-vector-machine>

<https://www.researchgate.net/post/How_can_we_find_the_optimum_K_in_K-Nearest_Neighbor>

<https://discuss.analyticsvidhya.com/t/how-to-choose-the-value-of-k-in-knn-algorithm/2606/2>

<https://stats.stackexchange.com/questions/126051/choosing-optimal-k-for-knn>