**import** pandas **as** pd  
**import** numpy **as** np  
**from** mpl\_toolkits.mplot3d **import** Axes3D  
**import** nltk **as** nlt  
**import** matplotlib.pyplot **as** plt  
**from** sklearn.pipeline **import** Pipeline  
**from** sklearn.feature\_extraction.text **import** CountVectorizer  
**from** sklearn.feature\_extraction.text **import** TfidfTransformer  
**from** sklearn.metrics **import** classification\_report,confusion\_matrix  
**from** sklearn.linear\_model **import** SGDClassifier  
**from** nltk.stem.snowball **import** SnowballStemmer  
  
*#Fetching the dataset 20Newsgroups***from** sklearn.datasets **import** fetch\_20newsgroups  
  
*#Storing it in a variable for training and testing*t\_train=fetch\_20newsgroups(subset=**'train'**,shuffle=**True**)  
t\_test = fetch\_20newsgroups(subset=**'test'**, shuffle=**True**)  
  
*#Printing them*print(**"The Headings Are: "**)  
print(t\_train.target\_names) *#Printing the headings of all files*print(**"The First line is: "**)  
print(**"\n"**.join(t\_train.data[0].split(**"\n"**)[:3]))  
  
*#Extracting the Features from the files*c\_vect=CountVectorizer()  
x\_train=c\_vect.fit\_transform(t\_train.data) *#Counting the more common words that appear in each file  
  
#reduce the weightage of more common words like (the, is, an etc.)  
#Using TF-IDF i.e Term Frequency times inverse document frequency.*tfidf\_transformer = TfidfTransformer()  
X\_train\_tfidf = tfidf\_transformer.fit\_transform(x\_train)  
print(X\_train\_tfidf.shape) *#Printing the frequency of the words in the form of document vector matrix  
  
#Training Support Vector Machines[SVM] and calculating its performance  
#nlt.download('stopwords') Downloading the nltk package for Stopping words*stemmer = SnowballStemmer(**"english"**, ignore\_stopwords=**True**) *#We use Snowball Stemmer for stemming***class** StemmedCountVectorizer(CountVectorizer):  
 **def** build\_analyzer(self):  
 analyzer = super(StemmedCountVectorizer, self).build\_analyzer()  
 **return lambda** doc: ([stemmer.stem(w) **for** w **in** analyzer(doc)])  
stemmed\_count\_vect = StemmedCountVectorizer(stop\_words=**'english'**)  
  
*#The stemmer then is passed to the SVM classifier.*text\_mnb\_stemmed = Pipeline([(**'vect'**, stemmed\_count\_vect), (**'tfidf'**, TfidfTransformer()),  
 (**'clf-svm'**, SGDClassifier(loss=**'hinge'**, penalty=**'l2'**,alpha=1e-3, n\_iter=5, random\_state=42))])  
  
text\_mnb\_stemmed = text\_mnb\_stemmed.fit(t\_train.data, t\_train.target)  
predicted\_mnb\_stemmed = text\_mnb\_stemmed.predict(t\_test.data)  
print(**"The Accuracy of the Model is:"**)  
print(np.mean(predicted\_mnb\_stemmed == t\_test.target))  
  
  
*#Plotting:*print(**"\nConfusion Matrix:\n"**,confusion\_matrix(t\_test.target,predicted\_mnb\_stemmed))  
cr = classification\_report(t\_test.target,predicted\_mnb\_stemmed)  
print(**"\nClassification Report:\n"**,cr)  
cr = cr.split()  
j=5;p=[];r=[]  
**for** i **in** range(0,20):  
 p.append(cr[j])  
 r.append(cr[j+1])  
 j+=5  
p = [float(i) **for** i **in** p]  
r = [float(i) **for** i **in** r]  
x = [i **for** i **in** range(0,20)]  
  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection=**'3d'**)  
ax.scatter(x, p, r, c=**'b'**, s=10)  
ax.set\_title(**'Precision-Recall Graph'**)  
ax.set\_xlabel(**''**)  
ax.set\_ylabel(**'Precision'**)  
ax.set\_zlabel(**'Recall'**)  
plt.show()