Project 1 Report
Classification Analysis on Textual Data

Yutong Han, 705025619
Ziye Xing, 704778088
Jan. 30th, 2018

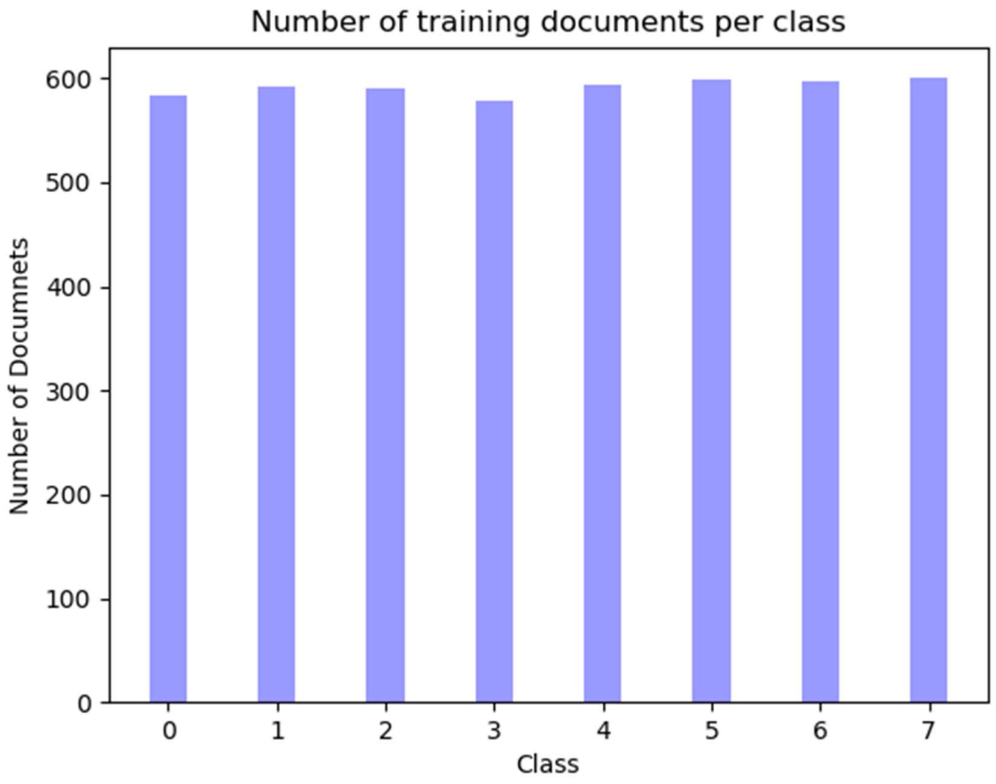
Dataset and Problem Statement

Task a

The 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. The data is organized into 20 different newsgroups, each corresponding to a different topic. Some of the newsgroups are very closely related to each other (e.g. **comp.sys.ibm.pc.hardware** / **comp.sys.mac.hardware**), while others are highly unrelated (e.g **misc.forsale** / **soc.religion.christian**)

The first thing here is that we should check if the training data set is evenly distributed. In order to do that, we count number of the training document per class, and plot the histogram to visualizate that.

Class	Number of training documents
`comp.graphics`	584
`comp.os.ms-windows.misc`	591
`comp.sys.ibm.pc.hardware`	590
`comp.sys.mac.hardware`	578
`rec.autos`	594
`rec.motorcycles`	598
`rec.sport.baseball`	597
`rec.sport.hockey`	600



Based on the graph and table above, we conclude that the training data set is evenly distributed.

Modeling Text Data and Feature Extraction

Task b

In this task, we are going to convert the text data into to feature vector using the tf-idf. Sklearn provides TfidfVectorizer to do such vectorize task. By using the **default vectorizer** we can convert to the text data into (4732, 79218), which means that there are 4732 documents in the data, and extract 798218 unique token in these documents. In practice, we should personalize the tokener for better performance. The first thing we could do it remove stop words, and after **remove stop word**, we can get the matrix as (4732, 78910).

Furthermore, we could personalize the vectorizer by adding an tokener with stemmer.

```
def stem_tokens(tokens):
   stemmer = PorterStemmer()
   stemmed = []
   for item in tokens:
       stemmed.append(stemmer.stem(item))
   return stemmed
```

```
def tokenize(text):
    '''
    override the tokenizer in CountVectorizer
    with stemming and reomoving punctuation
    '''
    tokenizer = RegexpTokenizer(r'[A-Za-z_]+') #remove punctuation
    tokens = tokenizer.tokenize(text)
    stems = stem_tokens(tokens)
    return stems
```

One things should be noticed here is, along with remove punctuation, we also remove the numeric character by using the RegexTokenizer. By adding the **stemmer and remove punctuation**. We get the matrix shape as (4732, 42602).

We also can set for minimum document frequency of vocabulary terms. By using min_df=2 we get matrix shape as (4732, 19812), and with min_df=5 we get shape as (4732, 8998)

Task c

In this task, we want to count the TFICF. We first combine the document in the whole class as a whole document, and use the TFIDF model to analyze them.
By using the default tokenizer we can get 10 most significant terms in each of the following classes

comp.sys.ibm.pc.hardware
[u'card', u'controller', u'organization', u'subject', u'lines', u'com', u'drive', u'ide', u'edu', u'scsi']
comp.sys.mac.hardware
[u'centris', u'com', u'scsi', u'quadra', u'apple', u'organization', u'mac', u'subject', u'lines', u'edu']
misc.forsale
[u'posting', u'university', u'com', u'new', u'organization', u'sale', u'lines', u'subject', u'00', u'edu']
soc.religion.christian
[u'bible', u'christ', u'lines', u'church', u'people', u'subject', u'jesus', u'christians',u'edu', u'god']

By using the personized tokenizer with stemming we can get:

comp.sys.ibm.pc.hardware
[u't', u'line', u'use', u'com', u'ide', u's', u'thi', u'edu', u'drive', u'scsi']
comp.sys.mac.hardware
[u'use', u'quadra', u't', u'organ', u'subject', u'line', u'mac', u'thi', u's', u'edu']
misc.forsale
[u'thi', u'new', u'post', u'com', u's', u'organ', u'subject', u'sale', u'line', u'edu']
soc.religion.christian
[u'church', u'hi', u'jesu', u'edu', u't', u'christian', u'wa', u'god', u's', u'thi']

Feature Selection

Task d

We can use both LSI and NMF to do the dimensionality reduction

```
def apply_lsi(train):

   svd = TruncatedSVD(n_components=50, random_state=42)
   SVD_train = svd.fit_transform(convert_to_tfidf(train))
   print SVD_train.shape
   return SVD_train

def apply_NMF(train):
   model = NMF(n_components=50, init='random', random_state=42)
   W_train = model.fit_transform(convert_to_tfidf(train))
   return W_train
```

Learning Algorithms

Task e

In this task we can use the SVM to train the binary classifier.
In order to get binary classifier, we first provide binary class data.

```
y = np.zeros(len(newsgroups_train.data))
   for i in range (len(newsgroups_train.data)):
       if newsgroups_train.target[i]<=3:
           y[i]=0
       else:
           y[i]=1
```

Then use high value γ and low value γ to set the SVM classifier with both LSI and NFM dimensionality reduction data.

Table 1. SVM classifier applying LSI model

	γ = 1000		γ=0.001	
	min_df =2	min_df=5	min_df=2	min_df=5
accuracy	0.975	0.974	0.505	0.505
recall	0.981	0.981	1.000	1.000
precision	0.968	0.968	0.505	0.505
confusion matrix	[[1511 49] [31 1559]]	[[1509 51] [30 1560]]	[[0 1560] [0 1590]]	[[0 1560] [0 1590]]
ROC AUC	0.9968	0.9967	0.9943	0.9941
ROC curve	Fig.1	Fig.2	Fig.3	Fig.4

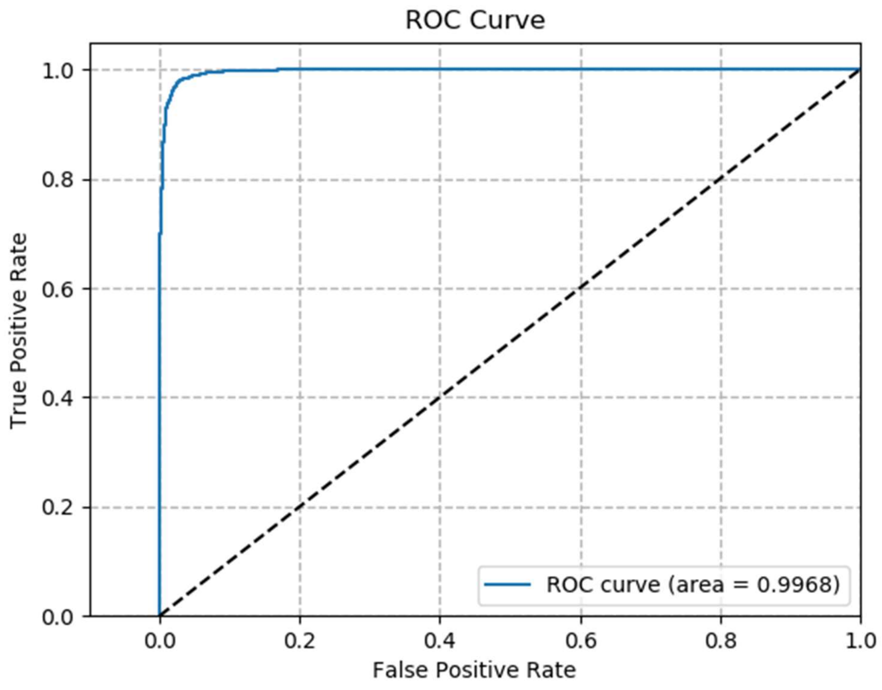


Fig.1 SVM classifier
ROC curve with LSI model
γ = 1000 min_df = 2

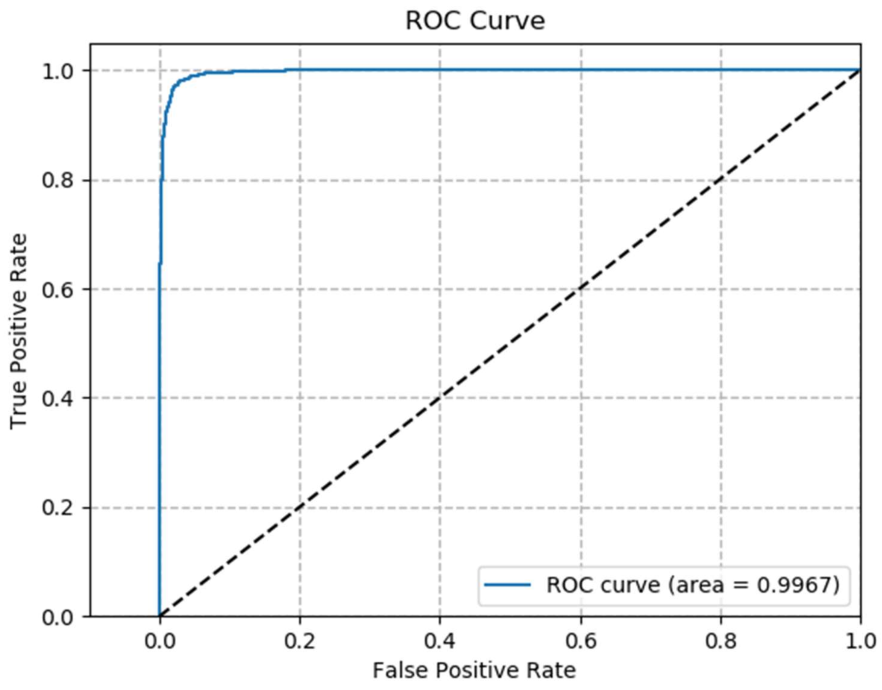


Fig.2 SVM classifier
ROC curve with LSI model
γ = 1000 min_df = 5

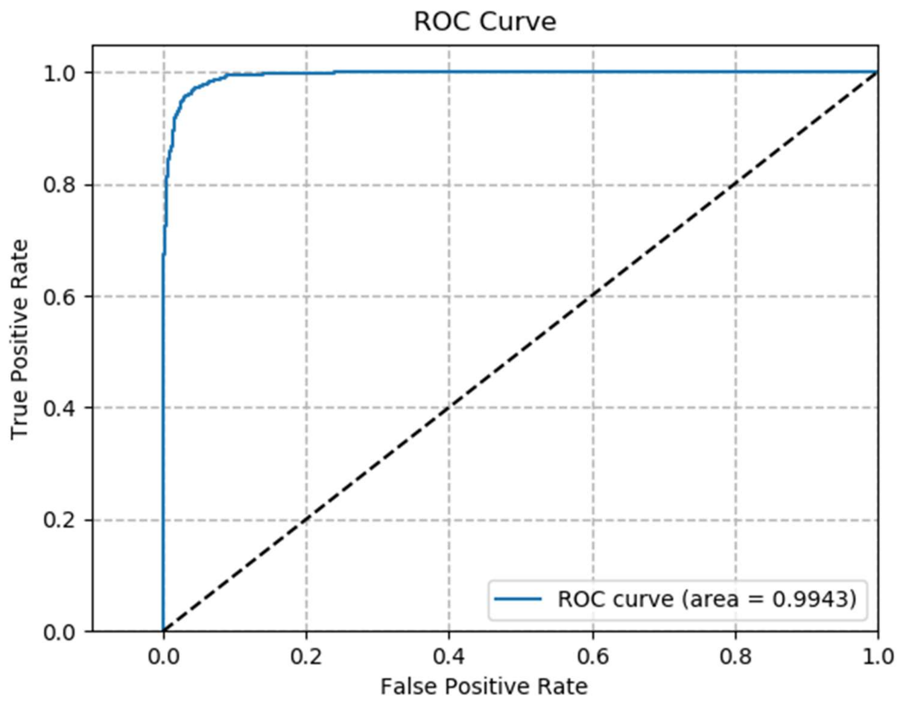


Fig. 3 SVM classifier
ROC curve with LSI model
γ = 0.001 min_df = 2

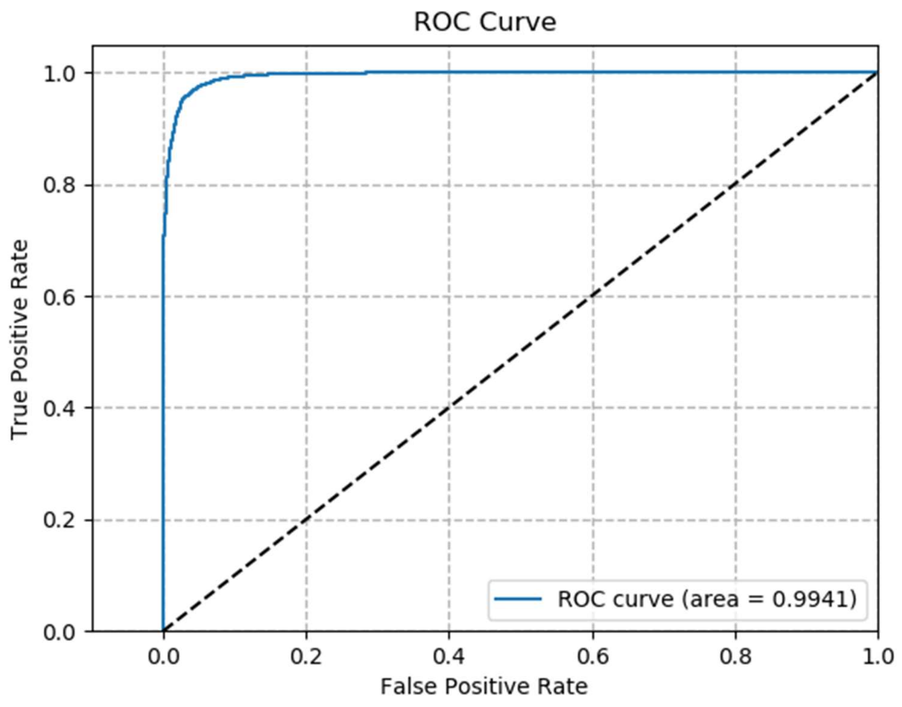


Fig. 4 SVM classifier
ROC curve with LSI model
γ = 0.001 min_df =5

Table 2. SVM classifier applying NMF model

	γ = 1000		γ=0.001	
	min_df =2	min_df=5	min_df=2	min_df=5
accuracy	0.961	0.967	0.505	0.505
recall	0.966	0.975	1.000	1.000
precision	0.957	0.960	0.505	0.505
confusion matrix	[[1491 69] [54 1536]]	[[1495 65] [40 1550]]	[[0 1560] [0 1590]]	[[0 1560] [0 1590]]
ROC AUC	0.9946	0.9952	0.9855	0.9684
ROC curve	Fig.5	Fig.6	Fig.7	Fig.8

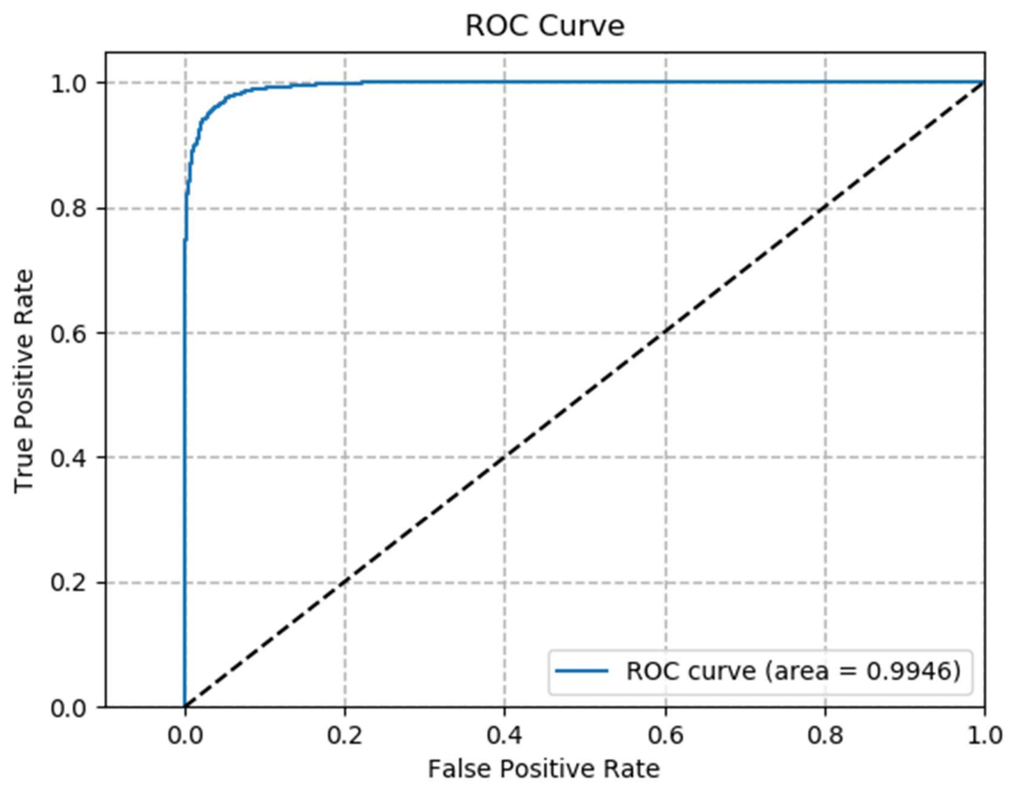


Fig. 5 SVM classifier
ROC curve with NMF model
γ = 1000 min_df = 2

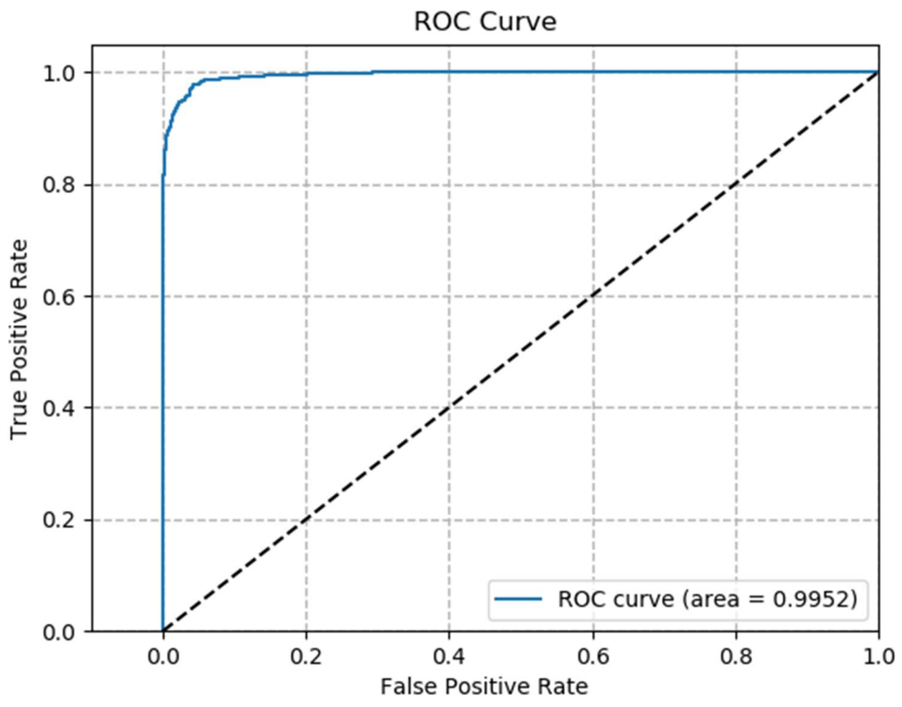


Fig.6 SVM classifier
ROC curve with NMF model
γ = 1000 min_df = 5

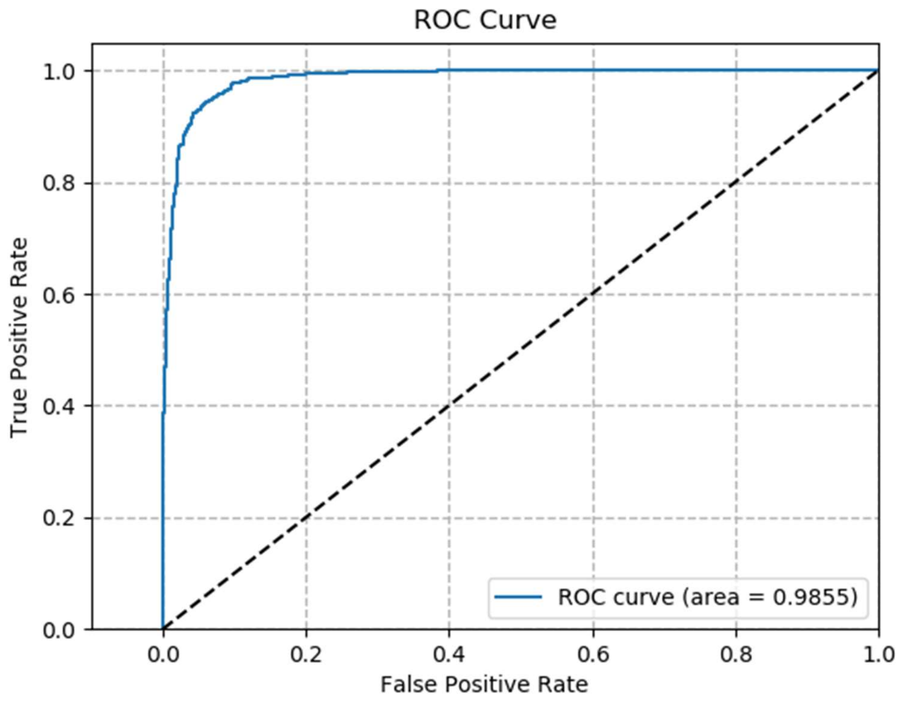


Fig.7 SVM classifier
ROC curve with NMF model
γ = 1000 min_df = 2

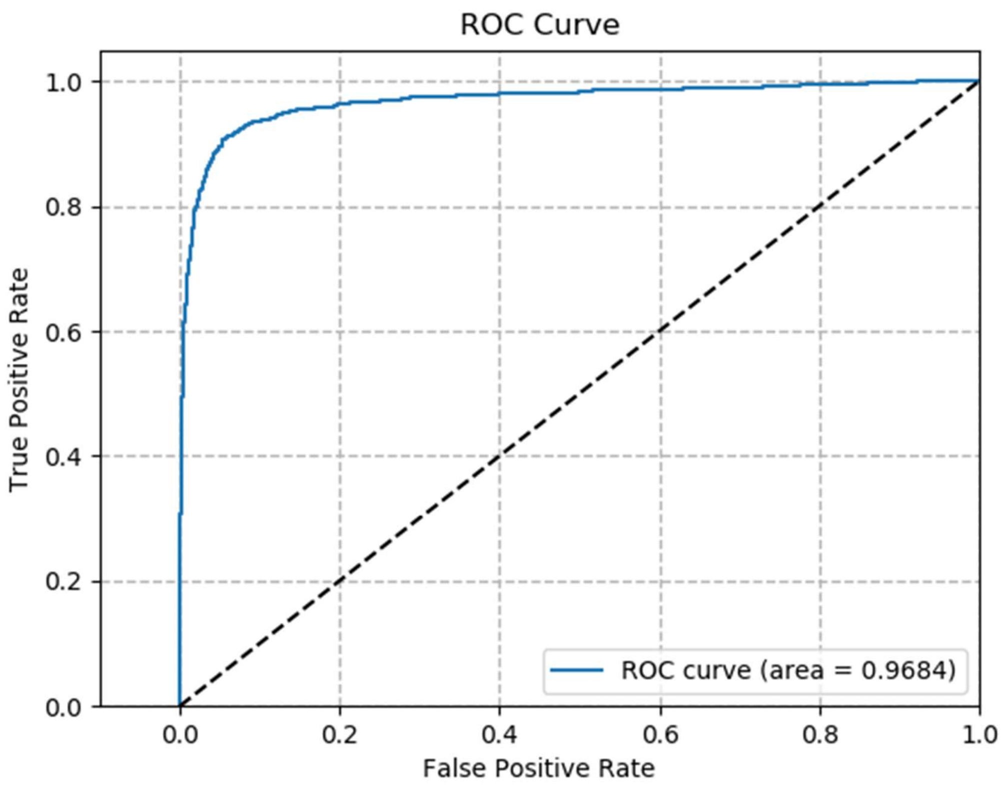


Fig.8 SVM classifier
ROC curve with NMF model
γ = 0.001 min_df = 5

Here we can see that although the soft margin will have a better recall score, it will also make the accuracy and precision score lower. We can also see that the LSI model have a better performance than the NMF model.

Task f
In this task, we want find the best γ. We apply NMF model and set min_df=2 in this task, and then change the γ into different value. By analyzing the accuracy score of 5-fold cross-validation, we can get the Table 3.

In this table, we can find that the accuracy and precision score will increase with the γ value. γ = 1000, we will get the highest accuracy score as 0.967458 and 0.969029. Although the recall score decrease a little, we still find γ = 1000 is the best parameter.

Table 3 Accuracy, Recall and Precision value of 5-fold cross-validation

γ	accuracy score	recall score	precision
0.001	0.504861	1.000000	0.504861
0.01	0.504861	1.000000	0.504861
0.1	0.508030	1.000000	0.506466
1	0.944210	0.964423	0.927988
10	0.956256	0.969865	0.944978
100	0.964920	0.966097	0.964518
1000	0.967458	0.966515	0.969029

Task g
In this task, we will apply multinomial naïve Bayes classifier to this binary classify task. In the table 4 we provide the accuracy, recall precision and confusion matrix of the classifiers.

Table 4 multinomial naïve Bayes classifier

	min_df =2	min_df=5
accuracy	0.941	0.939
recall	0.988	0.972
precision	0.904	0.913
confusion matrix	[[1393 167] [19 1571]]	[[1412 148] [44 1546]]
ROC AUC	0.9901	0.9880
ROC curve	Fig.9	Fig.10

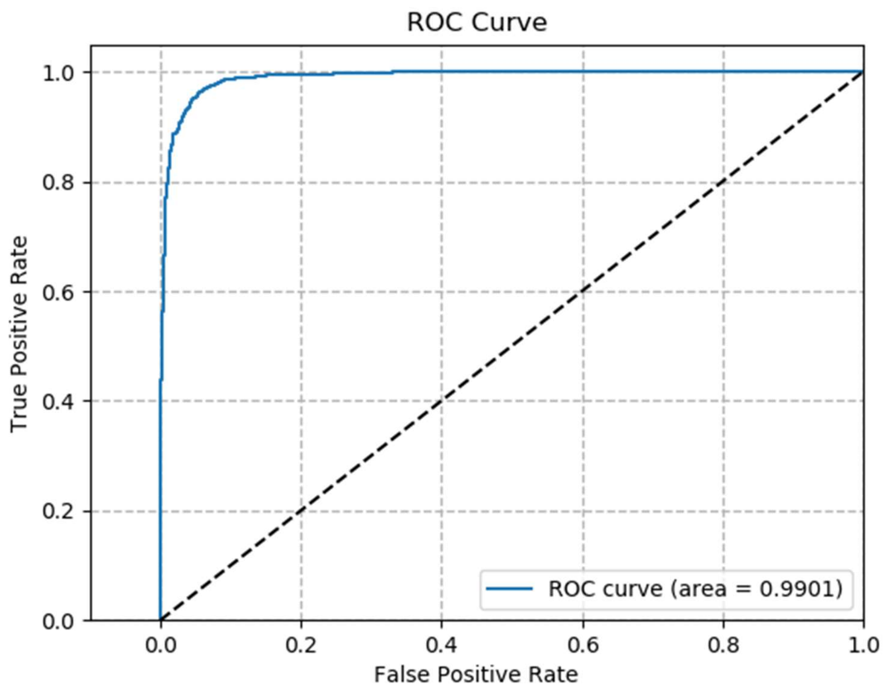


Fig.9 multinomial naïve Bayes classifier
ROC curve with NMF model
min_df = 2

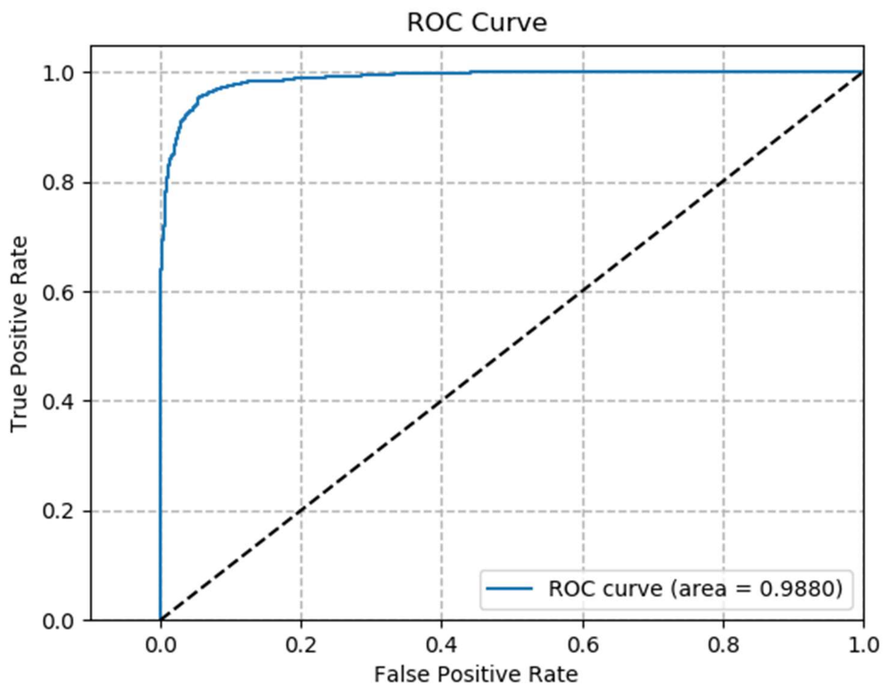


Fig. 10 multinomial naïve Bayes classifier
ROC curve with NMF model
min_df = 5

Task h
In this task, we will apply logistic regression classifier to this binary classify task.
In the table 5 we provide the accuracy, recall precision and confusion matrix of the classifiers.

Table 5 logistic regression classifier

	min_df =2	min_df=5
accuracy	0.941	0.943
recall	0.960	0.960
precision	0.926	0.930
confusion matrix	[[1438 122] [63 1527]]	[[1445 115] [63 1527]]
ROC AUC	0.9870	0.9839
ROC curve	Fig.11	Fig.12

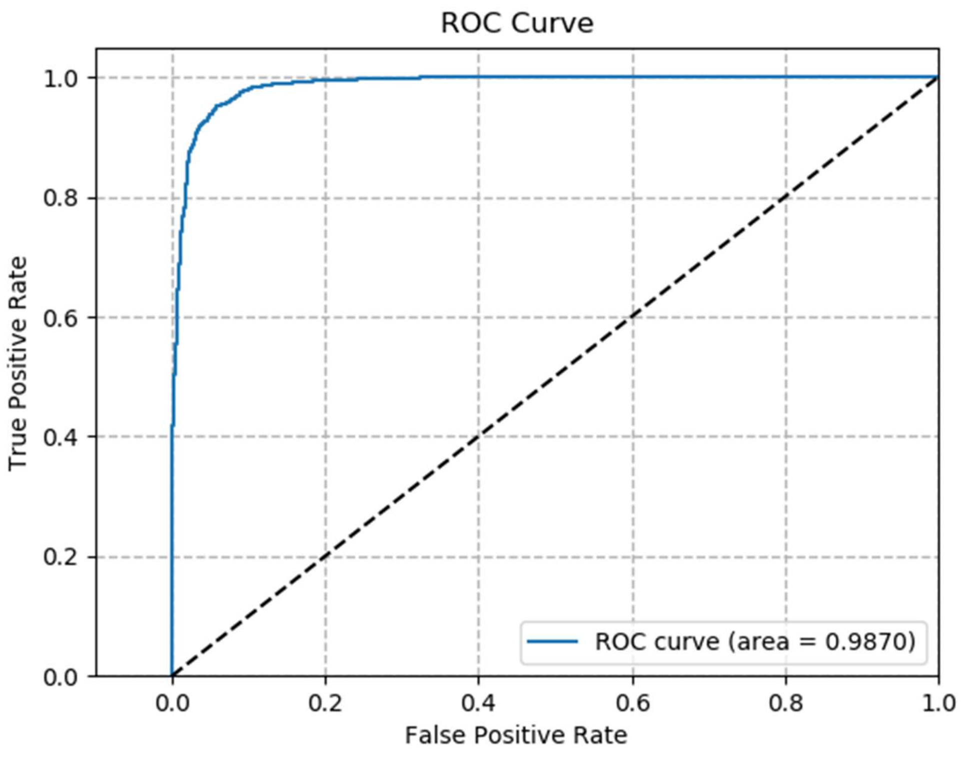


Fig. 11 Logistic Regression Classifier
ROC curve with NMF
min_df = 2

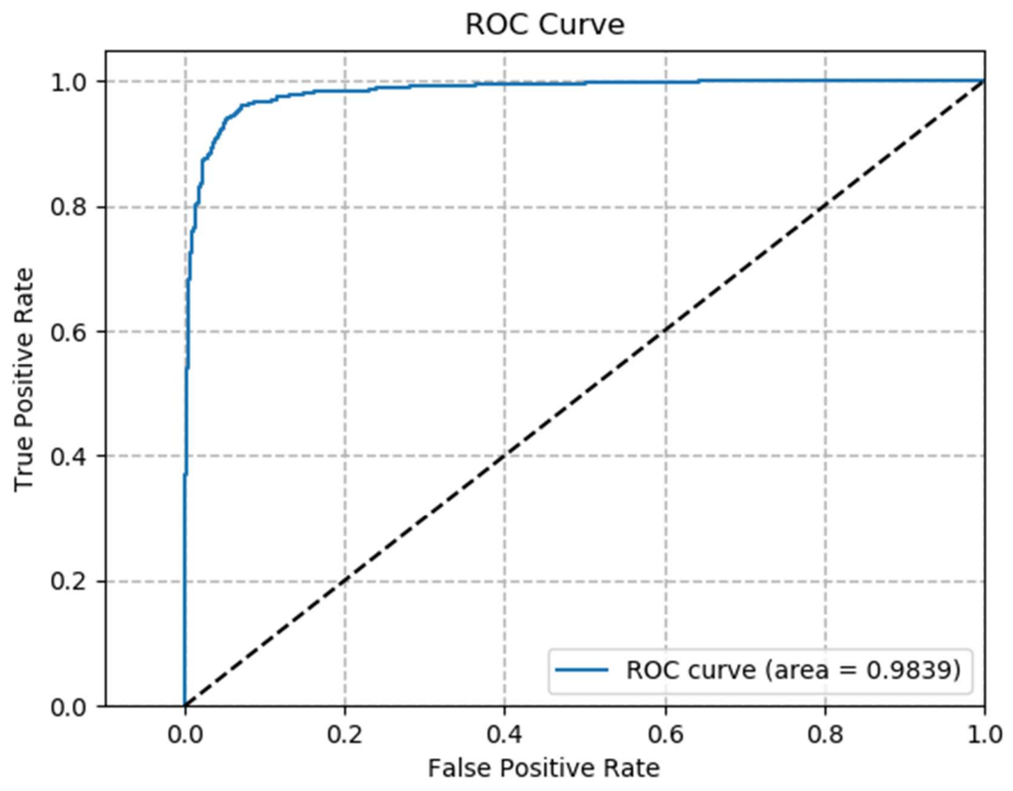


Fig. 12 Logistic Regression Classifier
ROC curve with NMF
min_df = 2

Task i

Table 6 Relationship between regulariztion and test accuracy

C	l1	l2
0.001	0.495	0.506
0.01	0.495	0.607
0.1	0.706	0.906
1	0.960	0.942
10	0.968	0.958
100	0.967	0.967
1000	0.966	0.969

In this task we will analyze the relationship between regularization and the test error. The coefficient here is the inverse of regularization strength and smaller values specify stronger regularization.

As one can see according to the result from table above, if the regularization term is too strong, the classifiers tend to perform very poorly since they are all overly generalized, as almost randomly guessing. In terms of cost function of our classifiers, the regularization coefficient controls the tradeoff between weights of model and regularization term. If the regularization term dominates the cost function, the weights parameter tend to have very small value which vanish any useful input features, resulting very bad performance prediction.

Regularization prevents that our models overfits the training data set by trying to elimiate high-order feature terms. This could be done by adding regulraiztion term in the cost function. L1-norm term tends to produce sparse weight matrix which keeps only few terms in the final model, called built-in feature section in some sense. On the other hand, L2-norm term has better computation efficiency since its derivative of square of coefficient will produce unique analytic solution.

Multiclass Classification

Task j

For multiclass classification, some classifiers such as Naive Bayes classifier inherently handle multiclass situation properly, since it computes the prior for each class and the likelihood for each target value. On the contrary, SVM tries to linearly seperate data points from two different classes. In order to classify multiclass data points, we need to train multiple classifiers either with one-vs-one scheme or one-vs-all scheme. The SVC implementation handles the multiclass support according to a one-vs-one scheme. The one-vs-all scheme of SVC implementation can be accomplished by using OneVsRestClassifier wrapper provided by sk-learn framework. The following tables shows the result for multiclass classification of these different classifiers with both NMF and LSI dimension reduction methods. Since Naive Bayes does not accept negative value features, only NMF is applied for Naive Bayes classifier.

Table 7 Multiclass classification accuracy with NMF dimension reduction

Metrics\Classifiers	Naive Bayes	one-vs-one SVM	one-vs-all SVM
Accuracy	0.813	0.753	0.820
Recall	0.811	0.751	0.819
Precision	0.820	0.820	0.816
Confusion Matrix	Fig. 13	Fig. 14	Fig. 15

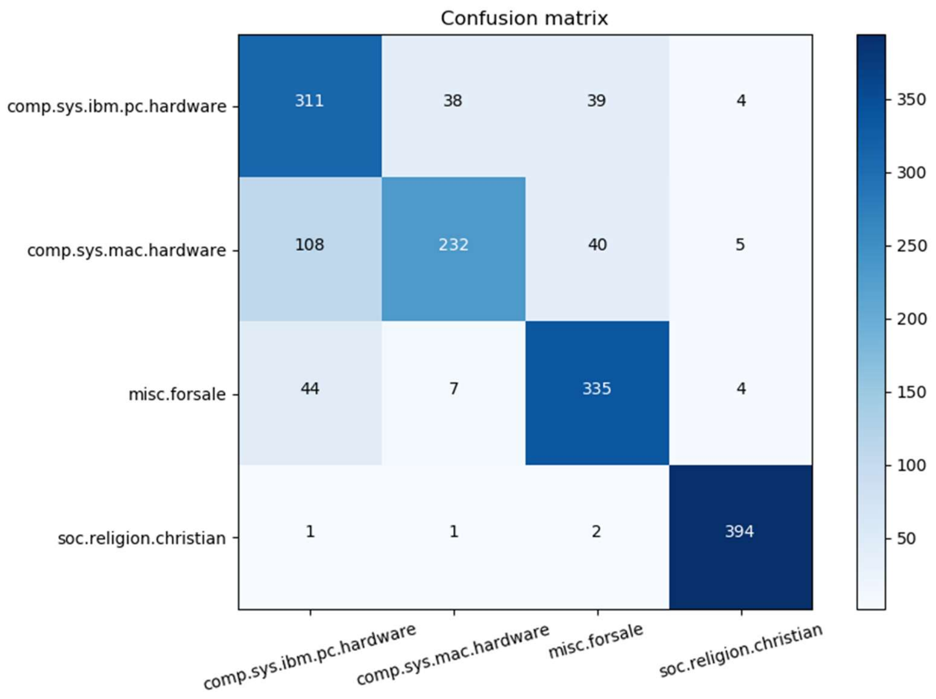


Fig. 13

NMF dimension reduction

Confusion Matrix of Naive Bayes classifier

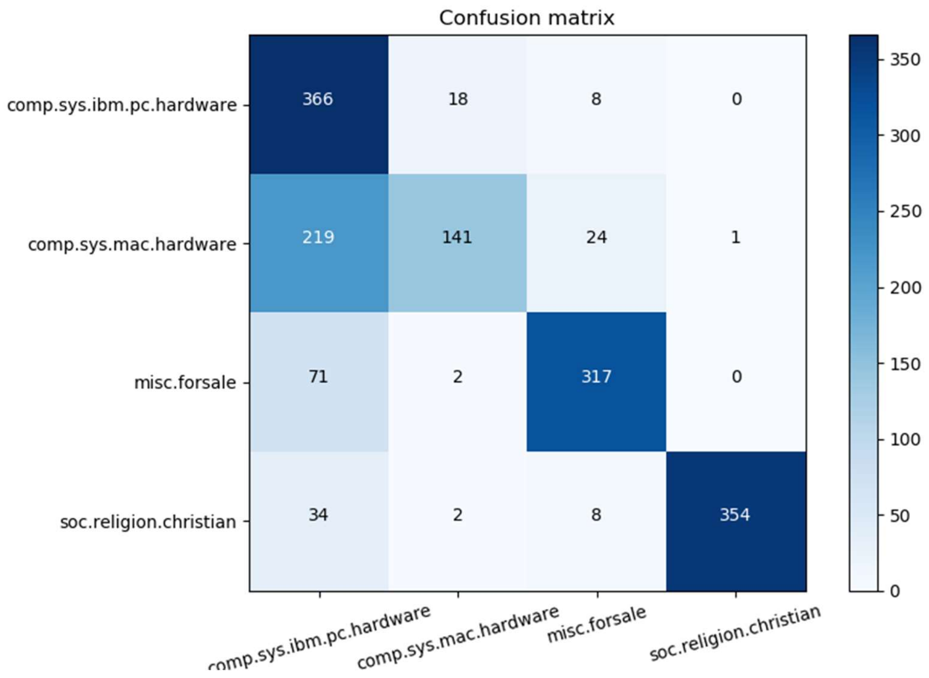


Fig. 14

NMF dimension reduction

Confusion Matrix of SVM with one-vs-one scheme

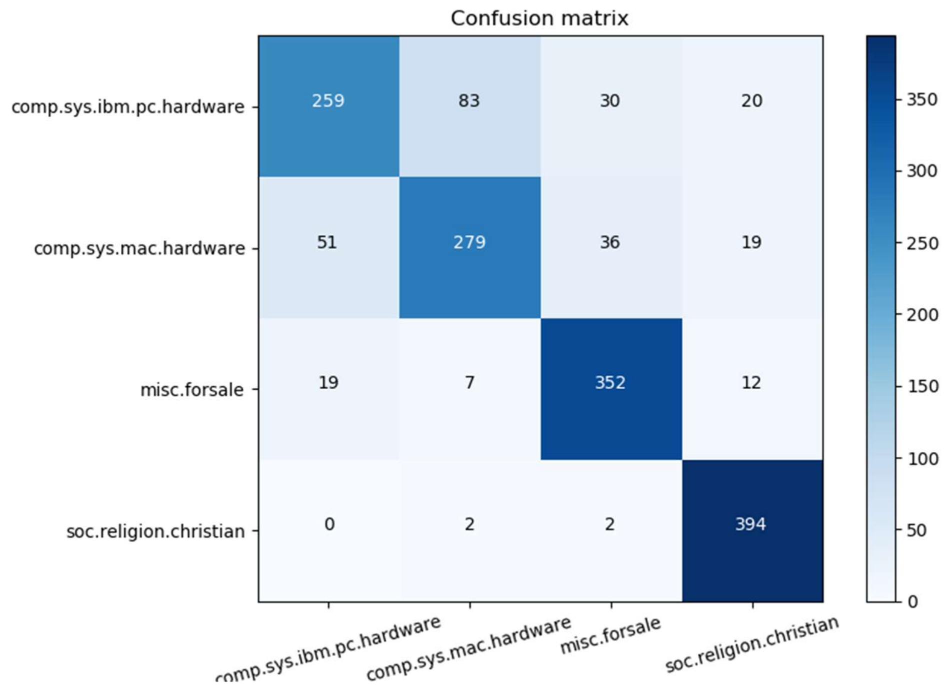


Fig. 15
NMF dimension reduction
Confusion Matrix of SVM with one-vs-all scheme

Table 8 Multiclass classification accuracy with LSI dimension reduction

Metrics\Classifiers	one-vs-one SVM	one-vs-all SVM
Accuracy	0.867	0.873
Recall	0.866	0.873
Precision	0.871	0.872
Confusion Matrix	Fig. 16	Fig. 17

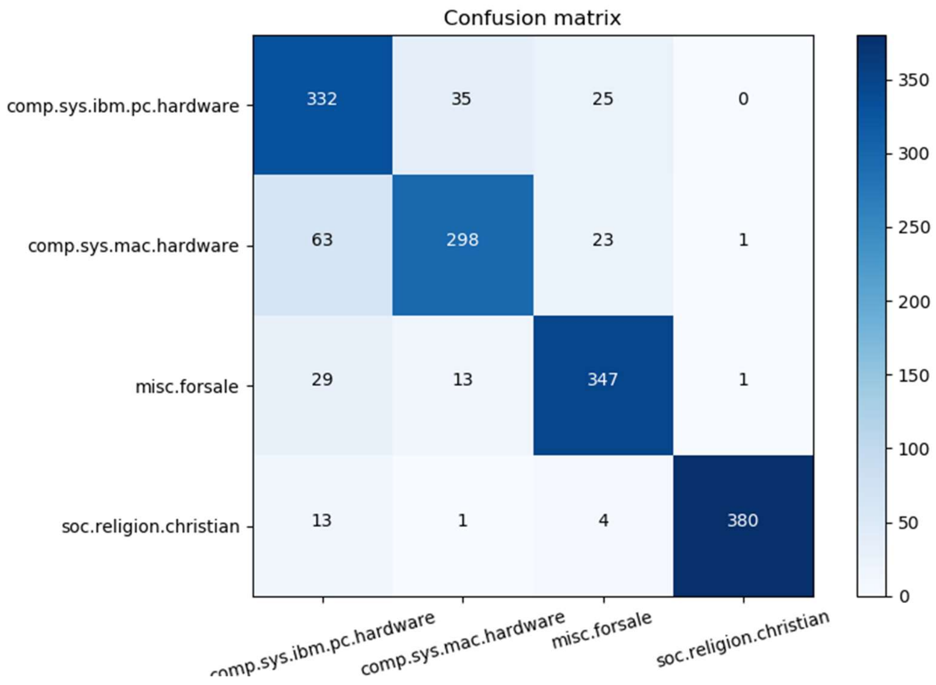


Fig. 16

LSI dimension reduction

Confusion Matrix of SVM with one-vs-one scheme

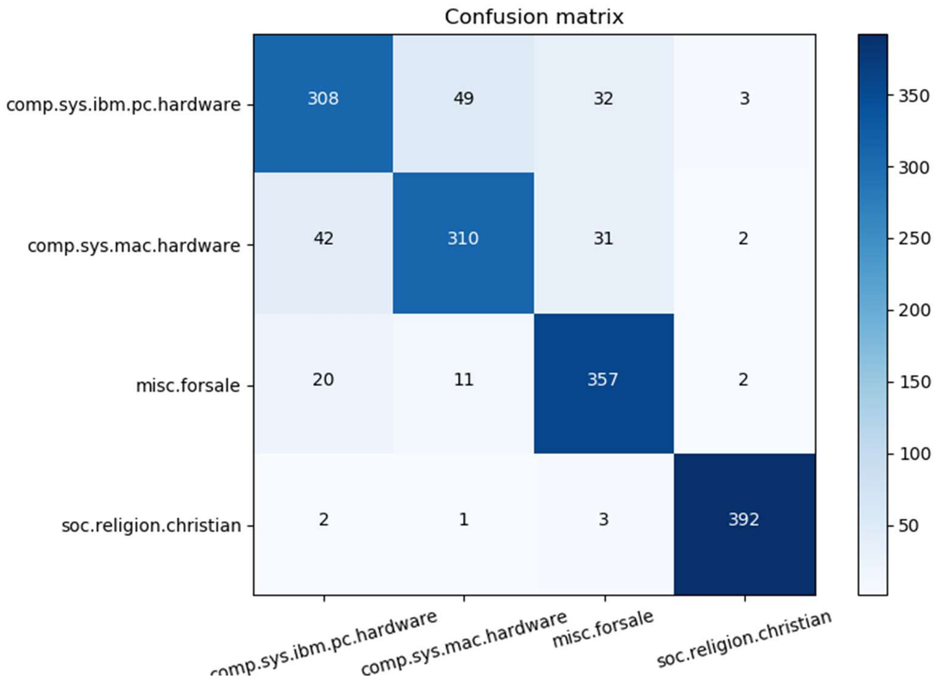


Fig. 17

LSI dimension reduction

Confusion Matrix of SVM with one-vs-all scheme