PROGRAM 6: Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Libraries can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

|  |
| --- |
| **from** **sklearn.datasets** **import** fetch\_20newsgroups  twenty\_train = fetch\_20newsgroups(subset='train', shuffle=**True**)  x = len(twenty\_train.target\_names)  print("**\n** The number of categories:",x)  print("**\n** The **%d** Different Categories of 20Newsgroups**\n**" %**x**)  i=1  **for** cat **in** twenty\_train.target\_names:  print("Category[**%d**]:" %**i**,cat)  i=i+1  print("**\n** Length of training data is",len(twenty\_train.data))  print("**\n** Length of file names is ",len(twenty\_train.filenames))  print("**\n** The Content/Data of First File is :**\n**")  print(twenty\_train.data[0]) |

# Considering only four Categories

|  |
| --- |
| categories = ['alt.atheism', 'soc.religion.christian','comp.graphics', 'sci.med']  twenty\_train = fetch\_20newsgroups(subset='train', categories=categories, shuffle=**True**, random\_state=42)  print("**\n** Reduced Target Names:**\n**",twenty\_train.target\_names)  print("**\n** Reduced Target Length:**\n**", len(twenty\_train.data))  print("**\n**First Document : ",twenty\_train.data[0]) |

|  |
| --- |
| **from** **sklearn.feature\_extraction.text** **import** CountVectorizer  count\_vect = CountVectorizer()  X\_train\_counts = count\_vect.fit\_transform(twenty\_train.data)  print("**\n**(Target Length , Distinct Words):",X\_train\_counts.shape)  print("**\n** Frequency of the word algorithm:", count\_vect.vocabulary\_.get('algorithm')) |

|  |
| --- |
| **from** **sklearn.feature\_extraction.text** **import** TfidfTransformer  tf\_transformer = TfidfTransformer(use\_idf=**False**).fit(X\_train\_counts)  X\_train\_tf = tf\_transformer.transform(X\_train\_counts)  X\_train\_tf.shape |

|  |
| --- |
| tfidf\_transformer = TfidfTransformer()  X\_train\_tfidf = tfidf\_transformer.fit\_transform(X\_train\_counts)  X\_train\_tfidf.shape  **from** **sklearn.naive\_bayes** **import** MultinomialNB  clf = MultinomialNB().fit(X\_train\_tfidf, twenty\_train.target) |

|  |
| --- |
| docs\_new = ['God is love', 'OpenGL on the GPU is fast']  X\_new\_counts = count\_vect.transform(docs\_new)  X\_new\_tfidf = tfidf\_transformer.transform(X\_new\_counts)  predicted = clf.predict(X\_new\_tfidf)  **for** doc, category **in** zip(docs\_new, predicted):  print('**%r** => **%s**' % (doc, twenty\_train.target\_names[category])) |

|  |
| --- |
| **from** **sklearn.pipeline** **import** Pipeline  text\_clf = Pipeline([('vect', CountVectorizer()),  ('tfidf', TfidfTransformer()),  ('clf', MultinomialNB()),  ]) |

|  |
| --- |
| text\_clf.fit(twenty\_train.data, twenty\_train.target) |

# Evaluation of the performance on the test set

|  |
| --- |
| **import** **numpy** **as** **np**  twenty\_test = fetch\_20newsgroups(subset='test',categories=categories, shuffle=**True**, random\_state=42)  docs\_test = twenty\_test.data  predicted = text\_clf.predict(docs\_test)  np.mean(predicted == twenty\_test.target) |

|  |
| --- |
| **from** **sklearn** **import** metrics  print(metrics.classification\_report(twenty\_test.target, predicted,  target\_names=twenty\_test.target\_names)) |

|  |
| --- |
| metrics.confusion\_matrix(twenty\_test.target, predicted) |