Pro one:

**from** sklearn.linear\_model **import** LogisticRegression  
**from** sklearn.metrics **import** accuracy\_score  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.datasets **import** load\_digits  
  
*# Loading the digits data into variable*db=load\_digits()  
ctn=db.data  
ar1=db.target  
sl, sw, pl, pw = train\_test\_split(ctn, ar1, test\_size=0.2, random\_state=30)  
*# Now let us perform LDA  
#ldperform = LinearDiscriminantAnalysis()  
# Here we are performing regression with logistic orientation*logistic=LogisticRegression()  
*#ldperfrom.fit(xtrain,ytrain)  
#yprediction=ldperfrom.predict(xtest) #Predicting the outcomes*logistic.fit(sl, pl)  
*# Now here we perform outcome predicting process.*yprediction=logistic.predict(sw)  
  
*# let us Calculate accuracy score with applyication of LinearDiscriminantAnalysis model*print(**"Accuracy"**, accuracy\_score(yprediction, pw))

Pro two:

**from** sklearn **import** svm  
**from** sklearn **import** datasets  
**from** sklearn **import** metrics  
**from** sklearn.model\_selection **import** train\_test\_split  
  
*# Default data set from the available, digits is being loaded*iris = datasets.load\_digits()  
  
*# loading data into the variable*xload = iris.data  
  
*# loading target into the variable*yload = iris.target  
  
*# We are going to divide the test data and training data in*xtrainer, xtesting, ytrainer, ytesting = train\_test\_split(xload, yload, test\_size=0.2)  
  
print(**"Support Vector Machine classification with Linear Model :"**)  
  
*# We are Creating model for Support vector machine with linear\_kernel*linearkernel = svm.SVC(kernel=**"linear"**)  
linearkernel.fit(xtrainer, ytrainer)  
yprediction = linearkernel.predict(xtesting)  
  
print(**"The accuracy that is obtained for support vector machine classification with Linear kernel: %.3f"** % metrics.accuracy\_score(ytesting, yprediction))  
  
print(**"\n Support vector machine classification with rbf kernel Model :"**)  
  
*# Creating model for Support Vecctor Machinde using RBF\_kernel*rbfmodel = svm.SVC(kernel=**"rbf"**)  
  
*# Fitting the data*rbfmodel.fit(xtrainer, ytrainer)  
  
yprediction = rbfmodel.predict(xtesting)  
  
  
*# Checking the accuracy score on comparing ytesting data and ypredicting data*print(**"The accuracy score that is obtained for Support Vector Machine Classification using RBF\_kernel is : %.3f"** % metrics.accuracy\_score(ytesting, yprediction))

Pro three:

**from** nltk.stem **import** WordNetLemmatizer  
**from** nltk.tokenize **import** word\_tokenize,sent\_tokenize  
**from** nltk.collocations **import** ngrams  
**from** nltk.util **import** bigrams  
**import** collections  
**from** collections **import** Counter  
**import** nltk  
**from** operator **import** itemgetter  
  
*# Opening the input text file with read mode.*f = open(**"task4.txt"**, **"r"**)  
  
*# Reading the text file*contentsoffile = f.read()  
  
*# Printing the contents of the file*print(**"Contents of the file:"**,**"\n"**, contentsoffile)  
print(**"----------------------------------------------------"**)  
print(**"-----------------------------------------------------"**)  
  
*# Performing word lemmatization*lem = WordNetLemmatizer()  
  
*# Tokenizing the contents of the file.*words = word\_tokenize(contentsoffile)  
  
print(**"Lemmatization with verb form of the words:"**)  
**for** word **in** words:  
 print(lem.lemmatize(word, pos=**'v'**))  
  
print(**"-----------------------------------------------------"**)  
print(**"-----------------------------------------------------"**)  
print(**"Performing bi-gram on the text:"**)  
cp = word\_tokenize(contentsoffile)  
li = []  
  
*# Accessing ngram function to find bigrams from the given text.*bigramfinder = ngrams(cp, 2)  
**for** a **in** bigramfinder:  
 li.append(a)  
print(li)  
  
print(**"-----------------------------------------------------"**)  
print(**"-----------------------------------------------------"**)  
  
*# Using counter function to cpunt the number of occurances of each bigram.*word\_count = Counter(li)  
print(**" Calculating the word frequency of Bi-Gram:"**)  
print(word\_count)  
  
print(**"-----------------------------------------------------"**)  
print(**"-----------------------------------------------------"**)  
print(**"Finding the top 5 bigrams:"**)  
freqcnt = nltk.FreqDist(li)  
tf = freqcnt.most\_common(5)  
  
*# Printing top five bigrams which are found out using most\_common function.*print(tf)  
  
**with** open(**'task4.txt'** , **'r'**) **as** file:  
 ln = file.readlines()  
fit= **''  
  
for** mp **in** ln:  
 fit= fit + mp  
senti = sent\_tokenize(fit)  
rep\_sent1 = []  
  
  
**for** sentences **in** senti:  
 **for** word,words **in** li:  
 **for** ((cp, mp), l) **in** tf:  
 **if** (word, words == cp, mp):  
 rep\_sent1.append(sentences)  
  
  
*# Printing the lines of text that contain the most common top five bigrams.*print(**"-----------------------------------------------------"**)  
print(**"-----------------------------------------------------"**)  
print (**"\n Summarising the text and finding out the lines with top five bigrams"**)  
fc = nltk.FreqDist(rep\_sent1)  
fff = fc.most\_common(5)  
**for** ke,val **in** fff:  
 print(**"\n"**,ke)

Pro four:

**from** sklearn.metrics **import** accuracy\_score  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn **import** neighbors,datasets  
  
*# loading the default dataset IRIS.*k1 = datasets.load\_iris()  
  
*# loading data into the variable*k2 = k1.data  
  
*# Loading target into the variable*labd1=k1.target  
  
*# Here we define and divide test data to 20% and training data at 80%*xtrainer, xtester, ytraining, ytesting = train\_test\_split(k2, labd1, test\_size=0.2, random\_state=42)  
  
*# Here we are seen setting the n-neighbours to the perform KNN*knn = neighbors.KNeighborsClassifier(n\_neighbors=50)  
knn.fit(xtrainer, ytraining)  
  
*# Now let us perform prediction*yprediction=knn.predict(xtester)  
  
  
*# now we perform accuracy calculation over knn algorithm*print(**" Accuracy for (k = 50) "**, accuracy\_score(yprediction, ytesting))

now we check accuracy when k = 1, by running the same above code.