**Question-01:**

c = LinearDiscriminantAnalysis()  
clas=np.array([3,2,1,3,2,1,3,2,1,3,2,1,3,2,1])  
df = pd.DataFrame.from\_csv(**"input.csv"**)  
data\_matrix = df.as\_matrix()  
print(data\_matrix)  
c.fit(data\_matrix,clas)  
print(**"Prediction for LDA:"**)  
print(c.predict([[2, 12]]))

|  |  |
| --- | --- |
| **Logistic regression** | **Linear Discriminant Analysis** |
| New method | Old method |
| The group can have any number of different n. | The group should have same n. |
| Estimates max. likelihood | Estimates least sq’s. |
| It is preferred usually due to because of its robustness | IF all the requirements are met, it is most commonly used. |

**Question-02:**

cds = datasets.load\_breast\_cancer()  
a = cds.data  
b = cds.target  
at1,at2,bt1,bt2=train\_test\_split(a,b,test\_size=0.2)  
  
  
lsvc = svm.SVC(kernel=**'linear'**)  
b\_predic = lsvc.fit(at1,bt1).predict(at2)  
print(**"linear kernel score is:"**)  
print(lsvc.score(a,b))  
  
print(**"linear kernel accuracy score :"**)  
print(metrics.accuracy\_score(bt2,b\_predic))  
  
rsvc = svm.SVC(kernel=**'rbf'**)  
b\_predic = rsvc.fit(at1,bt1).predict(at2)  
print(**"Rbf kernel score is:"**)  
print(rsvc.score(a,b))  
  
print(**"Rbf kernel accuracy score :"**)  
print(metrics.accuracy\_score(bt2,b\_predic))

**Question-03:**

**from** tokenize **import** tokenize  
  
**import** nltk  
**from** nltk.stem **import** WordNetLemmatizer  
**from** nltk.tokenize **import** word\_tokenize,sent\_tokenize,wordpunct\_tokenize  
**from** nltk.util **import** ngrams  
**from** collections **import** Counter, defaultdict  
  
input=**"input.txt"**file = open(**'input.txt'**, **'r'**)  
i= file.read()  
tiles=[]  
inp=i.split(**" "**)  
lem = WordNetLemmatizer()  
**for** word **in** inp:  
 tiles.append(lem.lemmatize(word,**"n"**)) *#lemmatization applied here*print(tiles)  
  
data = sent\_tokenize(i)  
tiles1 = []  
**for** words **in** data:  
 tiles1.append(word\_tokenize(words)) *#Word tokenization applied here on sentences which were tokenized above.*print(**"Printing Tokenization"**)  
print(tiles1)  
  
words = nltk.word\_tokenize(i)  
bigram = ngrams(words,2)  
print(**"Printing bigrams"**)  
print(Counter(bigram))  
  
count = defaultdict(int)  
**for** n **in** ngrams(words, 2, **False**):  
 count[n]=count[n]+1  
  
print(**"Frequencies:"**)  
**for** c, n **in** sorted(((c, n) **for** n, c **in** count.items()), reverse=**True**):  
 print(c, n)  
  
a = 0  
b = 0  
s = []  
  
**for** line **in** i.splitlines():  
 **for** a **in** range(5):  
   
 **if** count[a] **in** line.split() **and** count[a] **in** line.split():  
 *# print("1")* d = line.split().index(count[a])  
 s.append(d)  
  
print(s)

**Question-04:**

**The main metric with the k-nearest classification is Euclidean distance between the points. If the k value is 1, then from the center point, the distance is calculated to one nearest neighbor, it can have only one training set to perform. If the k value grows, the center point looks for the nearest k points and finds the distance for all k points, and it contains more training sets to consider, the accuracy decreases. So, if the k value increases, the accuracy gradually decreases.**