**EE239AS Project 2**

**Classification Analysis**

**Student1**: Xiangrui Liu (004516858) **Student2**: Qi Wang (604522309)

**Student3**: Qinyi Yan (704406413) **Student4**: Ziyin You (404412651)

**Dataset and Problem Statement:**

*Question.a*

There are two ways to solve this problem:

1) Using the function "fetch\_20newsgroups" in "sklearn.datasets" library;

2) Download the data manually, which are automatically divided into "train" and "test" sets.

We choose the second method, with the help of "**os**" library.

Here we summarize the number of files for all 20 topics, where the files include those in "train" set and those in "test" set. The result is plotted as below:

As shown above, those topics which belong to "Computer Technology" and "Recreational Activity" are highlighted, and it can be seen that they are evenly distributed.

Then, the number of documents in these two groups are calculated below (\*Notice again, all the files in "train" and "test" are included):

|  |  |
| --- | --- |
| Computer Technology | Recreational Activity |
| **3903** | **3979** |

**Modeling Text Data and Feature Extraction:**

*Question.b*

In this question, before creating TFxIDF vector representation, several things should be done:

First, the stop words should be chosen. The method is shown below:

from sklearn.feature\_extraction import text

stop\_words = text.ENGLISH\_STOP\_WORDS

After that, in order to remove the different stems for all the words in the documents, we use:

from nltk.stem.snowball import SnowballStemmer

stemmer = SnowballStemmer("english")

Then, all the documents, after remove the headers, footers and quotes, are used to create the TFxIDF vector. The code is shown below:

from sklearn.datasets import fetch\_20newsgroups

all\_data=fetch\_20newsgroups(subset='all',shuffle=True,random\_state=42, remove=('headers','footers','quotes'))

from sklearn.feature\_extraction.text import TfidfVectorizer

TFxIDF = TfidfVectorizer(analyzer='word', tokenizer=tokenizer\_fun, stop\_words=stop\_words, token\_pattern='[a-zA-Z]{2,}',)

where tokenizer\_fun is a function to keep the roots of the words in the input data.

TFxIDF\_data = TFxIDF.fit\_transform(all\_data.data)

The result is shown to be **72399** terms (within all the 18846 documents in "train" and "test" sets).

*Question.c*

The way to solve Question (c) is just like (b), while in (c), we use one of the four topics:

('comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'misc.forsale', 'soc.religion.christian')

instead of all the documents. Also, for the " TfidfVectorizer" function, in order to find the 10 most significant terms for each of the four topics, we add the sub-function " max\_features=10".

Then, the results are shown below (notice that the results are the **roots** of the terms):

|  |  |
| --- | --- |
| comp.sys.ibm.pc.hardware | 'scsi', 'mb', 'drive', 'control', 'work', 'use', 'problem', 'ani', 'disk', 'card' |
| comp.sys.mac.hardware | 'drive', 'like', 'know', 'mac', 'work', 'use', 'problem', 'ani', 'appl', 'monitor' |
| misc.forsale | 'sell', 'pleas', 'ship', 'offer', 'price', 'drive', 'use', 'includ', 'sale', 'new' |
| soc.religion.christian | 'say', 'god', 'church', 'christian', 'peopl', 'believ', 'think', 'sin', 'jesus', 'know' |

**Feature Selection:**

*Question.d*

With the LSI feature generated in (b)

**Learning Algorithms:**

For Question e to h, firstly, we extract required 8 sub-classes from the total 20 classes. Then, in order to avoid overfitting problem, we define two functions -- data\_fun and LSI\_fun to get training dataset, test dataset, and their LSI respectively. We use training dataset to train the classifiers, and predict the test dataset LSI and compare predicted results with the original target values.

Since the original target names are sub-classes instead of Computer Technology and Recreation Activity groups, we need to classify the 8 sub-classes into 2 groups. Using instruction *train\_set.target\_names,* we could get the index of each target name:

['comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.ibm.pc.hardware', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey']

From the result, it can be seen that the first four classes (0-3) are sub-classes of Computer Technology, and the last four classes (4 - 7) are sub-classes of Recreation Activity. Therefore, if we divide the target by 4 and round the results, we could turn target values into binary sets, in which 0 represents Computer Technology group and 1 represents Recreation Activity group.

train\_target\_group = [ int(x / 4) for x in train\_set.target]

test\_target\_group = [ int(x / 4) for x in test\_set.target]

In the following problems, we use the binary target sets to train and test the classifiers.

*Question.e*

In this problem, we use svm.LinearSVC() package to train the classifier and predict test data. The main code is shown below:

from sklearn import svm

lin\_clf = svm.LinearSVC()

lin\_clf.fit(train\_LSI, train\_target\_group)

After training the linear SVM classifier, use the generated classifier lin\_clf to predict test data:

svm\_predicted = lin\_clf.predict(test\_LSI)

Then, to plot the ROC curve, we use roc\_curve, auc package and decision\_funciton to get the scores of output:

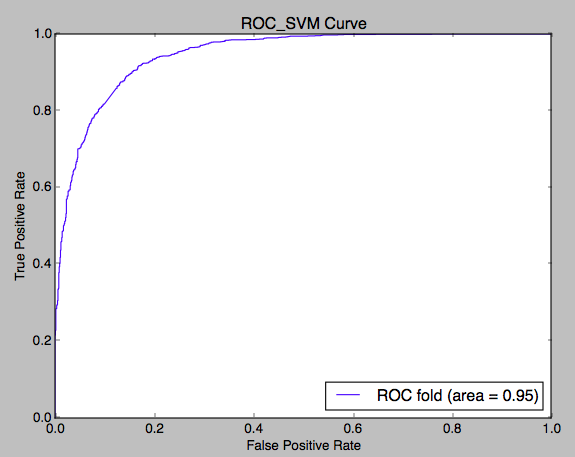
from sklearn.metrics import roc\_curve, auc

*y\_score\_test = lin\_clf.decision\_function(test\_LSI)*

*fpr, tpr, thresholds = roc\_curve(test\_target\_group, y\_score\_test)*

*roc\_auc = auc(fpr, tpr)*

The ROC curve of linear SVM is:



The main code to calculate confusion matrix, accuracy, precision, recall is shown here:

from sklearn.metrics import confusion\_matrix

confusion\_matrix(test\_target\_group, svm\_predicted)

from sklearn.metrics import accuracy\_score

svm\_accuracy = accuracy\_score(test\_target\_group, svm\_predicted)

from sklearn.metrics import precision\_score

svm\_precision\_score = precision\_score(test\_target\_group, svm\_predicted)

from sklearn.metrics import recall\_score

svm\_recall\_score = recall\_score(test\_target\_group, svm\_predicted)

The results of linear SVM classifier are shown in below table:

|  |  |  |  |
| --- | --- | --- | --- |
| Confusion Matrix | [[ 863, 312],  [ 64, 1526]] | Accuracy | 0.864014466546 |
| Recall | 0.830250272035 |
| Precision | 0.9959748427673 |

*Question.f*

In this problem, we mainly use cross\_validation package to find the best parameter lambda, and set the fold as 5. In order to find the best lambda, which makes the soft margin SVM classifier have best score, we do a for loop to train soft margin SVM classifiers with different lambda values and calculate their scores respectively. Then find the maximum score and its corresponding index. Then, we use the index to backtrace the lambda and train the final soft margin SVM classifier.

The main code is shown below:

from sklearn import svm

from sklearn import cross\_validation

import numpy as np

penalty = [-3, -2, -1, 0, 1, 2, 3]

for k in penalty:

soft\_svm\_clf = svm.LinearSVC(C=10\*\*k)

scores = cross\_validation. cross\_val\_score(soft\_svm\_clf, train\_LSI, train\_target\_group, cv=5)

print np.mean(scores)

Based the above code, we find that the best lambda is 1. Therefore, the soft margin SVM classifier is:

soft\_clf = svm.LinearSVC(C=10\*\*1)

soft\_clf.fit(train\_LSI, train\_target\_group)

soft\_svm\_predicted = soft\_clf.predict(test\_LSI)

The ROC curve of soft margin SVM classifier is:

The results of soft margin SVM classifier are shown in below table:

|  |  |  |  |
| --- | --- | --- | --- |
| Confusion Matrix | [[ 847, 328],  [ 79, 1511]] | Accuracy | 0.852802893309 |
| Recall | 0.950314465409 |
| Precision | 0.821642196846 |

*Question.g*

In this problem, we use naive\_bayes package to train the classifier and predict test data. The main code is shown below:

from sklearn.naive\_bayes import GaussianNB

nb\_clf = GaussianNB()

nb\_clf.fit(train\_LSI, train\_target\_group)

nb\_predicted = nb\_clf.predict(test\_LSI)

The ROC curve of naive Bayes classifier is:

The results of naive Bayes classifier are shown in below table:

|  |  |  |  |
| --- | --- | --- | --- |
| Confusion Matrix | [[936, 239],  [764, 826]] | Accuracy | 0.637251356239 |
| Recall | 0.519496855346 |
| Precision | 0.77558685446 |

*Question.e*

In this problem, we use logistic regression package to train the classifier and predict test data. The main code is shown below:

from sklearn import linear\_model, datasets

logreg = linear\_model.LogisticRegression(C=1e5)

logreg.fit(train\_LSI, train\_target\_group)

lr\_predicted = logreg.predict(test\_LSI))

The ROC curve of logistic regression is:

The results of logistic regression classifier are shown in below table:

|  |  |  |  |
| --- | --- | --- | --- |
| Confusion Matrix | [[ 850, 325],  [ 124, 1466]] | Accuracy | 0.837613019892 |
| Recall | 0.922012578616 |
| Precision | 0.818537130095 |

**Multiclass Classification:**

*Question.i*

In this part, we used three approaches to classify the data into four categories, instead of two. The 3 approaches are: Naïve Bayes algorithm classification, Support Vector Machine (SVM) classification with One Vs One (OVO) method and One Vs the Rest (OVR) method.

In order to build the multi-class classifier, the raw data should be processed based on the target categories, namely:

[‘comp.sys.ibm.pc.hardware’ , ‘comp.sys.mac.hardware’, ‘misc.forsale’, ‘soc.religion.christian’].

Upon on the process, the new set of training data and testing data along with the new targets will be generated.

The outcomes of the three classifiers are reported in the following pages.

***Naïve Bayes Multiclass Classifier***:

Since it’s a based on the assumption that each features is independent from the other features, it always indifferently performs classification, regardless of the number of targets. Therefore, the only modification on the Naïve Bayes classifier in order to perform multi-class classification is to expand the dimension of the target vector, from 2 to 4, in our case.

The confusion matrix and the accuracy, recall and precision is reported in the following chart:

|  |  |  |  |
| --- | --- | --- | --- |
| Confusion Matrix | [137, 161, 72, 22]  [109, 127, 105, 44]  [ 79, 45, 204, 62]  [ 97, 18, 56, 227] | Accuracy | 0.444089456869 |
| Recall | 0.444089456869 |
| Precision | 0.449276631965 |

***One Vs One SVM Multiclass Classifier***:

The OVO-svm classifier is based on performing the 2-way classification on each of the two pairs of targets and obtaining the one target with the highest vote. Upon this problem, we used the OneVsOneClassifier Object provided from the sklearn.multiclass library and applied it to the LinearSVC() as :

ovo\_classifier\_i = OneVsOneClassifier (LinearSVC())

The results are listed in the following chart:

|  |  |  |  |
| --- | --- | --- | --- |
| Confusion Matrix | [156, 175, 59, 2]  [145, 94, 136, 10]  [ 37, 37, 310, 6]  [ 5, 10, 35, 348] | Accuracy | 0.580191693291 |
| Recall | 0.580191693291 |
| Precision | 0.571965590504 |

***One Vs Rest SVM Multiclass Classifier***:

Unlike the OVO-svm classifier, the OVR-svm classifier is based on fitting one classifier to each of the classes, and obtaining the one class with the most appropriate fit. For this problem, we used the OneVsRestClassifier Object from the sklearn.multiclass library, and applied it to the Linear SVM as:

ovr\_classifier\_i = OneVsRestClassifier (LinearSVC())

The results are listed in the following chart:

|  |  |  |  |
| --- | --- | --- | --- |
| Confusion Matrix | [149, 178, 60, 5]  [124, 94, 140, 27]  [ 37, 35, 308, 10]  [ 4, 5, 28, 361] | Accuracy | 0.582747603834 |
| Recall | 0.582747603834 |
| Precision | 0.563982043464 |