Project 1

Members:
Daphne Chiou
Nien-Jen Cheng
Hongyang Li

Files included in the folder:
ECE219Project01.ipynb
Project01_Report.pdf

Prepare features

(a)

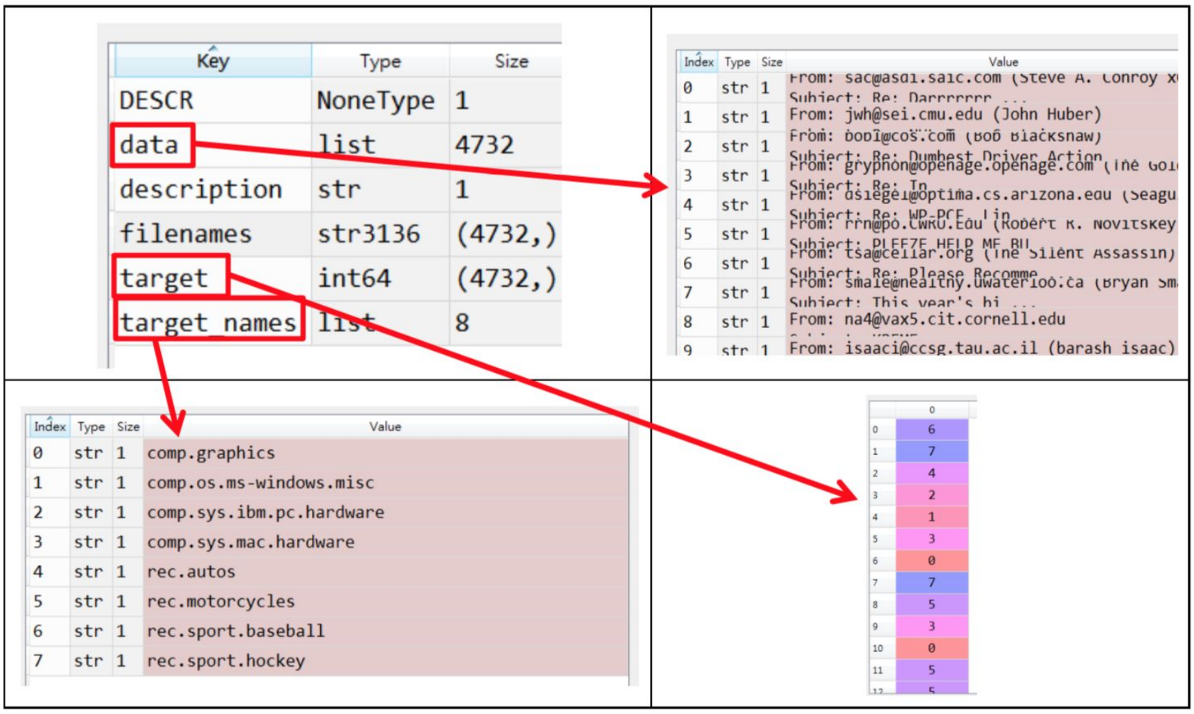
In this project, we use the data set, “20newsgroups.” We can fetch the data by the function “fetch_20newsgroups().” First of all, there are “train data” and “test data” in the data base. However, the function can only fetches one of the kinds in one time. Hence, we wrote a function to fetch both “train” and “test” data at once.

```
def fetch_data(categories):
        return fetch_20newsgroups(subset='train', categories=categories,
shuffle=True, random_state=42), \
    fetch_20newsgroups(subset='test', categories=categories, shuffle=True,
random_state=42)
```

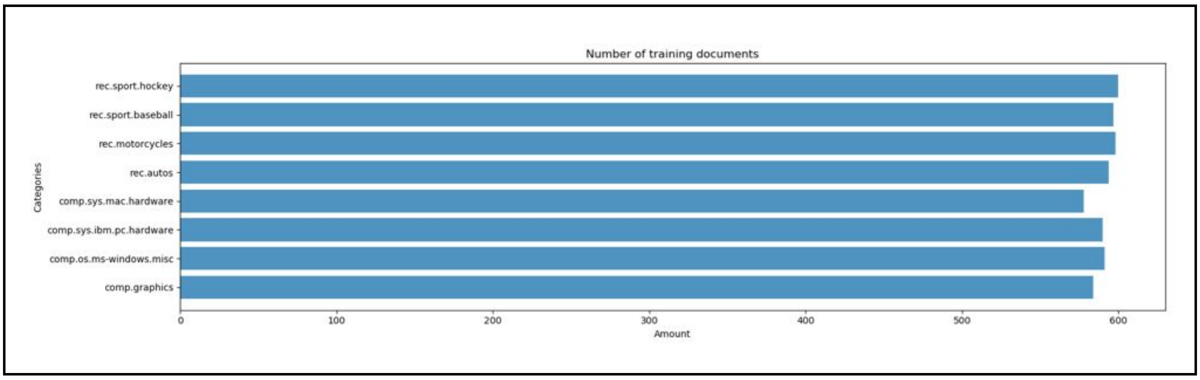
How to use:

```
twenty_train, twenty_test = fetch_data(categories)
```

Both “twenty_train” and “twenty_test” contain 6 lists. The “data” is the list of the articles. There are 4732 and 3150 documents in the “data” of “twenty_train” and “twenty_test.” The target shows the indices of the categories correspond to the articles in the “data”, and target_names shows the categories correspond to the indices in “target.”

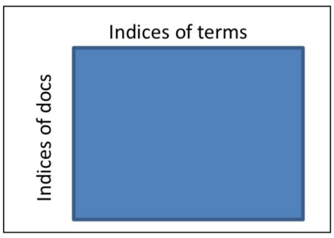


In section a, we are asked to provide the numbers of the documents of each categories in twenty_train. The histogram is shown below. We can see that the distribution of among the categories is quite balance.



(b-1)

In this section, we want to build an array that counts the frequencies of the terms in each document for analyzing. Take twenty_train as an example, there are 4732 docs in the data, and assume that there are T kinds of terms, the array is a (4732 X T) one shown below.



However, there were too many kinds of terms in the docs. Some terms are not important while comparing and analyzing the docs. We can filter some terms by the function “CountVectorizer()” with some attributions shown below.

1. min_df : Filter terms that appears only min_df times.
2. stop_words: Filter some common terms like “the, a, is….” In our project, we use the word set, “ENGLISH_STOP_WORDS”

The operation is shown below.

For training data:

```
count_vect = CountVectorizer(min_df=min_df, stop_words=stop_words)
X_train_counts = count_vect.fit_transform(train.data)
```

For testing data:

```
count_vect = CountVectorizer(min_df=min_df, stop_words=stop_words)
X_test_counts = count_vect.transform(test.data)
```

Furthermore, we excluded stemmed version of terms. (Ex “ goes, going, went” are stemmed versions of the term, “go”)

We made all things above into a function shown below.

```
def Stemmer(Data):
	for i in range(len(Data)):
	Data[i] = ' '.join(map(SnowballStemmer("english").stem, 
CountVectorizer().build_analyzer()(Data[i])))

def Counter(min_df, train, test):
	# The default regexp select tokens of 2 or more alphanumeric characters
	# punctuation is completely ignored and always treated as a token separator

	Stemmer(train.data)
	Stemmer(test.data)

	count_vect = CountVectorizer(min_df=min_df, stop_words=stop_words)
	X_train_counts = count_vect.fit_transform(train.data)
	XX_train  = X_train_counts.toarray()
	print("Shape of train =",XX_train.shape)
	X_test_counts = count_vect.transform(test.data)
	XX_test  = X_test_counts.toarray()
	print("Shape of test =",XX_test.shape)
	# here
	return X_train_counts, X_test_counts
```

How to use:

```
X_train_counts_2, X_test_counts_2 = Counter(2, twenty_train, twenty_test)
X_train_counts_5, X_test_counts_5 = Counter(5, twenty_train, twenty_test)
```

The outputs are sparse arrays. We can transform them to dense arrays later. EX: (XX_train = X_train_counts.toarray())

After we filtered the doc-data by min_df = 2 & 5, the kinds of terms were reduced to the numbers shown below.

	train	test
min_df = 2	25434	25434
min_df = 5	10670	10670

(b-2)

The next step is to transform our doc-term frequency arrays to a TFxIDF arrays. This transform normalizes the importance of the terms by documents. EX: Terms like "computer, software" present almost everywhere in the docs in "pc hardware" categories. These terms will be transform less important. We count how many documents have a certain term (Call it df(t)). The IDF is defined as $idf(t) = \log[n / df(t)] + 1$. Then the TFxIDF array is defined as $TFxIDF(t,d) = tf(t,d) * idf(t)$ This can be done by the function shown below. The function return sparse arrays, too.

```
def tfidf(X_train_counts, X_test_counts):
        from sklearn.feature_extraction.text import TfidfTransformer
        tfidf_transformer = TfidfTransformer()
        X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
        X_test_tfidf = tfidf_transformer.transform(X_test_counts)
        return X_train_tfidf, X_test_tfidf
```

How to use:

```
X_train_tfidf_2, X_test_tfidf_2 = tfidf(X_train_counts_2, X_test_counts_2)
X_train_tfidf_5, X_test_tfidf_5 = tfidf(X_train_counts_5, X_test_counts_5)
```

Due to the arrays are too large to print in this report, we only show part of the numbers in the arrays.

TFIDF Arrays:

(min_df = 2, train)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0.0534985	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.11498	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0.152658	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(min_df = 2, test)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0.0534985	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.11498	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0.152658	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(min_df = 5, train)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0.0534985	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.11498	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0.152658	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(min_df = 5, test)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0.055318	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0.0609665	0	0	0	0	0	0	0	0	0	0	0	0	0.0688038	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0727257	0	0	0.0730044	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0.170038	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0997344	0	0	0	0	0.0896216	0	0	0.163645	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(c)

In this section, instead of using TFIDF, which normalize the frequencies by counting how many docs have a certain term, we normalize the frequencies by counting how many categories have a certain term. We can use the same function, "TfidfTransformer()" to apply the transform. However, we need to map the doc-term arrays to categories arrays first. Also, because we need to list the "top words" later, we need to recover the word lists, which map the indices of terms to the terms, vocabularies. The function below returns TF array and word list.

After we got the TFICF arrays, in each row, we sorted the columns based on the frequencies. We listed the top 10 terms with their indices. Finally, we use the word lists to convert the indices to terms. The top 10 words among the 20 categories are:

min_df =5									
Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
edu	imag	ax	scsi	edu	window	edu	car	bike	edu
atheist	edu	max	edu	line	use	00	edu	com	game
god	jpeg	b8f	drive	mac	edu	line	com	edu	line
islam	line	g9v	line	subject	line	subject	line	line	organ
write	graphic	a86	com	organ	com	sale	subject	subject	subject
com	file	window	ide	use	widget	organ	organ	organ	year
subject	use	34u	subject	appl	subject	post	write	dod	pitcher
atheism	subject	edu	use	quadra	file	univers	articl	write	com
say	organ	75u	organ	post	motif	com	ani	articl	write
line	program	1d9	card	problem	xterm	new	post	motorcycl	articl
Class 10	Class 11	Class 12	Class 13	Class 14	Class 15	Class 16	Class 17	Class 18	Class 19
hockey	encrypt	edu	edu	edu	god	edu	armenian	edu	edu
edu	key	line	com	space	christian	gun	isra	stephanopoulo	god
game	com	use	organ	orbit	edu	com	turkish	com	com
nhl	clipper	subject	subject	line	church	firearm	israel	peopl	christian
team	use	organ	line	nasa	subject	peopl	edu	cramer	jesus
playoff	escrow	com	use	subject	jesus	line	jew	optilink	sandvik
line	edu	write	articl	organ	homosexu	write	muslim	write	subject
play	line	articl	candida	shuttl	peopl	organ	arab	articl	write
subject	chip	post	ani	write	line	subject	peopl	line	line
ca	subject	ani	write	launch	sin	fbi	armenia	think	organ

We are interested in [comp.sys.ibm.pc.hardware, comp.sys.mac.hardware, misc.forsale, soc.religion.christian] They are target [3,4,6,15]

Here is the list. The terms are reasonable with their categories.

comp.sys.ibm.pc.hardware	comp.sys.mac.hardware	misc.forsale	soc.religion.christian
scsi	edu	edu	god
edu	line	00	christian
drive	mac	line	edu
line	subject	subject	church
com	organ	sale	subject
ide	use	organ	jesus
subject	appl	post	homosexu
use	quadra	univers	peopl
organ	post	com	line
card	problem	new	sin

min_df =2									
Class 0 edu atheist god islam write com subject atheism say line	Class 1 imag edu jpeg line graphic file use subject organ program	Class 2 ax max b8f g9v a86 window 34u edu 75u 1d9	Class 3 scsi edu drive line com ide subject use organ card	Class 4 edu line mac subject organ use appl quadra post problem	Class 5 window use edu line com widget subject file motif xterm	Class 6 edu 00 line subject sale organ post univers com new	Class 7 car edu com line subject organ write articl ani post	Class 8 bike com edu line subject organ dod write articl motorcycl	Class 9 edu game line organ subject year pitcher com write articl
Class 10 hockey edu game nhl team playoff line play subject ca	Class 11 encrypt key com clipper use escrow edu line chip subject	Class 12 edu line use subject organ com write articl post ani	Class 13 edu com organ subject line use articl candida ani write	Class 14 edu space orbit line nasa subject organ shuttl write launch	Class 15 god christian edu church subject jesus homosexu peopl line sin	Class 16 edu gun com firearm peopl line write organ subject fbi	Class 17 armenian isra turkish israel edu jew muslim arab peopl armenia	Class 18 edu stephanopoulo com peopl cramer optilink write articl line think	Class 19 edu god com christian jesus sandvik subject write line organ

By using the threshold of min_df = 2, we got the same result.

comp.sys.ibm.pc.hardware	comp.sys.mac.hardware	misc.forsale	soc.religion.christian
scsi edu drive line com ide subject use organ card	edu line mac subject organ use appl quadra post problem	edu 00 line subject sale organ post univers com new	god christian edu church subject jesus homosexu peopl line sin

(d)

In the section (b), we recovered the tfidf arrays. However, there dimensions of them are too big that is bad for calculations. Before we train a model, we need to find good features. Terms frequencies array is not a good feature, because the density of the array is too low. It has zeros everywhere. What we need is to transform the array to a new coordinate that the data are concentrated at the first few entries. In this section, we use 2 methods, SVD and NMF to achieve it.

We print the first rows of "Training data" of "Testing data" with the two methods below.
min_df = 2

```
train
SVD
[ 1.29802119e-01  9.59930799e-02  2.90093939e-02  8.61313106e-03
 -6.23723541e-02 -7.12253481e-03 -9.08407606e-02  5.76950366e-02
 -5.64593525e-03 -3.94538368e-02  7.64642144e-03  2.02681920e-03
  2.31399119e-02  4.18087701e-02  2.77506690e-02 -1.40066832e-05
  1.01503993e-02 -9.25783487e-04 -1.59918584e-02 -3.65028930e-02
  6.71019375e-03  9.41947922e-03 -1.99420451e-02 -1.78236579e-02
 -2.29350997e-02 -2.91717256e-03  1.96736674e-02 -3.10914406e-02
  1.81886428e-02  4.90438949e-02 -2.16336014e-02  2.39709142e-02
  1.24417558e-02 -1.46635864e-02  2.43753111e-02 -1.75813494e-02
  3.07078092e-02  1.25067837e-02  1.26004159e-02 -6.29683898e-02
 -3.37391242e-02 -5.92107647e-02  1.05981965e-02 -1.92821490e-02
 -4.87261749e-03  1.23796101e-02 -1.74767179e-02 -1.70559434e-02
 -2.95082643e-02  1.15266045e-04]


NMF
[6.65189778e-02 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 8.08333262e-04 1.67852603e-04 3.27185421e-03
 8.86381559e-06 0.00000000e+00 0.00000000e+00 9.14291886e-03
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 4.27243501e-04
 4.67445323e-03 8.63636147e-03 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 2.03605318e-03
 2.33881842e-02 0.00000000e+00]
```

```
test
SVD
[ 0.11812855 -0.07563236  0.03601262  0.00178144  0.04812986  0.13759157
 -0.01370726 -0.04407133 -0.07019848 -0.08685676  0.00928956 -0.08250755
  0.16989523  0.01727924 -0.00783357 -0.03714927 -0.16437189  0.02078063
  0.10452952  0.02554394 -0.09339208  0.08384348 -0.11470264  0.10719673
  0.07229131  0.00413435  0.07162597  0.11713201 -0.08057904 -0.07639217
 -0.08294696  0.00881533  0.0427976   0.10789585  0.17621721 -0.01980751
  0.03449562  0.05524726  0.05649021  0.0912478  -0.03570298  0.07570317
  0.08258577  0.02160376  0.00848533  0.01528173 -0.00958437 -0.02579994
 -0.06480631 -0.01943209]


NMF
[0.00000000e+00 4.05572033e-01 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 3.18473715e-04 0.00000000e+00
 0.00000000e+00 0.00000000e+00 1.83992967e-02 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 4.15037258e-03 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00]
```

Training model

In section (a) to (d), we were focusing on preparing the training data, or "features" to fit in the classifiers (which we will cover in the following sections). In section (e) to (i), we implemented the data on binary classifications. We indicate an article as two of the chosen categories. The multiclass classification will be implemented in section (j).

The input data are the features retrieved from the articles, W_train, W_test. The output data are the indices of the categories. let's call them "Y's". However, we are using binary classifiers at this stage, we need to translate the indices, two parts of "0,1,2,3,4,5,6,7…..", for example "3, 15, 6" and "7,12,19" to "0" and "1" so that classifiers like SVM can understand.

In our project, we want to separate articles in to two categories. As shown below. Each category has several sub-categories.

Computer Technology Label = "0"
Com_Tech = ['comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware']
Recreational Activity Label = "1"
Rec_Act = ['rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey']

We wrote a function to reassign the output labels. As shown below.

For Y training

```
train_Y = []
zeros, ones = 0, 0
for i in range(len(twenty_train.target)):
    if twenty_train.target_names[twenty_train.target[i]] in Com_Tech:
        train_Y.append(0)
        zeros += 1
    if twenty_train.target_names[twenty_train.target[i]] in Rec_Act:
        train_Y.append(1)
        ones += 1
```

For Y testing

```
test_Y = []
zeros, ones = 0, 0
for i in range(len(twenty_test.target)):
    if twenty_test.target_names[twenty_test.target[i]] in Com_Tech:
        test_Y.append(0)
        zeros += 1
    if twenty_test.target_names[twenty_test.target[i]] in Rec_Act:
        test_Y.append(1)
        ones += 1
```

Numbers of articles in each set are shown below. The numbers of documents between the classes are quite balance.

	Computer Technology	Recreational Activity
Train	2343	2389
Test	1560	1590

After We prepared the Y's, the labels, we need to prepare tools that monitor our performance. In SVM, the machine separate two kinds of documents by a boundary (called threshold) in the feature space. Different threshold yield different sets of TPR and FPR.

TPR = TP/(TP+FN)
FPR = FP/(FP+TN)

TP, FP, FN, TN are defined in the confusion matrix shown below

	They are Positive	They are Negative
We predict Positive	True Positive(TP) 	False Positive(FP) 
We predict Negative	False Negative(FN) 	True Negative(TN) 

TPR, Recall, FPR, Accuracy, Precision(Positive predictive value) are defined below.

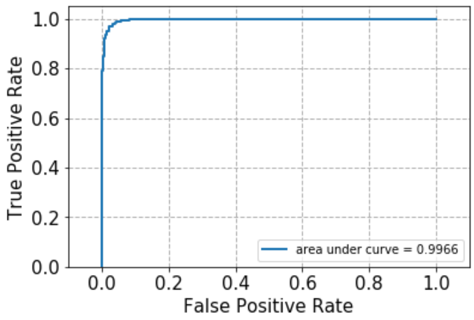
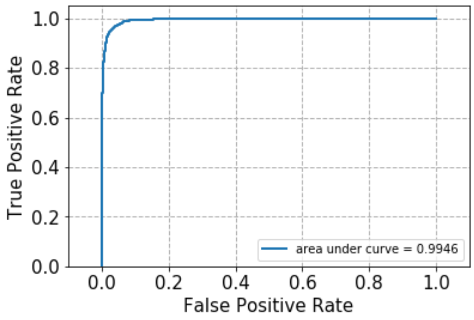
TPR = Recall =  ------------------------------  + 	FPR =  ------------------------------------  + 
Accuracy =  +  ---------------------------------- Total	Prececion =  --------------------------------  + 

A good classifier can have high TPR while FPR is low. We will plot TPR vs FPR (called ROC) in the following sections to find a model is good or not. The more the area under the curve, the better the model. Also, the accuracy gives the general performance of a model. We mainly judge the good or bad from this.

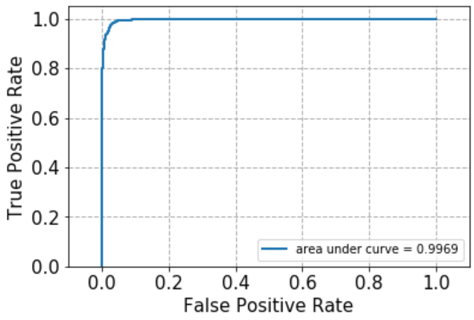
(e)

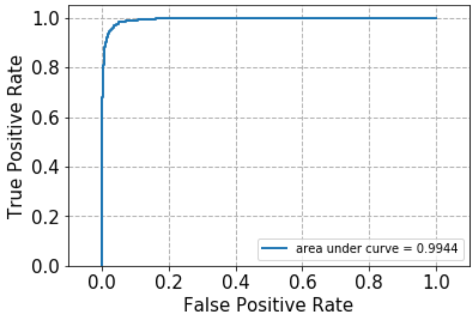
To begin with, we apply both Hard margin SVM and Soft margin SVM on the dataset. We use models trained on training dataset with different training attributes to predict the testing dataset.
By setting γ = 1000, we assume the margin is large enough and will perform like a Hard margin SVM.
The same trick is used on Soft margin SVM, where we set the γ = 0.001.

min_df=2, LSI

	ROC curve	confusion matrix	accuracy	recall	precision
Hard margin SVM (γ = 1000)		[[1495 65] [27 1563]]	0.9708	0.983	0.9601
Soft margin SVM (γ = 0.001)		[[1362 198] [8 1582]]	0.9346	0.9950	0.8888

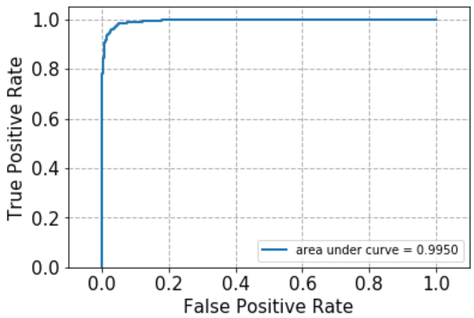
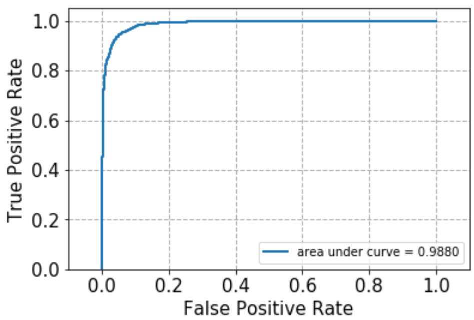
min_df=5, LSI

	ROC curve	confusion matrix	accuracy	recall	precision
Hard margin SVM (γ = 1000)		[[1423 137] [7 1583]]	0.9543	0.9956	0.9203

Soft margin SVM (γ = 0.001)		[[1373 187] [9 1581]]	0.9378	0.9943	0.8942

- Hard margin SVM performs better than Soft margin SVM.
- The accuracy with min_df=5 and min_df=2 were close, which implies that filtering the corpus with min_df=2 and min_df=5 would results in similar informational arrays. Tuning the threshold is more important than choosing between the two features.
- The more the areas under the curve, the better the performance. Based on the observation of the ROC curves, we retrieve quite good results with both margins.

min_df=2, NMF

	ROC curve	confusion matrix	accuracy	recall	precision
Hard margin SVM (γ = 1000)		[[1407 153] [15 1575]]	0.9467	0.9906	0.9115
Soft margin SVM (γ = 0.001)		[[9 1551] [0 1590]]	0.5076	1.0	0.5062

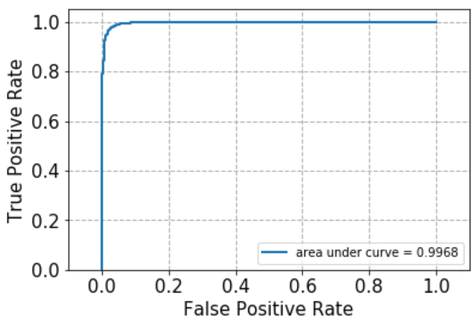
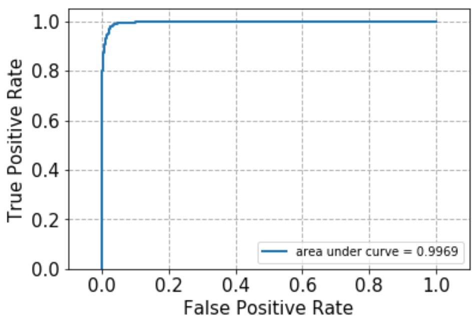
- Results with NMF reduction had bad result with soft margin. We can see that the feature generated by the NMF are more sensitive to the γ. When γ = 0.001, the margin is too flexible to distinguish two classes.

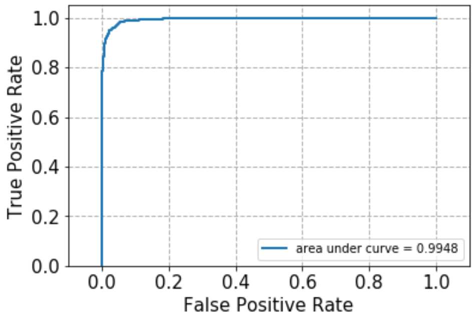
(f)

In this part, we use cross-validation to further determine the best value of k. That is, to decide which margin width would classify the two classes most neatly. The experiments are done on values ranging from -3 to 3.
Note that γ = 10^k

k/ accuracy	-3	-2	-1	0	1	2	3
min_df=2 LSI	0.9364	0.9683	0.9719	0.9744	0.9765	0.9751	0.9715
min_df=5 LSI	0.9400	0.9656	0.9706	0.9730	0.9751	0.9746	0.9675
min_df=2 NMF	0.5116	0.8229	0.9529	0.9609	0.9660	0.9690	0.9613

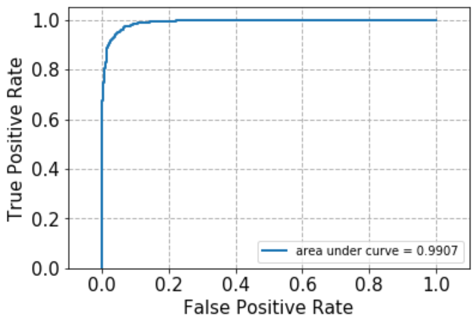
- LSI with min_df=2 and min_df=5 have the same result K=1, while NMF has that at K=2.
- In general, LSI perform better than NMF.

	ROC curve	confusion matrix	accuracy	recall	precision
min_df=2, LSI, K = 3 (γ = 1000)		[[1505 55] [30 1560]]	0.9765	0.9811	0.9659
min_df=5, LSI, K = 2 (γ = 100)		[[1507 53] [29 1561]]	0.9739	0.9817	0.9671

min_df=2, NMF, K = 3 (γ = 1000)		[[1492 68] [52 1538]]	0.9619	0.9672	0.9576

(g)

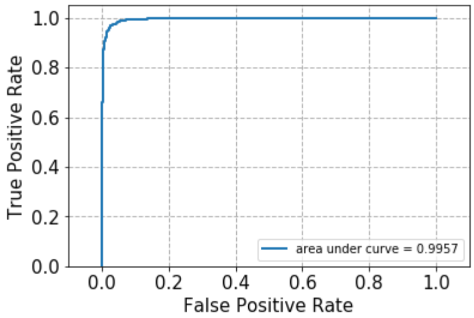
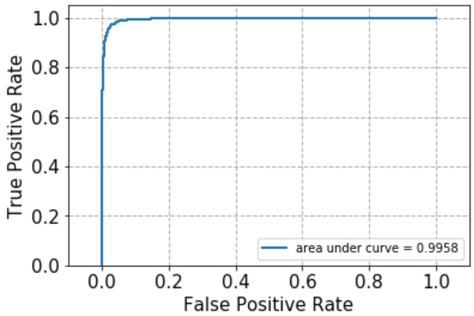
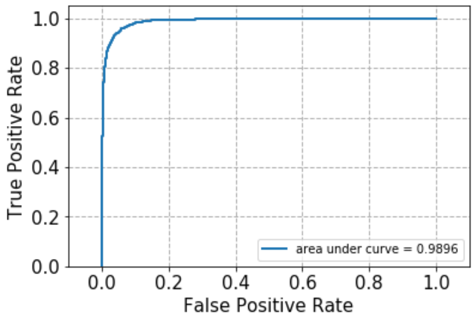
Next, we use Naive Bayes Model trained on training dataset to predict the testing dataset. Since Naive Bayes does not allow negative value in the data representation, we use array using NMF reduction as input.

	ROC curve	confusion matrix	accuracy	recall	precision
min_df=2, NMF		[[1384 176] [21 1569]]	0.9375	0.9867	0.8991

- The results was 93.75%, which is only slightly worse than using SVM under the same condition.

(h)

From the above sections, we have discovered SVM and Naive Bayes models. We further experiment on Logistic Regression, where three models are trained with different attributes.

	ROC curve	confusion matrix	accuracy	recall	precision
min_df=2, LSI		[[1489 71] [33 1557]]	0.9670	0.9792	0.9563
min_df=5, LSI		[[1493 67] [28 1562]]	0.96984	0.9824	0.9589
min_df=2, NMF		[[1441 119] [50 1540]]	0.9463	0.9686	0.9283

Accuracies			
	SVM Best γ	Naive Bayes	Logistic
min_df=2, LSI	0.9765	-	0.9670
min_df=5, LSI	0.9751	-	0.96984
min_df=2, NMF	0.9690	0.9375	0.9463

- Aagain, the LSI has better result in Logistic
- We paste the previous results with the ones in this section. We see that Logistic is not better than SVM, but slightly better than Naive Bayes.

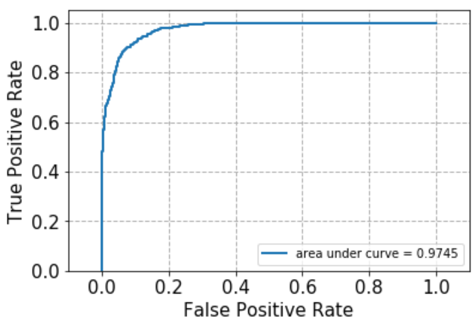
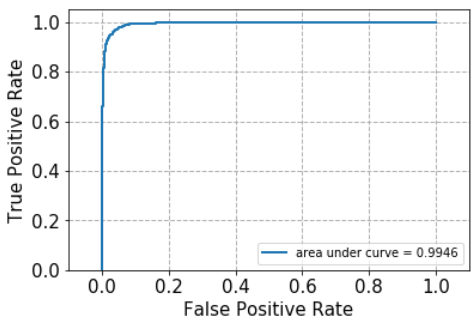
(i)

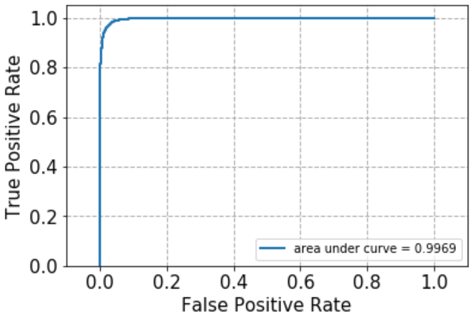
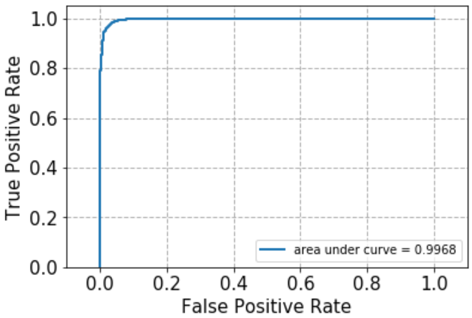
Next, we dive into Logistic Regression a little bit more by adding regularization terms L1 and L2.
L1 is known as least absolute errors, which minimizes the absolute differences between the target values and the estimated values, whereas L2 is equivalent to least squares errors, which minimizes the squares errors instead.

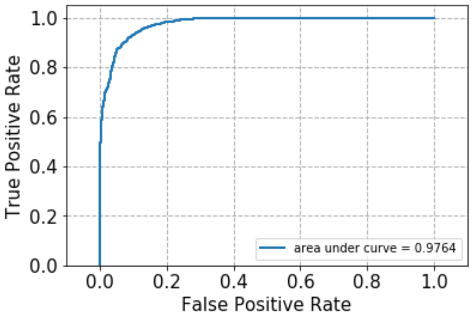
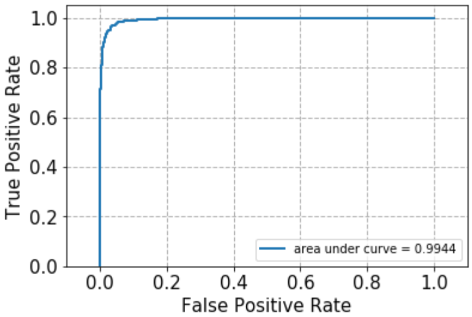
L1: $$S = \sum_{i=1}^{n} |y_i - f(x_i)|.$$

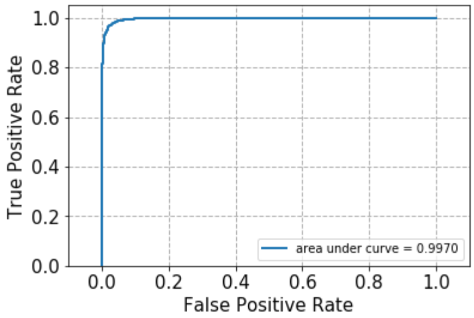
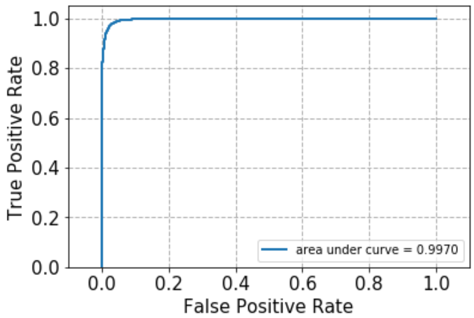
L2: $$S = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

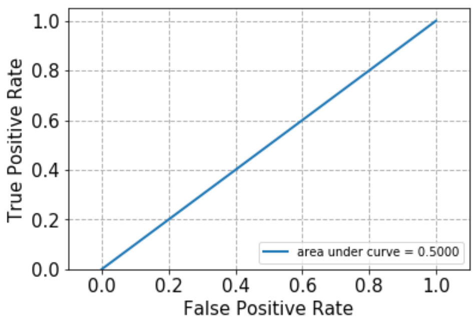
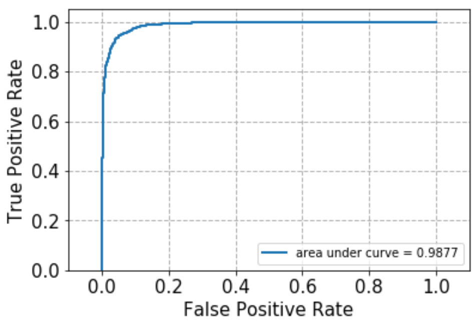
The C term determines the strength of regularization. Here we use 100 to imitate strong regularization and 0.01 to imitate weak regularization. Experiment both strength on L1 and L2.

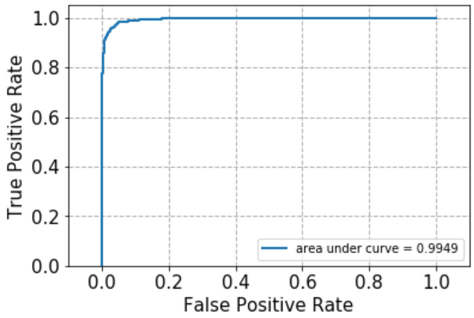
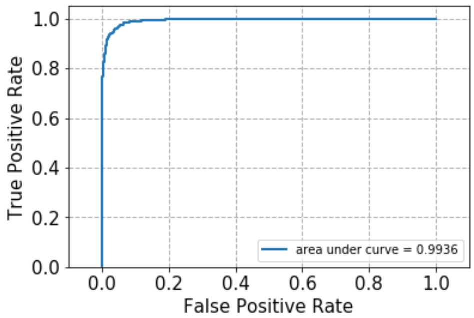
LSI	min_df=2				
	ROC curve	confusion matrix	accuracy	recall	precision
LSI, l1, C=0.01		[[1477 83] [227 1363]]	0.9015	0.857	0.9426
LSI, l2, C=0.01		[[1385 175] [9 1581]]	0.9415	0.9943	0.9003

LSI, l1, C=100		[[1506 54] [30 1560]]	0.9733	0.9811	0.9665
LSI, l2, C=100		[[1504 56] [29 1561]]	0.9730	0.9817	0.9653

LSI	min_df=5				
	ROC curve	confusion matrix	accuracy	recall	precision
LSI, l1, C=0.01		[[1450 110] [159 1431]]	0.9146	0.9	0.9286
LSI, l2, C=0.01		[[1388 172] [14 1576]]	0.9409	0.9911	0.901

LSI, l1, C=100		[[1503 57] [32 1558]]	0.9717	0.9798	0.9647
LSI, l2, C=100		[[1505 55] [28 1562]]	0.9736	0.9823	0.9659

NMF min_df=2					
	ROC curve	confusion matrix	accuracy	recall	precision
NMF, l1, C=0.01		[[1560 0] [1590 0]]	0.4952	0	0
NMF, l2, C=0.01		[[16 1544] [0 1590]]	0.5098	1.0	0.5073

NMF, l1, C=100		[[1498 62] [55 1535]]	0.9628	0.9654	0.9611
NMF, l2, C=100		[[1481 79] [49 1541]]	0.959	0.9691	0.9512

After we have done the experiments, we are ready to answer some questions.

How does the regularization parameter affect the test error?
First, we can see that L1 and L2 norms provide similar result. But sometimes, L1 gives bad ROC curve with NMF. Just like we mentioned before, NMF is sometimes sensitive. Within the same norm, for both LSI and NMF reductions, a larger regularization parameter C results in a higher accuracy.

How are the coefficients of the fitted hyperplane affected?
Also, as the regularization parameter increases, the fitted plane move away from the origin.

Why might one be interested in each type of regularization?
The performance of each type of regularization highly depends on the characteristic of the data representation. The L1 regularization needs the number of samples that increase logarithmically in the number of irrelevant features. As for the L2's, it increases linearly. Therefore, experiment on a variety of regularization terms may help find the best model.

(j)

In this section, we want to predict the categories of documents among multiple ones. For Naive Bayes, we find the labels correspond to their maximum likelihood. However, since SVC is a binary classifier, so we could conduct the predictions one vs one or one vs rest to convert it into a multiple classifier. A faster way to do it is to use the one vs rest way. We performed both methods for SVC in this section.

The categories we our interested in are the following 4 again.

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

(1) Naive Bayes

	confusion matrix	accuracy	recall	precision
min_df=2, NMF	[[331 25 36 0] [114 218 51 2] [38 12 337 3] [3 1 2 392]]	0.8166	[0.84438776 0.56623377 0.86410256 0.98492462]	[0.68106996 0.8515625 0.79107981 0.98740554]

- Since the Naive Bayes Model does not accept data representation with LSI reduction, we have no choice but to use NMF reduction instead, which performs poorer than the others as it does in binary classification.

(2) multiclass SVM
One vs One

	confusion matrix	accuracy	recall	precision
min_df=2, LSI	[[324 44 24 0] [50 314 21 0] [24 14 351 1] [5 2 4 387]]	0.8792	[0.82653061 0.81558442 0.9 0.97236181]	[0.80397022 0.83957219 0.8775 0.99742268]
min_df=2, NMF	[[311 62 19 0] [72 289 23 1] [31 29 330 0] [10 13 6 369]]	0.8300	[0.79336735 0.75064935 0.84615385 0.92713568]	[0.73349057 0.73536896 0.87301587 0.9972973]

One vs Rest

	confusion matrix	accuracy	recall	precision
min_df=2, LSI	[[315 52 24 1] [38 320 25 2] [21 14 354 1] [4 1 2 391]]	0.8817	[0.80357143 0.83116883 0.90769231 0.98241206]	[0.83333333 0.82687339 0.87407407 0.98987342]
min_df=2, NMF	[[308 58 26 0] [66 284 33 2] [23 21 343 3] [4 4 6 384]]	0.8428	[0.78571429 0.73766234 0.87948718 0.96482412]	[0.7680798 0.77384196 0.84068627 0.98714653]

- The data representation with LSI reduction has significantly better results, 87~88%, than that of NMF reduction, 83~84%.
- However, the recall and precision values vary among different classes, suggesting that the model tends to predict some of the classes over the rest of them.
- Overall, the One vs One and the One vs Rest strategies have the similar results in this case.