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

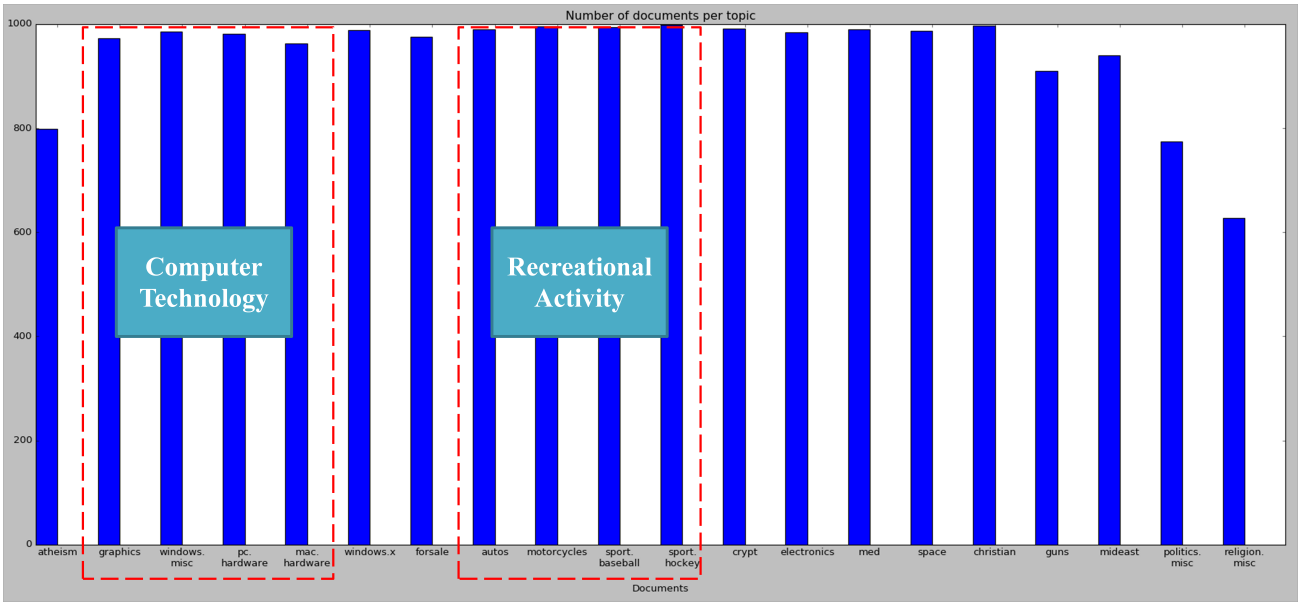
There are two ways to solve this problem:

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), LSI feature can be generated using following code:

```
from sklearn.decomposition import TruncatedSVD
X = TruncatedSVD(n_components=50, algorithm='arpack')
LSI = X.fit_transform(TFxIDF_data)
```

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)
```

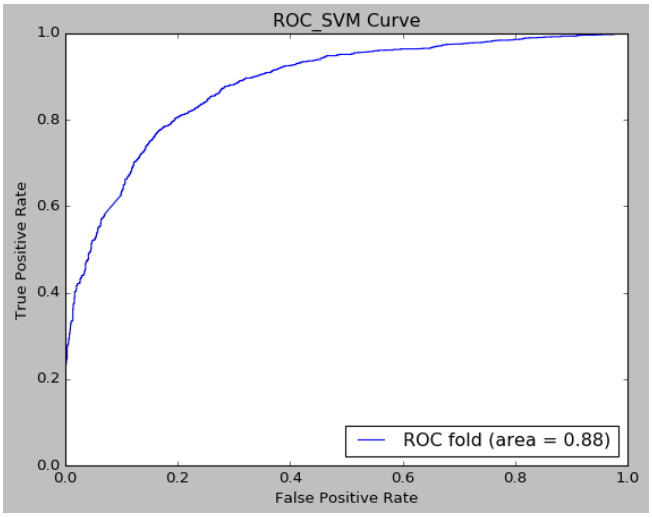
After training the linear SVM classifier, use the generated classifier lin_clf to predict test data:

```
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	[1322, 238] [393, 1197]	Accuracy	0.799682539683
		Recall	0.752830188679
		Precision	0.834146341463

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.cross_validation import KFold
kf = KFold(n=len(twenty_train_target), n_folds=5, shuffle=False, random_state=None)
for train_index, test_index in kf:
    X_train_soft, X_test_soft = twenty_train_data[train_index], twenty_train_data[test_index]
```

```
for k in [-3, -2, -1, 0, 1, 2, 3]:
 soft_svm_clf = svm.LinearSVC(C=10**k)
    soft_svm_clf.fit(X_train, X_train_target_group)
    scores[i][j]= soft_svm_clf.score(X_test, X_test_target_group)
    j=j+1
i=i+1
j= 0
```

Scores is a two-dimensional array that stores 5*7 scores with different lambda values (each lambda value has 5-fold cross validation), then we need to find the lambda with average largest scores:

```
ave_score = list(map(lambda x: (x[0]+x[1]+x[2]+x[3]+x[4])/5, zip(scores[0], scores[1], scores[2], scores[3], scores[4])))
max_score = max(ave_score)
index=ave_score.index(max_score)
r = [-3, -2, -1, 0, 1, 2, 3]
print ('The best lambda is',10**(-r[index]))
```

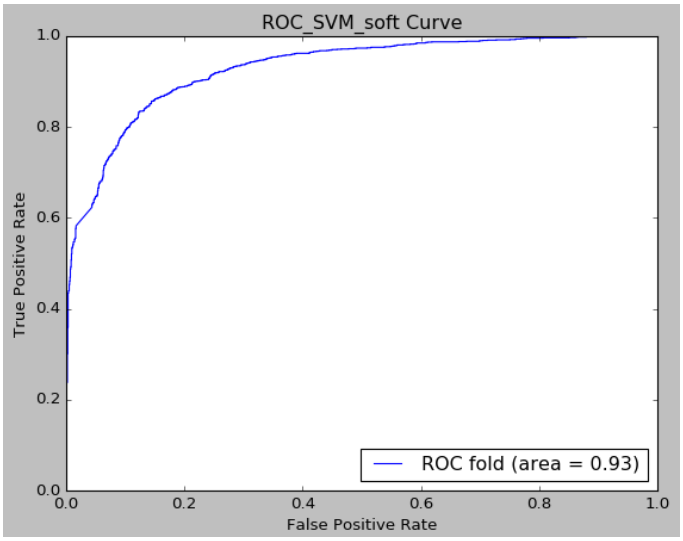
Result shows that when k = 2, we get the best parameter lambda, which is 0.01.

Then generate soft margin SVM classifier:

```
soft_clf = svm.LinearSVC(C=10**-2)

soft_clf.fit(train_LSI, train_target_group)
```

The ROC curve of soft margin SVM classifier is:



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

Confusion Matrix	[1331, 229] [228, 1362]	Accuracy	0.854920634921
		Recall	0.856603773585
		Precision	0.856065367693

Question.g

In this problem, we use naive_bayes package to train the classifier and predict test data. In order to obtain a better performance of classification, we compared the Gaussian Naive Bayes classifier with the Bernoulli Naive Bayes classifier, whose performance is given in the following charts. The difference between the two classifiers lies in the assumption of likelihood of features.

GaussianNB() implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

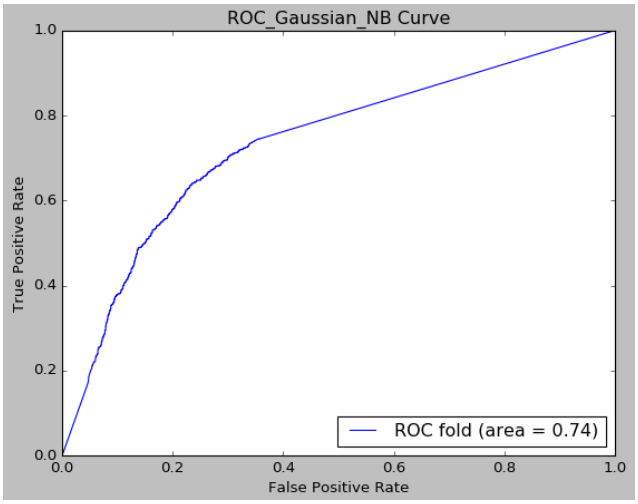
BernoulliNB() implements the Bernoulli Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Bernoulli:

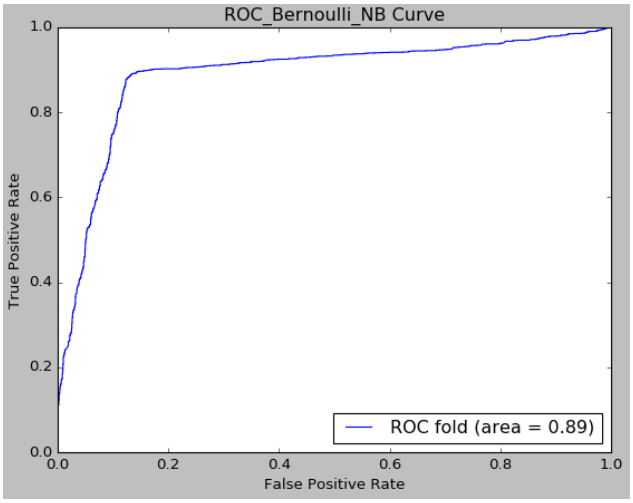
$$P(x_i \mid y) = P(i \mid y)x_i + (1 - P(i \mid y))(1 - x_i)$$

The main code is shown below:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
gnb_clf = GaussianNB()
bnb_clf = BernoulliNB()
gnb_clf.fit(train_LSI, train_target_group)
bnb_clf.fit(train_LSI, train_target_group)
gnb_predicted = gnb_clf.predict(test_LSI)
bnb_predicted = bnb_clf.predict(test_LSI)
```

The ROC curve of two naive Bayes classifiers are shown:





The results of two naive Bayes classifiers are shown in below tables:

Gaussian Naïve Bayes Classifier:

Confusion Matrix	[1257, 303] [686, 904]	Accuracy	0.686031746032
		Recall	0.568553459119
		Precision	0.748964374482

Bernoulli Naïve Bayes Classifier:

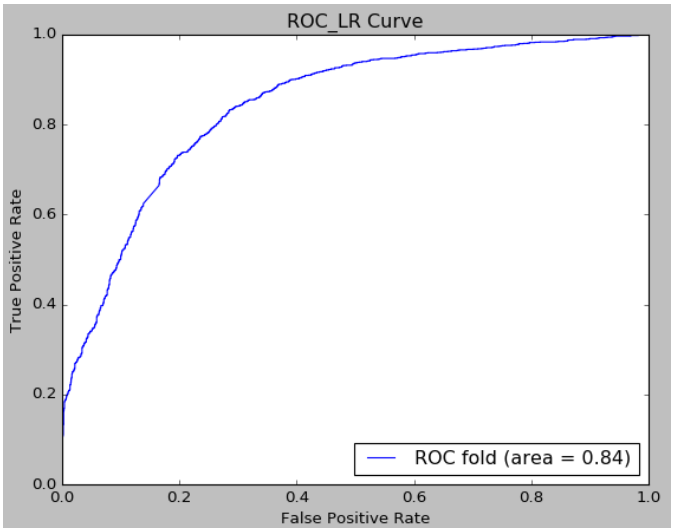
Confusion Matrix	[1366, 194] [202, 1388]	Accuracy	0.874285714286
		Recall	0.872955974843
		Precision	0.877370417193

Question.h

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	[1270, 290] [466, 1124]	Accuracy	0.76
		Recall	0.706918238994
		Precision	0.794908062235

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 of the Gaussian Naive Bayes classifier and the Bernoulli Naive Bayes classifier is reported in the following charts, respectively.

Gaussian Naïve Bayes Multiclass Classifier:

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

Bernoulli Naïve Bayes Multiclass Classifier:

Confusion Matrix	[163, 168, 54, 7] [200, 86, 80, 19] [63, 22, 293, 12] [18, 26, 47, 307]	Accuracy	0.54249201278
		Recall	0.54249201278
		Precision	0.542353955084

As shown, the Bernoulli Naive Bayes classifier provides better overall classification performance.

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