EE219 Project 1

Classification Analysis on Textual Data

Winter 2018

Weijie Tang	305029285	twjeric@cs.ucla.edu
Wandi Cui	905024671	wandicui@ucla.edu

1. Introduction

In this project, we try to classify textual data with different classification models. Our work mainly includes following steps:

1) Data Exploration (observing the imbalanced data);
2) Feature Extraction and Selection (get TFxIDF, implement dimension reduction method such as LSI and NMF);
3) Apply binary classification model to solving the problem, such as Hard/Soft Margin SVM, Multinomial Naïve Bayes and Logistic Regression; and then extend to multiclass classification with SVM and Naïve Bayes algorithms;
4) Cross Validation (5 fold);
5) Model Evaluation (utilize confusion matrix, accuracy, precision, recall and ROC curve);
6) Data Visualization

2. Data Exploration (Problem a)

We used the 20 newsgroups dataset, which comprises around 18000 newsgroups posts on 20 topics split in two subsets: one for training and the other one for testing.[1] In a real-world classification problem, the data we have is generally imbalanced, which may impact the effective of the learning algorithms. So, to avoid this in our dataset, we plotted a histogram of the number of training documents per class to check if they are evenly distributed. We only cared about the following 8 classes:

- 'comp.graphics'
- 'comp.os.ms-windows.misc'
- 'comp.sys.ibm.pc.hardware'
- 'Comp.sys.mac.hardware'
- 'rec.autos'
- 'rec.motorcycles'
- 'rec.sport.baseball'
- 'rec.sport.hockey'

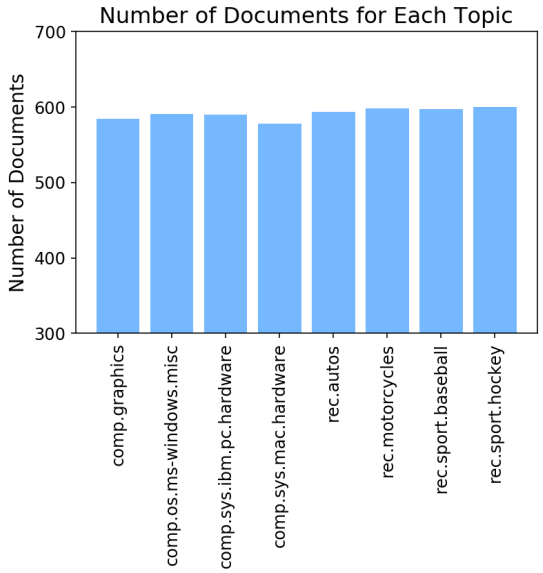


Figure 2. Histogram of the number of documents for each topic

Figure 2 is the histogram of the number of documents for these topics. Through the diagram, we can see that the data is already evenly distributed, so we can directly use it in next steps.

3. Feature Extraction and Selection

3.1 TFxIDF Vector Representations

3.1.1 Extract Terms

First, we tokenized each document; then removed punctuations and stop words. Next, extracted stem of each words to get the final terms. We created a function "tokenize_and_stem" to realize this process. The codes are shown below:

```
def tokenize_and_stem(text):
    tokens = nltk.tokenize.word_tokenize(text)
    tokens = [token.strip(string.punctuation) for token in tokens if token.isalnum()]
    tokens = [stemmer.stem(token) for token in tokens]
    return tokens
```

3.1.2 Create TFxIDF Vector Representations (Problem b)

To create TFxIDF Vector, we used the class TfidfVectorizer in scikit-learn. By setting parameter 'tokenizer = tokenize_and_stem', we could successfully feed the processed data into other models and functions; and then we used the function 'fit_transform' to turn raw document data into document~term matrix: TFxIDF.

By setting parameter 'min_df = 2' or 'min_df = 5', we got different numbers of terms, the results are shown below:

- When min_df = 2, we got 4732 documents with 12828 terms.
- When min_df = 5, we got 4732 documents with 5797 terms.

3.1.3 Find the 10 Most Significant Terms (Problem c)

To discover the 10 most significant terms in specified classes, TFxICF matrix is needed. We applied class 'CountVectorizer' and 'TfidfTransformer' together with the function 'fit_transform' this time to transfer document data into class~term matrix: TFxICF. What is worth to mention is that integrating document data into certain classes should be done before counting and transforming.

By setting parameter 'min_df = 2', and 'min_df = 5', we got different number of terms, the results are shown in two tables below:

Table 3.1. 10 Most Significant Terms When min_df = 2

Class	Top 10 Terms
comp.sys.ibm.pc.hardware	drive, thi, scsi, use, ide, card, disk, control, ani, bio
comp.sys.mac.hardware	thi, mac, use, appl, drive, simm, problem, ha, scsi, quadra
misc.forsale	1, 2, thi, new, sale, offer, use, includ, ship, price
soc.religion.christian	thi, god, wa, christian, jesu, hi, church, christ, peopl, say

Table 3.2. 10 Most Significant Terms When min_df = 5

Class	Top 10 Terms
comp.sys.ibm.pc.hardware	drive, thi, scsi, use, ide, card, disk, control, ani, bio
comp.sys.mac.hardware	thi, mac, use, appl, drive, simm, problem, ha, scsi, doe
misc.forsale	1, 2, thi, new, sale, offer, use, includ, ship, price
soc.religion.christian	thi, god, wa, christian, jesu, hi, church, christ, peopl, say

3.2 Feature Selection (Problem d)

We have already generated features so far, however, the representation vectors (TFxIDF vectors) is high-dimensional and sparse. They may cause algorithms to perform poorly, which is known as "The Curse of Dimensionality" problem.

Therefore, we applied two dimensionality reduction transform methods, one was Latent Semantic Indexing (LSI), and the other one was Non-Negative Matrix Factorization (NMF). We utilized 'TruncatedSVD' and 'NMF' in package 'sklearn.decomposition' to implement LSI and NMF respectively, and reduced the dimension to 50. The codes are shown below:

```
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=50, n_iter=7, random_state=42)
lsiVectors = svd.fit_transform(tfidfVectors)
from sklearn.decomposition import NMF

nmf = NMF(n_components=50, init='random', random_state=0)
nmfVectors = nmf.fit_transform(tfidfVectors)
```

4. Model Building and Classification

4.1 Preprocessing Pipeline

To avoid duplication of effort, we prepared a preprocessing pipeline that can complete term extraction, dimensionality reduction, and normalization sequentially. The codes are shown below:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import Normalizer
from sklearn import metrics

# preprocessing pipeline
pp = Pipeline([('vect', tfidfVectorizer), ('dim_red', svd), ('norm', Normalizer(copy=False))])

x_train = pp.fit_transform(newsgroups_train.data)
x_test = pp.transform(newsgroups_test.data)

vfunc = np.vectorize(lambda t : t / 4)
y_train = vfunc(newsgroups_train.target)
y_test = vfunc(newsgroups_test.target)
```

The numpy vectorized function ‘vfunc’ is used to combine 8 subclasses into two target classes: ‘Computer Technology’ and ‘Recreational Activity’.

4.2 ROC Curves Plotting Function

Since we need to plot ROC curves for many times, we applied the tool ‘Plotly’[2] and defined a plotting function ‘draw_roc_curve’.

```
def draw_roc_curve(fpr, tpr, roc_auc, name):
    data = []
    width, height = 800, 600

    if type(name) != list or len(name) == 1:
        width, height = 600, 450
        trace1 = go.Scatter(x=fpr, y=tpr,
                            mode='lines',
                            line=dict(color='darkorange', width=2),
                            name='%s (area = %0.2f)' % (name, roc_auc))
        data.append(trace1)
    else:
        for i in range(len(name)):
            trace1 = go.Scatter(x=fpr[i], y=tpr[i],
                                mode='lines',
                                line=dict(width=2),
                                name='%s (area = %0.2f)' % (name[i], roc_auc[i]))
            data.append(trace1)

    trace2 = go.Scatter(x=[0, 1], y=[0, 1],
                        mode='lines',
                        line=dict(color='navy', width=2, dash='dash'),
                        showlegend=False)
    data.append(trace2)

    layout = go.Layout(title='Receiver Operating Characteristic',
                       autosize=False,
                       width=width,
                       height=height,
                       xaxis=dict(title='False Positive Rate', ticks='outside', mirror=True, linewidth=1),
                       yaxis=dict(title='True Positive Rate', ticks='outside', mirror=True, linewidth=1, range=[0, 1.05]),
                       legend=dict(x=.5, y=.05, bordercolor='#D3D3D3', borderwidth=1))

    fig = go.Figure(data=data, layout=layout)
    py.iplot(fig)
```

This function could either plot the ROC curve for one kind of classifier or plot all ROC curves together for different kinds of classifiers.

4.3 Binary Classification

4.3.1 SVM Classifier (Problem e)

SVM model performs really well in binary classification. It constructs margins in the high-dimensional space to separate different categories by learning from training data. Furthermore, linear SVM is proved to be effective and efficient when dealing with sparse high-dimensional dataset, including textual data. There are two ways to implement SVM, one is called Hard Margin SVM, and the other is Soft Margin SVM. The former strictly separates the data into two classes without any error tolerance, which means a single outlier can determine the boundary and causes overfitting, so we also tried the Soft Margin SVM which sets a slack variable that can allow some outliers exist in a wrong class.

To realize hard margin SVM, we applied SVC in ‘sklearn.svm’ package, and the codes are as follows (including the model evaluation part):

```
from sklearn.svm import SVC

# hard margin SVM classifier
clf_svm1 = SVC(C=1000.0, kernel='linear')
clf_svm1.fit(x_train, y_train)
score = clf_svm1.decision_function(x_test)
predicted = clf_svm1.predict(x_test)

print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(y_test, predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(y_test, predicted, average='macro')))
print("%-12s %f" % ('Recall:', metrics.recall_score(y_test, predicted, average='macro')))
print("Confusion Matrix: \n{0}".format(metrics.confusion_matrix(y_test, predicted)))

fpr_svm1, tpr_svm1, thresholds = metrics.roc_curve(y_test, score)
roc_auc_svm1 = metrics.auc(fpr_svm1, tpr_svm1)
draw_roc_curve(fpr_svm1, tpr_svm1, roc_auc_svm1, 'ROC curve')
```

Specifically, 'decision_function' method returns the distance between each sample and boundary; and 'predict' method could get the predicted label for each test data. The parameter 'C = 1000.0', a pretty large number, realized 'Hard Margin SVM'.

Table 4.1. Hard Margin SVM Results

Parameters	Accuracy	Precision	Recall	Confusion Matrix
min_df = 2, LSI	0.933333	0.934091	0.933146	[[1425 135] [75 1515]]
min_df = 5, LSI	0.930794	0.931456	0.930618	[[1423 137] [81 1509]]
min_df = 2, NMF	0.926667	0.927342	0.926488	[[1416 144] [87 1503]]

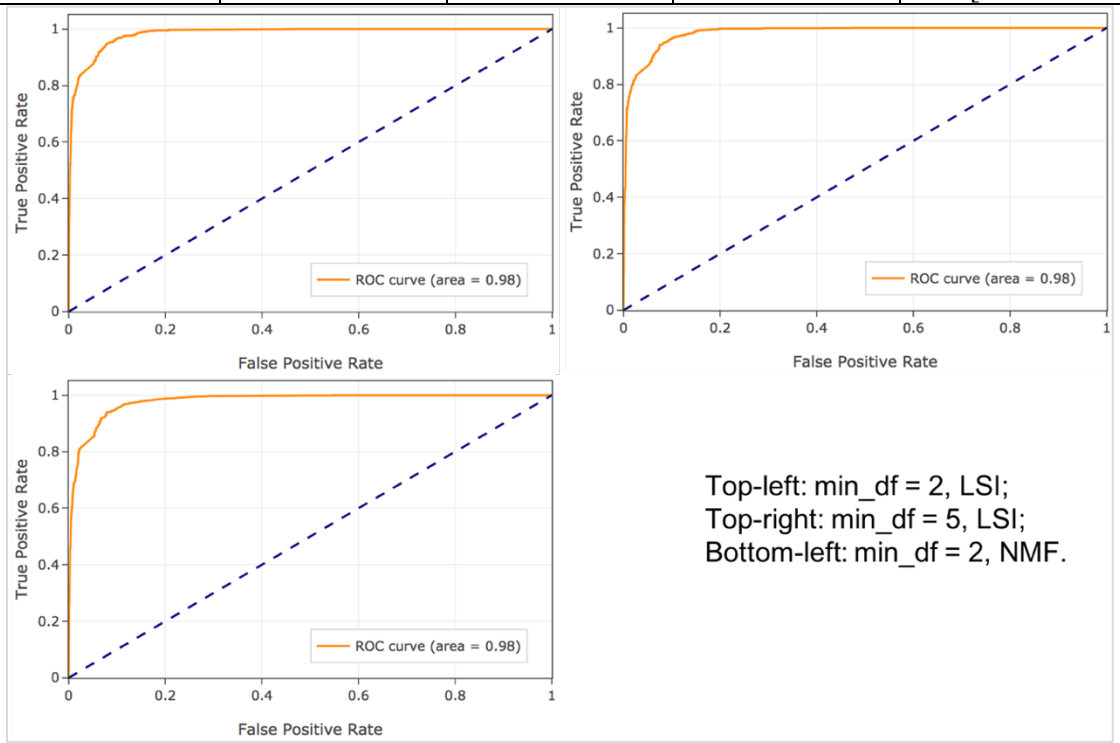


Figure 4.1. Hard Margin SVM ROC

To realize Soft Margin SVM, set ‘C = 0.001’.

Table 4.2. Soft Margin SVM Results

Parameters	Accuracy	Precision	Recall	Confusion Matrix
min_df = 2, LSI	0.504762	0.252381	0.500000	[[0 1560] [0 1590]]
min_df = 5, LSI	0.504762	0.252381	0.500000	[[0 1560] [0 1590]]
min_df = 2, NMF	0.504762	0.252381	0.500000	[[0 1560] [0 1590]]

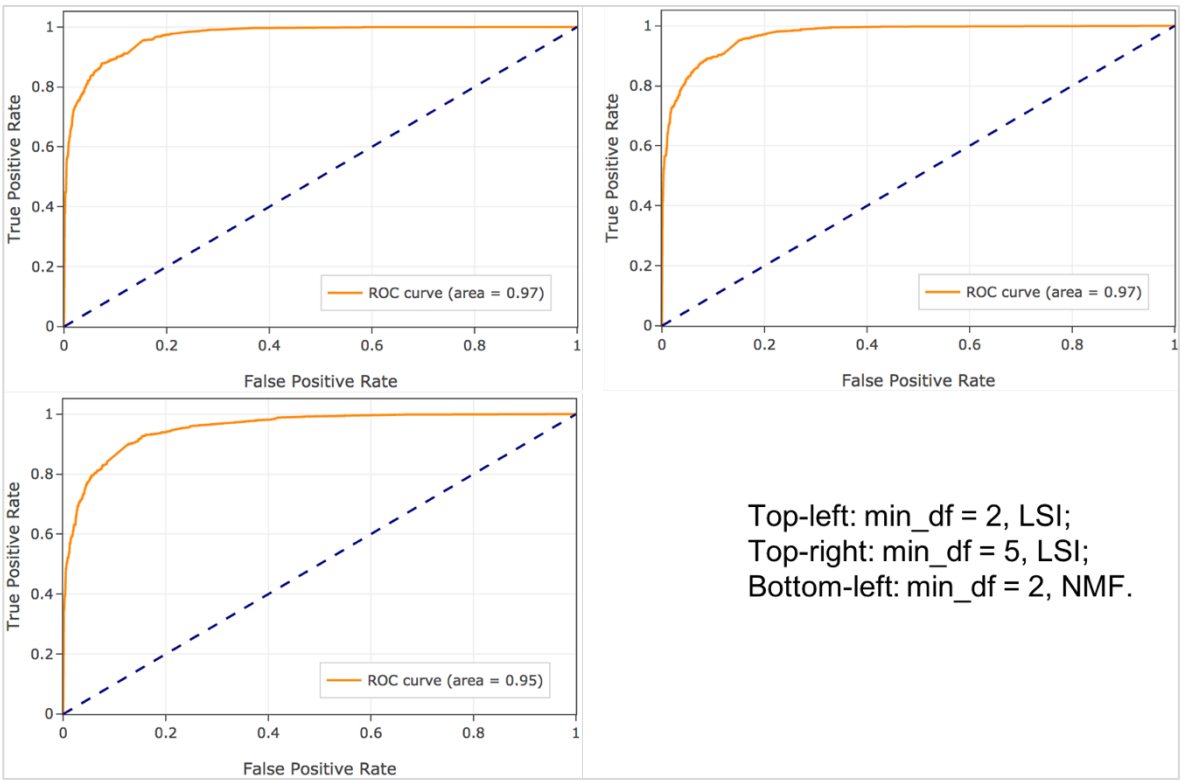


Figure 4.2. Soft Margin SVM ROC

In summary, for Hard Margin SVM, applying LSI can get better results. The results of min_df = 2 are slightly better than those of min_df = 5. For Soft Margin SVM, the confusion matrix in this part seems weird without true positive and false positive. The reason why this happened is that we chose ‘C = 0.001’, which gives the algorithm such a loose constraint that all the data points are separated into one class.

4.3.2 Grid Search with 5-fold Cross Validation (Problem f)

In this section, we applied grid search with 5-fold cross validation to get the best value of gamma in SVM classifier (parameter C in SVC) under different conditions of ‘min_df’ and dimensionality reduction methods. The codes are shown below:

```
from sklearn.model_selection import GridSearchCV
import math

parameters = {'C': [math.pow(10, k) for k in range(-3,4)]}

gs_clf = GridSearchCV(SVC(kernel='linear'), parameters, n_jobs=-1, cv=5)
gs_clf.fit(x_train, y_train)

print("Best Parameters: ")
print(gs_clf.best_params_)

score = gs_clf.decision_function(x_test)
predicted = gs_clf.predict(x_test)
```

To implement 5-fold cross validation, we imported class 'GridSearchCV' from 'sklearn.model_selection' package with parameter 'cv=5'. We tried 7 different values of parameter gamma (the minimum is 10^{-3} and the maximum is 10^{3}). The attribute 'best_params_' could show the best value of gamma for us.

Table 4.3. SVM Cross Validation Results

Parameters	Best 'C'	Accuracy	Precision	Recall	Confusion Matrix
min_df = 2, LSI	1.0	0.932063	0.932772	0.931882	[[1424 136] [78 1512]]
min_df = 5, LSI	1.0	0.929841	0.930568	0.929657	[[1420 140] [81 1509]]
min_df = 2, NMF	1000.0	0.926667	0.927342	0.926488	[[1416 144] [87 1503]]

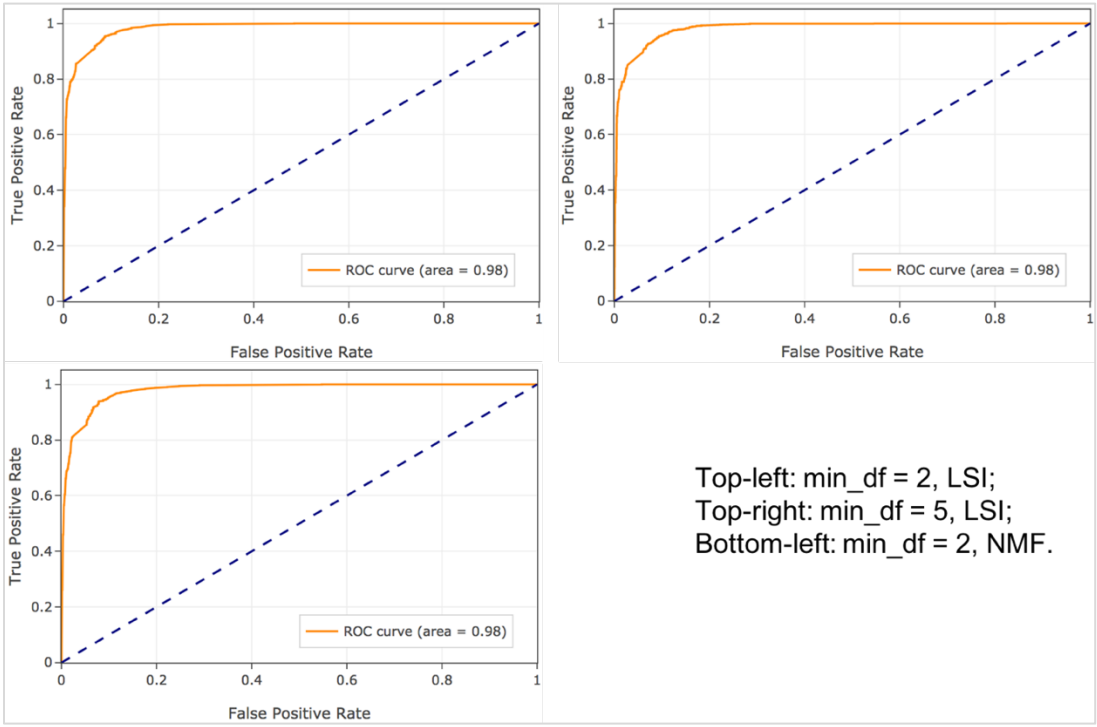


Figure 4.3. SVM Cross Validation ROC

According to the result, no matter the parameter min_df = 2 or 5, when applying LSI, the best value of gamma is 1.0 and when applying NMF, the best value of gamma is 1000.0. That is to say, when applying NMF, SVM classifier needs to have a much more restrictive condition than applying LSI.

4.3.3 Multinomial Naïve Bayes Classifier (Problem g)

We trained a Multinomial Naïve Bayes classifier for the same problem, as presented above, we also did comparison under different conditions of 'min_df' and dimensionality reduction methods. The codes are shown below.

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.preprocessing import MinMaxScaler

clf_nb = Pipeline([('scaler', MinMaxScaler()), ('clf', MultinomialNB())])
clf_nb.fit(x_train, y_train)
score = clf_nb.predict_proba(x_test)[:,1]
predicted = clf_nb.predict(x_test)
```

It is worth mentioning that after LSI transformation, some of the feature vectors may contain negative elements, so we applied a scaling method 'MinMaxScaler' before feeding data into Multinomial Naïve Bayes model.

Table 4.4. Multinomial Naïve Bayes Results

Parameters	Accuracy	Precision	Recall	Confusion Matrix
min_df = 2, LSI	0.895556	0.900580	0.895029	[[1310 250] [79 1511]]
min_df = 5, LSI	0.894286	0.899644	0.893741	[[1305 255] [78 1512]]
min_df = 2, NMF	0.911111	0.911579	0.910958	[[1396 164] [116 1474]]

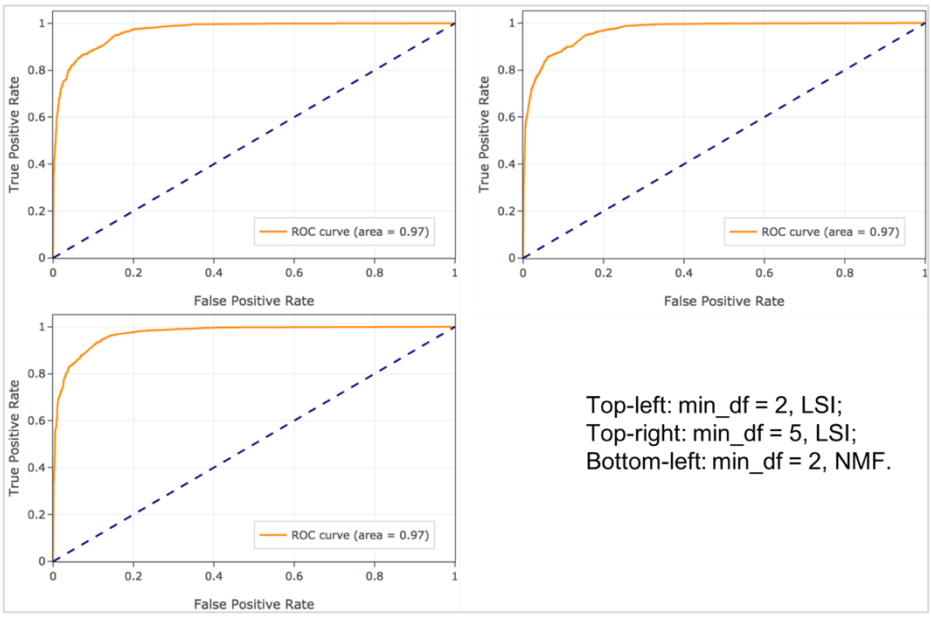


Figure 4.4. Multinomial Naïve Bayes ROC

For Multinomial Naïve Bayes classifier, applying NMF can get better results. Besides, smaller min_df helps the model perform a little better.

4.3.4 Logistic Regression Classifier (Problem h & i)

Logistic Regression Model maps the linear regression results into probabilities ranging from 0 to 1. It can deal with both discrete and continuous variables. It assumes a parametric form for the distribution P(Y|X), where X is features and Y is the classification result; then directly estimates its parameters from the training data.

We will discuss the results in two parts: whether or not adding a regularization term to the optimization objective, and each part still have the parameter comparison of different 'min_df' and dimensionality reduction methods.

To implement Logistic Regression model, we utilized class 'LogisticRegression' from package 'sklearn.linear_model'. By default, it uses l2 norm regularization and C = 1.0 for regularization strength.

Table 4.5. Logistic Regression Results

Parameters	Accuracy	Precision	Recall	Confusion Matrix
min_df = 2, LSI	0.928571	0.929208	0.928399	[[1420 140] [85 1505]]
min_df = 5, LSI	0.927937	0.929010	0.927709	[[1410 150] [77 1513]]
min_df = 2, NMF	0.919365	0.919466	0.919297	[[1423 137] [117 1473]]

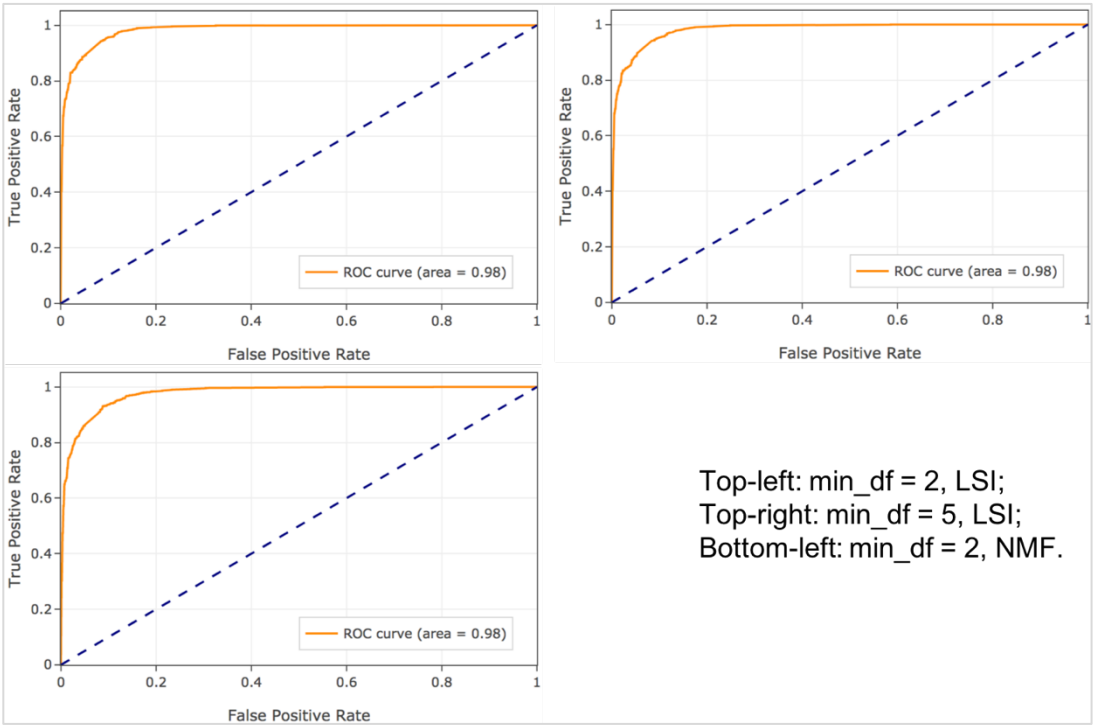


Figure 4.5. Logistic Regression ROC

For Logistic Regression with regularization, we used both *l1* and *l2* norm regularization; and in each norm regularization, we changed parameter 'C' from 0.001 to 1000 as well as compared the differences among parameter groups of different 'min_df' and dimensionality reduction methods. This 'C' parameter must be a positive float type number, and smaller C values represent stronger regularization.

Table 4.6. Logistic Regression Grid Search Scores (L1 norm)

Penalty	C	Accuracy	Rank
l1	0.001	0.495139	7
l1	0.01	0.872358	6
l1	0.1	0.911665	5
l1	1	0.935123	1
l1	10	0.934700	2
l1	100	0.933855	3
l1	1000	0.933432	4

Penalty	C	Accuracy	Rank
l1	0.001	0.495139	7
l1	0.01	0.871302	6
l1	0.1	0.913567	5
l1	1	0.933009	4
l1	10	0.933432	2
l1	100	0.933643	1
l1	1000	0.933221	3

Penalty	C	Accuracy	Rank
l1	0.001	0.495139	7
l1	0.01	0.710059	6
l1	0.1	0.899620	5
l1	1	0.925402	1
l1	10	0.924979	2
l1	100	0.924979	2
l1	1000	0.924556	4

Top-left: min_df = 2, LSI;
Top-right: min_df = 5, LSI;
Bottom-left: min_df = 2, NMF.

According to the grid search results, we got the best parameters as shown below.

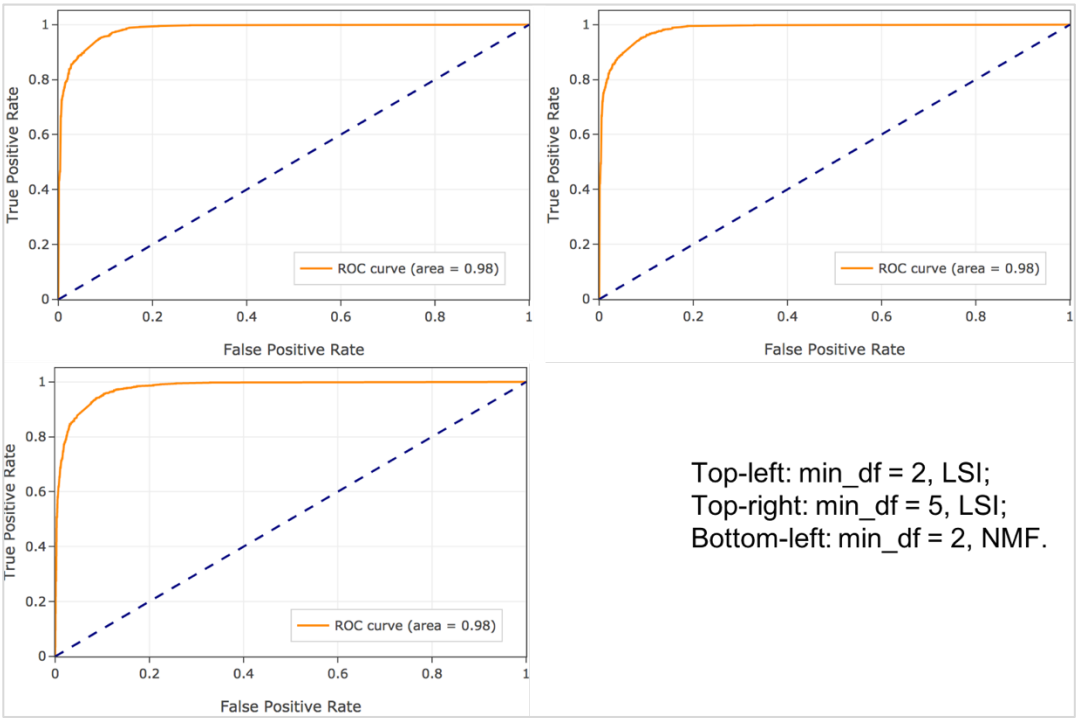


Figure 4.6. Logistic Regression (L1 norm) ROC

Table 4.7. Logistic Regression (L1 norm) Results

Parameters	Best 'C'	Accuracy	Precision	Recall	Confusion Matrix
min_df = 2, LSI	1.0	0.931111	0.931796	0.930933	[[1423 137] [80 1510]]
min_df = 5, LSI	100.0	0.930794	0.931691	0.930588	[[1418 142] [76 1514]]
min_df = 2, NMF	1.0	0.926032	0.926339	0.925913	[[1425 135] [98 1492]]

Table 4.8. Logistic Regression Grid Search Scores (L2 norm)

Penalty	C	Accuracy	Rank
l2	0.001	0.909552	7
l2	0.01	0.916737	6
l2	0.1	0.927303	5
l2	1	0.936179	1
l2	10	0.935123	2
l2	100	0.934277	3
l2	1000	0.933855	4

Penalty	C	Accuracy	Rank
l2	0.001	0.909975	7
l2	0.01	0.917371	6
l2	0.1	0.927092	5
l2	1	0.935334	1
l2	10	0.933643	3
l2	100	0.933855	2
l2	1000	0.933432	4

Penalty	C	Accuracy	Rank
l2	0.001	0.882502	7
l2	0.01	0.883981	6
l2	0.1	0.900465	5
l2	1	0.918005	4
l2	10	0.924556	3
l2	100	0.925190	1
l2	1000	0.925190	1

Top-left: min_df = 2, LSI;
Top-right: min_df = 5, LSI;
Bottom-left: min_df = 2, NMF.

According to the grid search results, we got the best parameters as shown below.

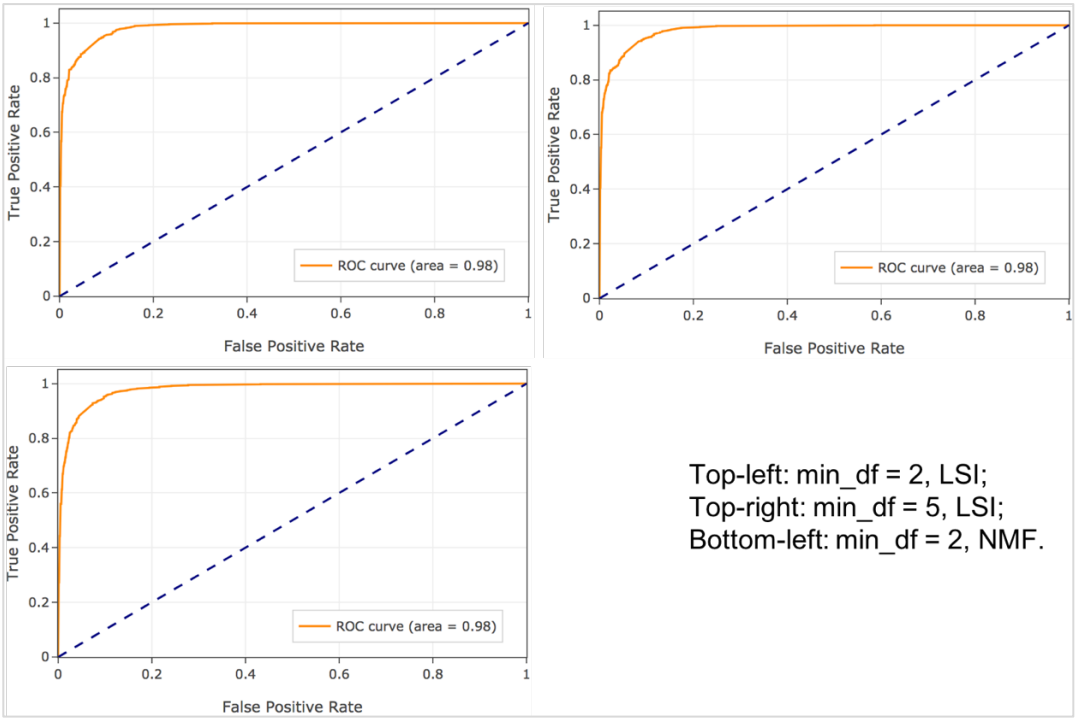


Figure 4.7. Logistic Regression (L2 norm) ROC

Table 4.9. Logistic Regression (L2 norm) Results

Parameters	Best 'C'	Accuracy	Precision	Recall	Confusion Matrix
min_df = 2, LSI	1.0	0.928571	0.929208	0.928399	[[1420 140] [85 1505]]
min_df = 5, LSI	1.0	0.927937	0.929010	0.927709	[[1410 150] [77 1513]]
min_df = 2, NMF	100.0	0.924127	0.924713	0.923960	[[1414 146] [93 1497]]

In summary, observing the results we got in the tables, l1 norm and l2 norm can get similar best accuracy, but obviously the best results we got are not under the same scenario. That is to say, there is not a fixed best combination of parameters. If applies LSI, l2 norm always performs better than l1 norm with the parameter 'C = 1', generating the best accuracy no matter what value is min_df. If applies NMF, the result reverses that l1 norm becomes more effective with C = 1 when min_df = 2 and C = 10 when min_df = 5. However, no matter which regularization method we use, smaller C tends to perform worse.

Theoretically, regularization is to add a penalty factor to loss function; and l1 norm method is to add an item associated with the l1 norm of parameter vectors, so does l2 norm. L1 norm can make weight matrix sparse (more zeros in the matrix) to realize feature selection, while l2 norm tends to make all the weights smaller to avoid overfitting.

To observe the effect that regularization makes on the coefficients (weights), we print the coefficients when applying l1 norm, l2 norm and without regularization (Appendix A). Since the logistic regression class in python set l2 norm as default, we set C to 1e8 to simulate removing regularization.

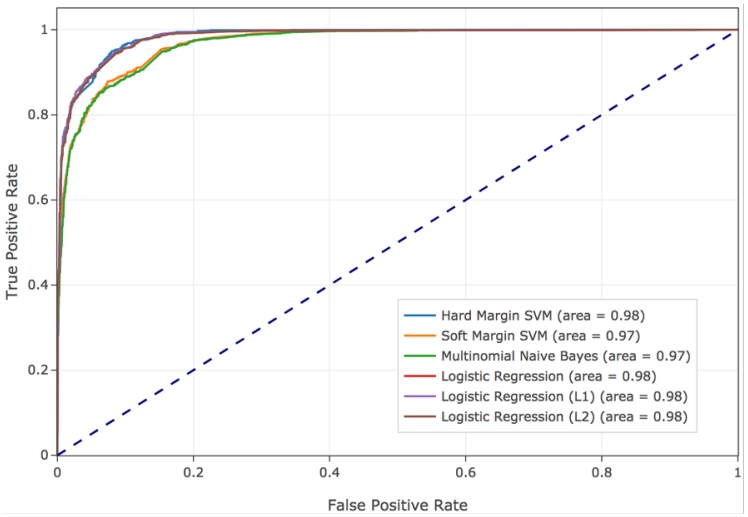
According to the following results (min_df = 2, LSI), we can see that without regularization, the absolute values of coefficients are bigger than those with regularization. Comparing l1 norm with l2 norm, some of the coefficients are 0 under l1 norm, while nearly all the coefficients have small absolute value under l2 norm. This is how they work for reducing the number of features and avoiding overfitting respectively as aforementioned.

4.3.5 Conclusion

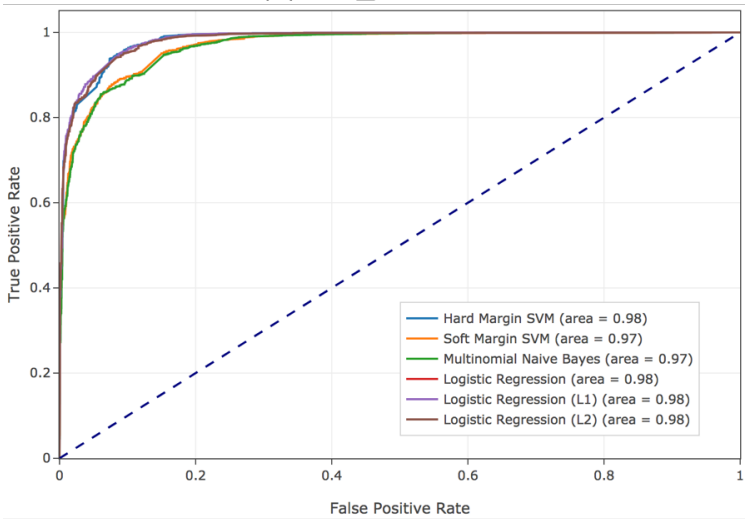
To figure out which binary classification model performs the best under certain scenario, we drew the ROC curves of them together in the same plot (Figure 4.8). In conclusion, the ROC curves illustrate that in our task, Hard Margin SVM and Logistic Regression performs nearly the same and are the best classification models. However, regularization (l1 norm and l2 norm) for Logistic Regression does not make a big difference to the result. When threshold is small, Hard Margin SVM performs slightly better than Logistic Regression. Moreover，Soft Margin SVM generates the worst results under every parameter setting conditions.

Apart from this, in our tasks, applying LSI can get a better accuracy than NMF.

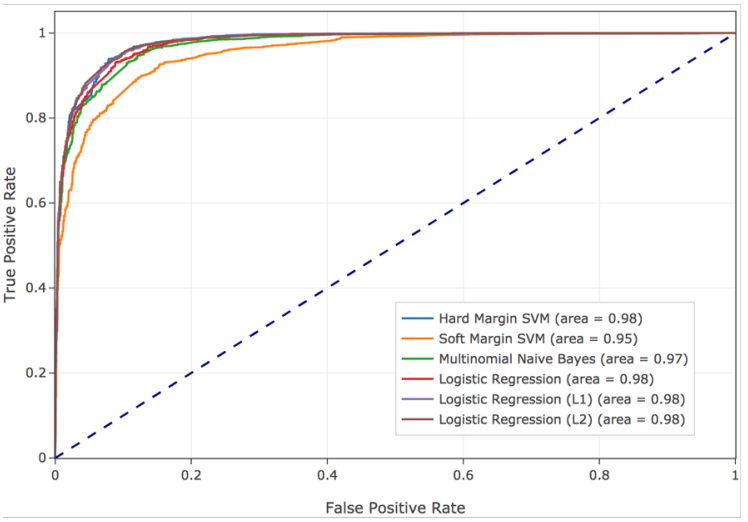
Last but not least, parameter 'min_df = 2' tend to get better results than 'min_df = 5' in all classifiers, especially the number of true positive and true negative. It is probably because 'min_df = 5' removes so much information from the documents.



(a) min_df = 2, LSI



(b) min_df = 5, LSI



(c) min_df = 2, NMF

Figure 4.8. ROC curves of six classifiers

4.4 Multiclass Classification (the last problem)

After trying to classify documents into two topics, in this part, we will propose the methods and results we got when classified documents into four topics:

- Comp.sys.ibm.pc.hardware
- Comp.sys.mac.hardware
- Misc.forsale
- soc.religion.christian.

4.4.1 Naïve Bayes Classifier

In fact, Naïve Bayes algorithm finds the class with maximum likelihood given the data, regardless of the number of classes. Therefore, the code is basically the same as the Multinomial Naïve Bayes mentioned in binary classification part.

The results are shown below:

When min_df = 2, and applied LSI:

```
                          precision    recall  f1-score   support

comp.sys.ibm.pc.hardware       0.66      0.80      0.73       392
   comp.sys.mac.hardware       0.75      0.62      0.68       385
            misc.forsale       0.87      0.78      0.82       390
  soc.religion.christian       0.88      0.94      0.91       398

             avg / total       0.79      0.78      0.78      1565

Accuracy:     0.784665
Precision:    0.789425
Recall:       0.783319
Confusion Matrix:
[[314  40  22  16]
 [105 237  21  22]
 [ 49  23 304  14]
 [  5  17   3 373]]
```

When min_df = 2, and applied NMF:

```
                          precision    recall  f1-score   support

comp.sys.ibm.pc.hardware       0.64      0.74      0.69       392
   comp.sys.mac.hardware       0.71      0.64      0.67       385
            misc.forsale       0.85      0.73      0.79       390
  soc.religion.christian       0.89      0.96      0.92       398

             avg / total       0.77      0.77      0.77      1565

Accuracy:     0.769329
Precision:    0.772742
Recall:       0.767981
Confusion Matrix:
[[290  72  19  11]
 [ 93 247  24  21]
 [ 61  27 285  17]
 [  8   2   6 382]]
```

In summary, when min_df = 2, and applying LSI, Naïve Bayes model gets the better results.

4.4.2 SVM Classifier

For SVM classifier, there are two ways to extend the binary classifier to multiclass classifier, one is called “One VS One” and the other is called “One VS the rest”. “One VS One” means training n*(n-1)/2 classifiers, where n is the number of class. Then test data with these classifiers and classify each test data into the class with highest total vote by these classifiers. While “One VS the Rest” method reduces the number of classifiers by fitting one classifier per class.

“One VS One”

To realize this method, we just need to set the parameter of SVC ‘decision_function_shape = ovo’.

The results are shown below:

When min_df = 2, and applied LSI:

```
                          precision    recall  f1-score   support

comp.sys.ibm.pc.hardware       0.66      0.82      0.73       392
   comp.sys.mac.hardware       0.73      0.66      0.70       385
            misc.forsale       0.88      0.78      0.83       390
  soc.religion.christian       0.96      0.91      0.94       398

             avg / total       0.81      0.80      0.80      1565

Accuracy:     0.796805
Precision:    0.806761
Recall:       0.795750
Confusion Matrix:
[[321  51  17   3]
 [105 256  17   7]
 [ 49  29 306   6]
 [ 11  15   8 364]]
```

When min_df = 2, and applied NMF:

```
                          precision    recall  f1-score   support

comp.sys.ibm.pc.hardware       0.51      0.88      0.64       392
   comp.sys.mac.hardware       0.77      0.41      0.53       385
            misc.forsale       0.88      0.71      0.79       390
  soc.religion.christian       0.95      0.88      0.91       398

             avg / total       0.78      0.72      0.72      1565

Accuracy:     0.720128
Precision:    0.776302
Recall:       0.718103
Confusion Matrix:
[[343  28  16   5]
 [206 156  18   5]
 [ 85  19 278   8]
 [ 44   0   4 350]]
```

"One VS Rest"

To realize this method, we imported class 'LinearSVC' from 'sklearn.svm' package. The results are shown below:

When min_df = 2, and applied LSI:

```
                          precision    recall  f1-score   support

comp.sys.ibm.pc.hardware       0.73      0.73      0.73       392
   comp.sys.mac.hardware       0.70      0.75      0.72       385
            misc.forsale       0.85      0.81      0.83       390
  soc.religion.christian       0.96      0.94      0.95       398

             avg / total       0.81      0.81      0.81      1565

Accuracy:     0.808307
Precision:    0.810544
Recall:       0.807526
Confusion Matrix:
[[288  74  26   4]
 [ 68 287  23   7]
 [ 37  33 316   4]
 [  3  15   6 374]]
```

When min_df = 2, and applied NMF:

```
                          precision    recall  f1-score   support

comp.sys.ibm.pc.hardware       0.66      0.77      0.71       392
   comp.sys.mac.hardware       0.68      0.67      0.68       385
            misc.forsale       0.86      0.77      0.81       390
  soc.religion.christian       0.96      0.93      0.94       398

             avg / total       0.79      0.78      0.79      1565

Accuracy:     0.784665
Precision:    0.791419
Recall:       0.783567
Confusion Matrix:
[[301  65  23   3]
 [101 257  20   7]
 [ 46  37 300   7]
 [  7  17   4 370]]
```

In conclusion, generally in multiclass classification tasks, SVM (both "One VS One" and "One VS Rest") perform better than Naïve Bayes. Comparing two SVM multiclass classification methods, "One VS Rest" runs faster than "One VS One" and gets a better result according to accuracy, precision and recall.

5. Additional Work

5.1 More Realistic Training

In this part, we will explain the difference of results under two circumstances: whether to remove ‘headers', 'footers' and 'quotes' in documents or not.

To find out the influence of it, we ran our code with whole document data to get the top 10 most significant terms for certain class in problem (c).

The results are shown below:

- comp.sys.ibm.pc.hardware: drive, scsi, ide, use, line, subject, organ, card, control, disk
- comp.sys.mac.hardware: line, mac, subject, organ, quadra, use, simm, appl, problem, drive
- misc.forsale: line, subject, sale, organ, 1, 2, univers, new, use, offer
- soc.religion.christian: god, christian, jesus, church, subject, peopl, line, say, christ, believ

As we can see, “line”, “organ” and “subject” nearly show up in all the classes, which are in fact the common content in ‘headers’, ‘footers’ or ‘quotes’. Our classification model might be overfitting due to this situation, which means even though we may get a slightly low accuracy by removing these content, it actually makes more sense if we want to get robust results when the test data are documents from other window of time.

As result, we removed all ‘headers’, ‘footers’ and ‘quotes’ when downloading the dataset.

5.2 Different ‘stem’ Packages Comparison

During the project, we found that there are several packages in python that we can use to extract “stem”, and we tried ‘PorterStemmer’ and ‘SnowballStemmer’. ‘SnowballStemmer’ is an updated version of ‘PorterStemmer’, which removes more information and makes the textual data more concise. So we decided to use ‘PorterStemmer’ throughout the project.

Here, we showed the difference between the results of problem (c) when applying both packages with the parameter min_df = 2:

For ‘SnowballStemmer’:

- comp.sys.ibm.pc.hardware: drive, scsi, use, ide, card, disk, control, ani, bio, jumper
- comp.sys.mac.hardware: mac, use, appl, drive, simm, problem, scsi, quadra, doe, duo
- misc.forsale: 1, 2, dos, new, sale, offer, use, includ, ship, price
- soc.religion.christian: god, christian, jesus, church, christ, peopl, say, bibl, believ, doe

For ‘PorterStemmer’:

- comp.sys.ibm.pc.hardware: drive, thi, scsi, use, ide, card, disk, control, ani, bio
- comp.sys.mac.hardware: thi, mac, use, appl, drive, simm, problem, ha, scsi, quadra
- misc.forsale: 1, 2, thi, new, sale, offer, use, includ, ship, price
- soc.religion.christian: thi, god, wa, christian, jesu, hi, church, christ, peopl, say

5.3 Stemming and Lemmatization

Apart from stemming, lemmatization[3] is another way to extract valid terms from textual data. For example, stemming might return just *s* when input is token *saw*, whereas lemmatization would return either *see* or *saw* depending on whether the use of the token was as a verb or a noun.

We did a little comparison between these two methods under the parameter setting "min_df = 2, applying LSI". In fact, in our case, the result nearly the same, but only when using Logistic Regression, lemmatization preprocess method performs better than stemming preprocess method.

The ROC curve for lemmatization preprocess method are shown below:

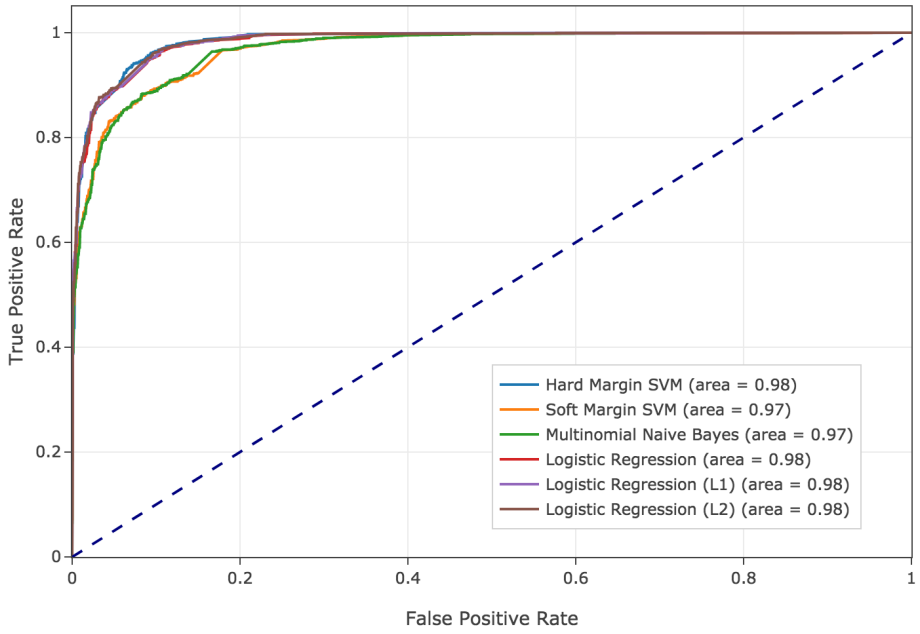


Figure 5.1. ROC curves of six classifiers (min_df = 2, LSI) (lemmatization)

5.4 Interactive Results

Along with the codes and this report, we also submitted an html file, which is a short interactive report generated from our IPython Notebook. The main features of that html file are:

1. A 'Show/Hide codes' button:
   By default, all the codes are hidden when opening this html file, making it clear to read the results for every problem. One can click the button to show all the codes and click another time to hide them again.
2. Interactive table:
   The table can be downloaded or edited online. Besides, one can change the order of the column by simply dragging.
3. Interactive plots:
   When hovering over the plots, several functions supported by Plotly will appear on the top right corner, such as downloading, editing, zooming, and panning. Also, the exact data on the plot will show when hovering over the lines. We found the interactive plots are quite helpful for analysis when the ROC curves are close to each other in one plot.

We can first focus on a specific area by selecting it using the mouse. Then click on the legends to choose which lines we are interested in. Finally, just double-click the plot, it will auto-scale back.

Reference

[1] Twenty newsgroups dataset. http://scikit-learn.org/stable/datasets/twenty_newsgroups.html
[2] Plotly for Python. https://plot.ly/python/
[3] Stemming and lemmatization. https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

Appendix

A. Coefficients of Logistic Regression Classifiers

L1 norm regularization:

```
('coefficent:', array([[ -4.42526468e+00,   1.98068879e+01,  -4.82262681e-01,
         -6.14901361e+00,   2.18237921e+00,   7.74755044e+00,
         -5.69214810e-01,   9.20365358e+00,  -1.10466867e+00,
          7.83936181e-01,  -4.40529575e-01,  -1.75186274e+00,
         -1.18899334e+00,   0.00000000e+00,  -7.03111482e-01,
          9.95966893e-01,  -2.43641639e+00,  -1.44291439e+00,
         -2.45226503e+00,   3.97813521e-01,   4.43606477e+00,
          1.53334212e+00,  -1.49983878e-02,   4.61429393e-01,
          4.44829381e-01,  -2.10183591e-01,   1.32119186e-01,
          1.39315681e+00,   1.94557804e+00,  -3.89837582e-01,
          2.42216068e+00,   1.82406056e+00,  -6.45018054e-01,
          3.71986448e-01,   0.00000000e+00,  -5.85838277e-01,
          1.22966999e+00,  -1.10292643e+00,  -6.30247583e-01,
          0.00000000e+00,   9.19994540e-02,   1.12625619e+00,
         -6.24980849e-01,   1.56182409e+00,   9.10966495e-01,
         -3.01258852e-01,  -8.69242890e-01,  -8.83055628e-01,
          0.00000000e+00,  -8.53623206e-01]]))
```

L2 norm regularization:

```
('coefficent:', array([[ -3.11426459,  14.21129442,  -0.59796853,  -4.17478472,
          1.22960194,   5.0355193 ,  -0.72766903,   6.15163079,
         -0.37001079,   1.65068324,  -0.87916643,  -1.67124533,
         -0.84825377,   0.05332459,  -0.91484649,   0.66584497,
         -1.73280009,  -1.07093175,  -1.78995951,   0.32948373,
          2.66042079,   0.96668206,   0.08672121,   0.62416975,
          0.43323224,  -0.16089791,   0.13757595,   1.02369714,
          1.29005042,  -0.8340294 ,   1.64587951,   1.36569811,
         -0.61887699,   0.49225567,   0.1397421 ,  -0.7846443 ,
          1.15594766,  -0.92047784,  -0.66691528,  -0.3668202 ,
          0.38627897,   0.75562802,  -0.40856064,   1.48143344,
          0.66977181,  -0.35517872,  -0.87120264,  -0.56989733,
          0.05409755,  -0.84041867]]))
```

Without regularization:

```
('coefficent:', array([[ -5.25803108,  22.39510392,  -1.1455659 ,  -7.40057418,
          4.1823438 ,   9.76541256,  -1.0404641 ,  11.27537643,
         -2.0178457 ,   1.82053011,  -1.76396306,  -2.30701983,
         -1.95353184,   0.291952  ,  -0.73234463,   1.42242317,
         -3.55503586,  -2.26441386,  -2.72227926,   1.50159043,
          5.56658809,   2.25244324,  -0.79055274,   1.19348346,
          0.97936134,  -0.53316748,   0.55930007,   1.77123727,
          2.75874785,  -0.50863624,   2.67947271,   2.37733137,
         -1.01570537,   0.75090801,  -0.0367912 ,  -0.86707447,
          1.5926107 ,  -1.70346435,  -0.96507594,  -0.10119033,
          0.25758759,   1.60539889,  -1.03306586,   1.98024025,
          1.54766961,  -0.70821647,  -1.23148164,  -1.43441788,
          0.10909756,  -0.94258853]]))
```