EE219 Large Scale Data Mining

Winter 2018

# Project 1 Classification Analysis on Textual Data

Yucong Wang 305036163

Shizhong Hao 605035020

## 1 Introduction

In this project, we work with "20 Newsgroups" dataset. It is a collection of approximately 20,000 newsgroup documents, partitioned evenly across 20 different newsgroups, each corresponding to a different topic. Each newsgroup document is a textual string that contains raw information. Our goal is to correctly classify all the documents into their corresponding classification. We used the training dataset to build the classifier and then used the testing dataset to verify the effectiveness of the classifier. While we train the classifier, features of the newsgroup documents are extracted after some pre-processing of the raw data. The documents were mostly classified using several classifiers such as Naive Bayesian, SVMs and Logistic Regressors. The report below presents the techniques we used, also the plots and results we obtained.

## 2 Question a

For the purposes of this project we will be working with only 8 of the classes as shown in Table 1. First we loaded the training and testing data for the following 8 subclasses of two major classes 'Computer Technology' and 'Recreational activity'.

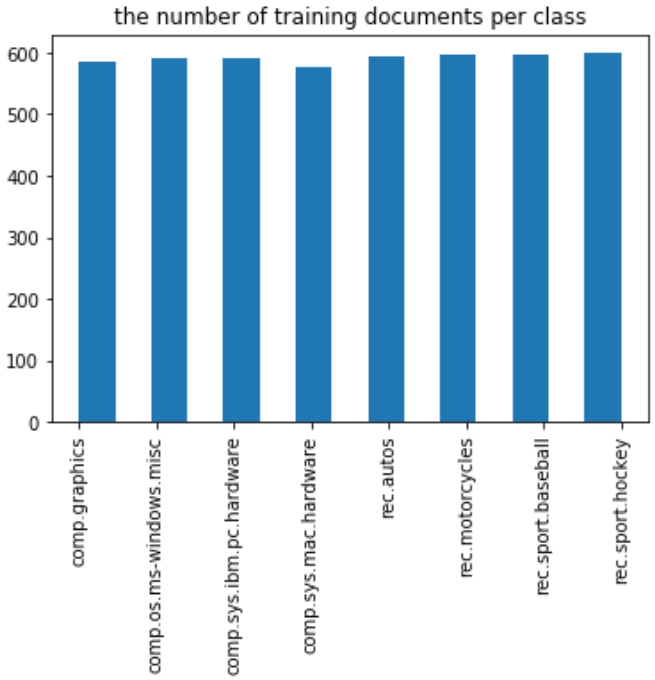
Table 1. Subclasses of 'Computer technology' and 'Recreational activity'

| Computer technology | Recreational activity |
|---|---|
| comp.graphics | rec.autos |
| comp.os.ms-windows.misc | rec.motorcycles |
| comp.sys.ibm.pc.hardware | rec.sport.baseball |
| comp.sys.mac.hardware | rec.sport.hockey |

Before we carry out the classification analysis, we have to make sure that the dataset is properly balanced with respect to every collection of documents that we are interested in. It is super important for the classifier to build accurately. To get started, we plotted a histogram of the number of training documents per class to make sure they are evenly distributed.

Result:

```
(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)
('Computer Technology ', 2343.0)
('Recreational Activity', 2389.0)
```




When we look at all train sets, it is obvious to see that the number of documents in each class are very close. It is reasonable to guess that the numbers of documents are properly and evenly distributed in all classes, as well as in the subclasses of Computer Technology and Recreational Activity.

## 3 Question b

The goal of this section was to first tokenize each document into words. Then, excluding the stop words, punctuations, and using stemmed version of words, create a TFxIDF vector representations.

Packages: We used stopwords as text.ENGLISH_STOP_WORDS from sklearn package, and we used SnowballStemmer, CountVectorizer, TfidfTransformer from nltk package.

First we removed newsgroup headers, footer signatures and quotes of every post in the dataset when fetching data groups, because means information is surely irrelevant to the content of the post. Then removed \r \n \t and punctuations in the data, and the Count Vectorizer was used to count word occurances and stopwords were removed. The whole procedure is a pre-processing for all the datasets, and the documents were processed.

Finally, a TFxIDF transformer from the Sklearn package was used to build the TFxIDF matrix. Using our TFxIDF matrix, extracted term number from train dataset is 16449 when min_df=2; extracted term number is 6910 when min_df=5.

## 4 Question c

Our task is to find the 10 most significant terms in the 4 classes. First, we append the documents in one class as one document that we did in Question b, before excluding punctuations and finding the stemmed word. Then we did pre-processing of the training document as before. After that, we also used Count Vectorizer and TFIDF method to extract the terms and wrote a function to go over every class based on index to get the top ten words.

The 10 most significant terms we found in the 4 classes are shown below, we also list them in a table.

```
top_ten = get_top_words(ibm)
print "top ten terms for comp.sys.ibm.pc.hardware"
print (top_ten)
```

```
top ten terms for comp.sys.ibm.pc.hardware
[u'card', u'organ', u'subject', u'com', u'line', u'use', u'ide', u'edu', u'drive', u'scsi']
```

```
top_ten = get_top_words(mac)
print "top ten terms for comp.sys.mac.hardware "
print (top_ten)
```

```
top ten terms for comp.sys.mac.hardware
[u'scsi', u'simm', u'appl', u'use', u'quadra', u'organ', u'subject', u'line', u'mac', u'edu']
```

```
top_ten = get_top_words(misc)
print "top ten terms for misc.forsale "
print (top_ten)
```

```
top ten terms for misc.forsale
[u'com', u'univers', u'new', u'post', u'organ', u'subject', u'sale', u'line', u'00', u'edu']
```

```
top_ten = get_top_words(christian)
print "top ten terms for soc.religion.christian "
print (top_ten)
```

```
top ten terms for soc.religion.christian
[u'believ', u'say', u'line', u'peopl', u'subject', u'church', u'jesus', u'edu', u'christian', u'god']
```

## 10 most significant terms in these 4 classes

| Comp.sys.ibm.pc.hardware | Comp.sys.mac.hardware | Misc.forsale | Soc.religion.christian |
|---|---|---|---|
| card | scsi | com | believ |
| organ | simm | univers | say |
| subject | appl | new | line |
| com | use | post | people |
| line | quadra | organ | subject |
| use | organ | subject | church |
| ide | subject | sale | jesus |
| edu | line | line | edu |
| drive | mac | 00 | christian |
| scsi | edu | edu | god |

The 10 most significant terms in these 4 classes are consistent with terms we expected. For example, the words in the soc.religion.christian class like church and god are very relevant to the topic.

We notice that this is a result straight after stemming, without any modification. And due to how current stemming engines work, some words no longer maintain its proper spelling (but nonetheless “readable”). For example, “people” becomes “peopl”, “please” becomes “pleas”. The idea behind this is just that the stemming engine only care about making sure different stems of the same word become the exact same word after stemming, but not care if the resulting word spelling is also correct. It may cause the reader to feel a bit weird, but this is good for making extraction and prediction.

## 5 Question d

For this question, we are expected to apply LSI to the TFxIDF matrix to reduce dimensionality corresponding to the 8 classes. and set k=50; so each document is mapped to a 50-dimensional vector.

The principle behind LSI is that some words that are used in the same contexts usually have meanings that are similar. We reduce the features to lower dimensional space by representing data in term document matrix, with columns of TFxIDF representation of documents that we have already got. To increase the performance of any classifier, dimensionality of the data must be reduced, usually by selecting a non-sparse subset of the total features. We used Latent Semantic Indexing (LSI), which reduces dimensionality using mean-squared errors. Steps are as followed:

First, we imported NMF and TruncatedSVD from sklearn package, then did the pre-processing for all the datasets, and used TFxIDF matrix to fit and transform the data, extracted term number from train dataset is 16449 when min_df=2; extracted term number is 6910 when min_df=5. After applying LSI or NMF method, the dimension goes down to 50.

Alternatively, we use Non-Negative Matrix Factorization (NMF) to do the same thing, and the dimension also goes down to 50.

## 6 Question e

For the following questions, our task would be to classify the documents into two categories Computer Technology vs Recreational Activity and perform the binary classification. We used hard margin SVM classifier (SVC) by setting $\gamma$ to 1000, and soft margin SVM classifier (SVC) by setting $\gamma$ to 0.001.

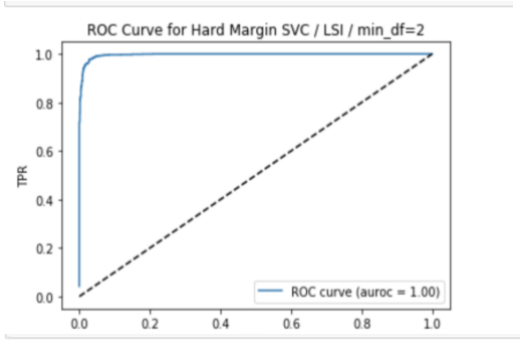
First, we combined documents of sub-classes of each class to form the 2 classes, Computer Technology and Recreational Activity. Then we performed the pre-processing procedure to train set and test set. We imported pipeline form sklearn package, and designed the pipeline of CountVectorizer, tfidfTransformer, LSI(or NMF) and SVC(with changeable C) method. We finished different combination of hard/soft margin SVC, LSI/NMF and min_df=2/min_df=5 to

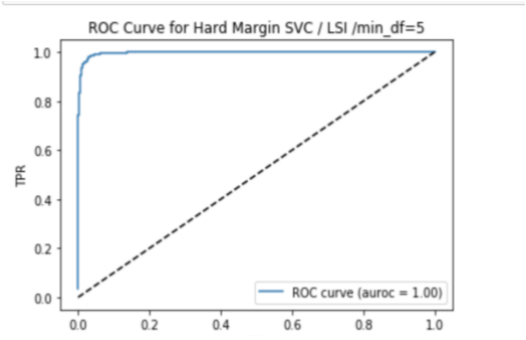
compare the results.

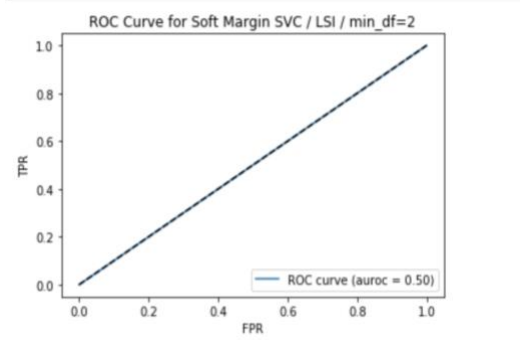
We report the confusion matrix and calculate the accuracy, recall and precision of different classifiers. We also plot ROC, to observe the trade-off between the two components of the predictions. We plot the false positive rate (x-axis) vs the true positive rate (y-axis). The results are as followed.

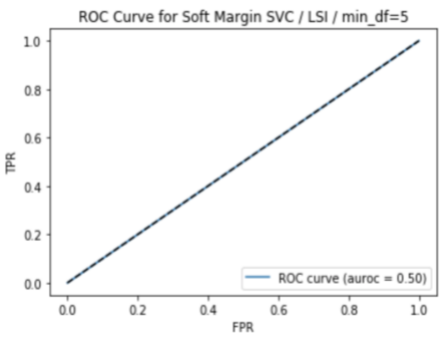
| Mindf | Hard margin& LSI | Soft margin & LSI | Hard margin & NMF | Soft margin & NMF |
|---|---|---|---|---|
| 2 | **confusion matrix** | **confusion matrix** | **confusion matrix** | **confusion matrix** |
| | [[1506 54]<br>[ 31 1559]] | [[ 0 1560]<br>[ 0 1590]] | [[1493 67]<br>[ 46 1544]] | [[ 0 1560]<br>[ 0 1590]] |
| | **accuracy** | **accuracy** | **accuracy** | **accuracy** |
| | 0.973016 | 0.504762 | 0.964127 | 0.504762 |
| | **precision** | **precision** | **precision** | **precision** |
| | 0.966522 | 0.504762 | 0.958411 | 0.504762 |
| | **recall** | **recall** | **recall** | **recall** |
| | 0.980503 | 1.000000 | 0.971069 | 1.000000 |
| 5 | **confusion matrix** | **confusion matrix** | ------- | ------- |
| | [[1509 51]<br>[ 32 1558]] | [[ 0 1560]<br>[ 0 1590]] | | |
| | **accuracy** | **accuracy** | | |
| | 0.973651 | 0.504762 | | |
| | **precision** | **precision** | | |
| | 0.968303 | 0.504762 | | |
| | **recall** | **recall** | | |
| | 0.979874 | 1.000000 | | |

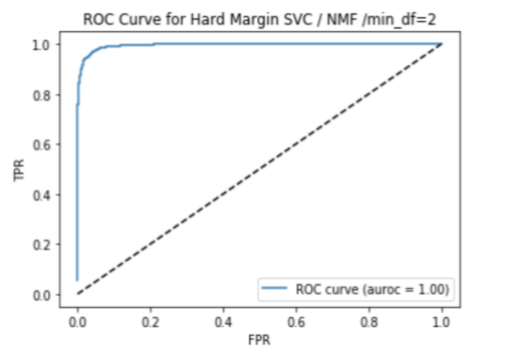
The ROC curves for each situation are as followed.

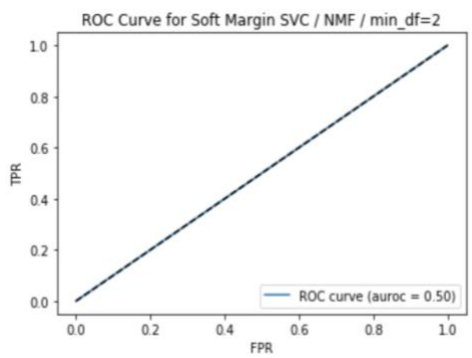



As can be seen from the data, by using a hard margin SVM, about 97% of positive events are true positive as can be shown by the precision. And we can also see, 97% of all the positive data can be detected by our algorithm, as can be shown by the recall.

From the results we learned that there are not so much difference between different situations under soft margin SVC, the accuracy and the average score maintain at about 0.5. Besides, for hard margin SVC, LSI is better than NMF for accuracy and precision. And by setting min_df higher, we could get better accuracy and precision performance in classification. Finally, we got the best accuracy at the combination of hard margin SVM, LSI method and min_df=5.

## 7 Question f

For this question, we use a 5-fold cross-validation, to find the best value of the parameter $\gamma$ in the array [0.001,0.01,0.1,1,10,100,1000]. Using the same method as question e, we designed pipelines with different $\gamma$ value, and the results are as follows.

-----using LSI/ min_df=2------

| C value | ave_score | accuracy | precision | recall |
|---|---|---|---|---|
| 0.001 | 0.504761904762 | 0.504762 | 0.504762 | 1.000000 |
| 0.01 | 0.504761904762 | 0.504762 | 0.504762 | 1.000000 |
| 0.1 | 0.504761904762 | 0.504762 | 0.504762 | 1.000000 |
| 1 | 0.939682539683 | 0.958730 | 0.932464 | 0.989937 |
| 10 | 0.973333333333 | 0.966984 | 0.956388 | 0.979245 |
| 100 | 0.977777777778 | 0.973968 | 0.966007 | 0.983019 |
| 1000 | 0.983492063492 | 0.973016 | 0.967101 | 0.979874 |

-----using NMF/ min_df=2------

| C value | ave_score | accuracy | precision | recall |
|---|---|---|---|---|
| 0.001 | 0.504761904762 | 0.504762 | 0.504762 | 1.000000 |
| 0.01 | 0.504761904762 | 0.504762 | 0.504762 | 1.000000 |
| 0.1 | 0.504761904762 | 0.504762 | 0.504762 | 1.000000 |
| 1 | 0.504761904762 | 0.504762 | 0.504762 | 1.000000 |

| | | | | |
|---|---|---|---|---|
| 10 | 0.716825396825 | 0.930794 | 0.922414 | 0.942138 |
| 100 | 0.958095238095 | 0.946984 | 0.928873 | 0.969182 |
| 1000 | 0.965079365079 | 0.959365 | 0.950123 | 0.970440 |

------using LSI/ min_df=5------

| C value | ave_score | accuracy | precision | recall |
|---|---|---|---|---|
| 0.001 | 0.504761904762 | 0.504762 | 0.504762 | 1.000000 |
| 0.01 | 0.504761904762 | 0.504762 | 0.504762 | 1.000000 |
| 0.1 | 0.504761904762 | 0.504762 | 0.504762 | 1.000000 |
| 1 | 0.952063492063 | 0.955873 | 0.928024 | 0.989308 |
| 10 | 0.973333333333 | 0.970159 | 0.957772 | 0.984277 |
| 100 | 0.97873015873 | 0.974603 | 0.966625 | 0.983648 |
| 1000 | 0.982222222222 | 0.973651 | 0.968886 | 0.979245 |

From the results, we could get the basic conclusion that when C comes larger than 1, it turns out the accuracy score is improved together with precision and recall score. For each specific method, we could always get best result at C=1000, that is the same as what we got for comparing hard margin SVC and soft margin SVC. As for the LSI and NMF method for dimension reduction, we could got better performance on accuracy, precision and also average score by using LSI.

Based on the results of average score and the accuracy, finally we choose C = 1000 as the best value, and we use min_df=2 and LSI method. The r esult at the best situation is as followed.

Accuracy: 0.973015873016

Confusion matrix:

[[1507 53]

[ 32 1558]]

Classification report:

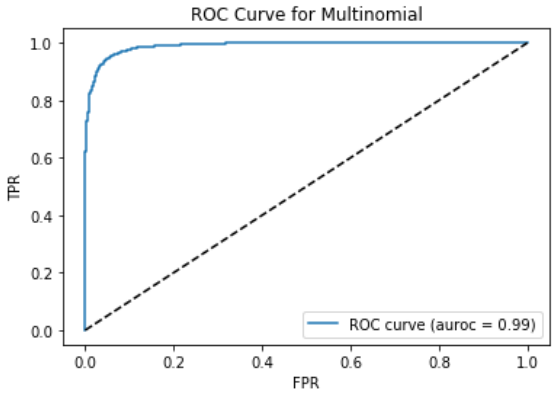
| | precision | recall | F1-score | support |
|---|---|---|---|---|
| Comp | 0.98 | 0.97 | 0.97 | 1560 |
| Rec | 0.97 | 0.98 | 0.97 | 1590 |
| Avg/total | 0.97 | 0.97 | 0.97 | 3150 |

## 8 Question g

In part g, we aimed to use a multinomial naïve Bayes classifier to perform the same task, which is binary classification. The algorithm estimates the maximum likelihood probability of a class given a document with feature set x, using Bayes rule, based upon the assumption that given the class, the features are statistically independent. Multinomial naïve Bayes classifier is able to represent the frequencies with which certain events have been generated by multinomial where p is the probability occurs.

Since we are using multinomial naïve bayes classifier, we are only able to perform an NMF in dimension reduction since LSI the data contains negative values and cannot be used to train MultinomialNB.

In this part we are using min_df=2. The ROC curve and the confusion matrix and calculate the accuracy, recall and precision of our classifier are as followed.




Result for MultinomialNB using NMF / min_df=2

confusion_matrix:

[[1400 160]

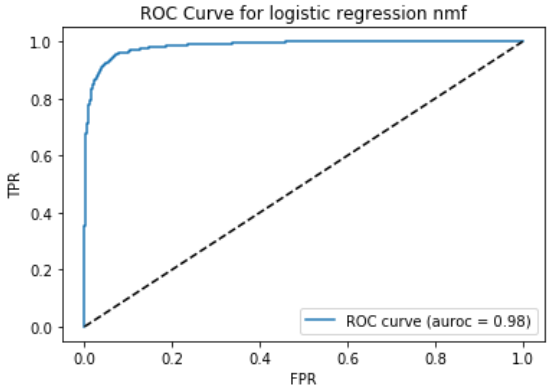
[ 35 1555]]

accuracy = 0.938095, precision = 0.906706, recall = 0.977987

Multinomial naïve bayes classifier gives us an accuracy of 0.938 which is a relatively good classifier to use. However, comparing to SVM we used in previous question, it is still less precise and accurate.

## 9 Question h

In this part, we aimed to use logistic regression classifier to perform our task again. We started with min_df=2 and using NMF to reduce dimension. The only difference is to change clf to logistic regression. The ROC curve and the confusion matrix and calculate the accuracy, recall and precision of our classifier are as followed. Logistic regression refers to a classifier that classifies an observation into one of two classes.




Result for Logistic Regression NMF / min_df=2:

confusion_matrix:

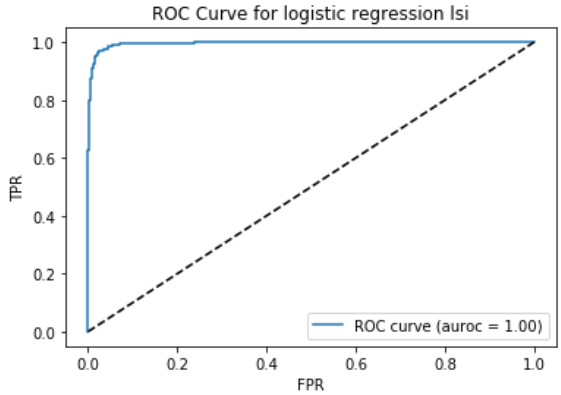
[[1450 110]

[ 74 1516]]

accuracy = 0.941587, precision = 0.932349, recall = 0.953459

Then we change our classifier to LSI with both min_df=2 and then min_df=5 to perform classification. It turns out the accuracy score is improved together with precision and recall score. It shows the LSI performs better than NMF, which is the agreement with we learned before.

Results for min_df=2:



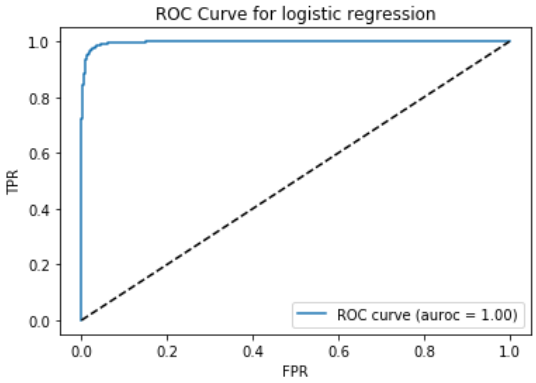

confusion_matrix:

[[1492 68]

[ 35 1555]]

accuracy = 0.967302, precision = 0.958102, recall = 0.977987

Results for min_df=5:




Result for logreg

confusion_matrix:

[[1492 68]

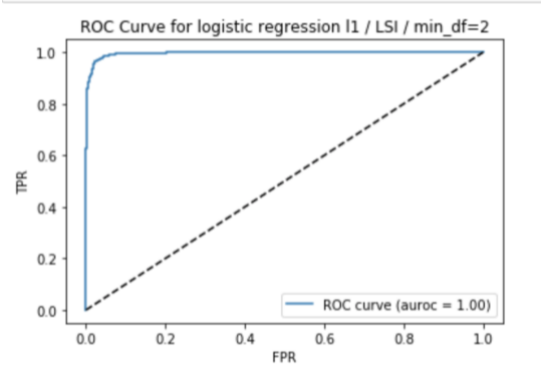
[ 30 1560]]

accuracy = 0.968889, precision = 0.958231, recall = 0.981132

From the results, we can see that min_df=5 with LSI decomposition gives us the best result and confusion matrix. It has accuracy of 96.8% and precision of 95.8%. Since LSI takes in the negative value as well, comparing to NMF it gives more accurate result on the dataset. Logistic Classifier is a relatively good classifier to complete this task.

## 10 Question i1

For the part of i1, we are asked to perform a logistic regression classification with a regularization term to optimization objective. We try to perform both l1 and l2 norm regularization in our case. Results are listed as followed.



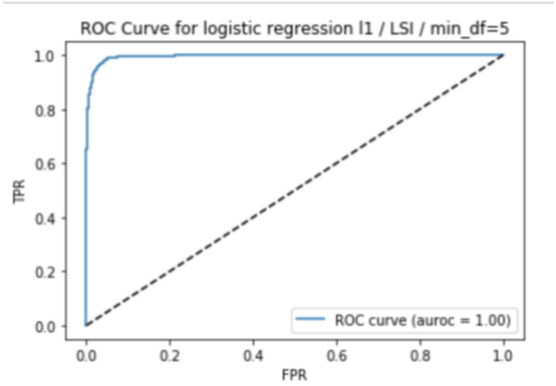

Result for logreg_L1 / LSI /min_df=2

confusion_matrix:

[[1495 65]

[ 33 1557]]

accuracy = 0.968889, precision = 0.959926, recall = 0.981132



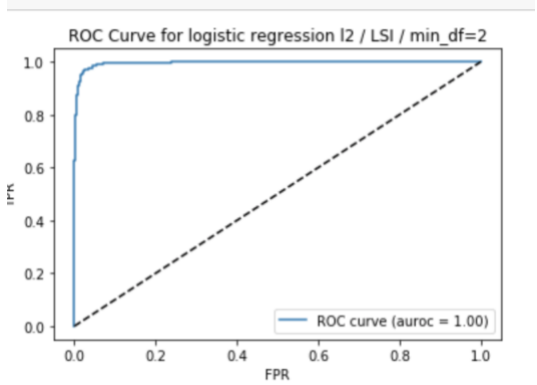

Result for logreg_L1 / LSI /min_df=5

confusion_matrix:

[[1495 65]

[ 33 1557]]

accuracy = 0.968889, precision = 0.959926, recall = 0.981132



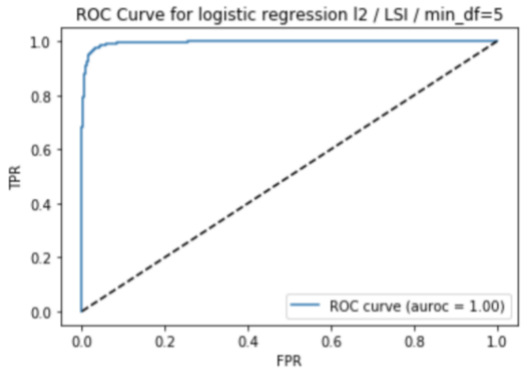

Result for logreg_L2 / LSI /min_df=2

confusion_matrix:

[[1492 68]

[ 33 1555]]

accuracy = 0.967302, precision = 0.958102, recall = 0.977987



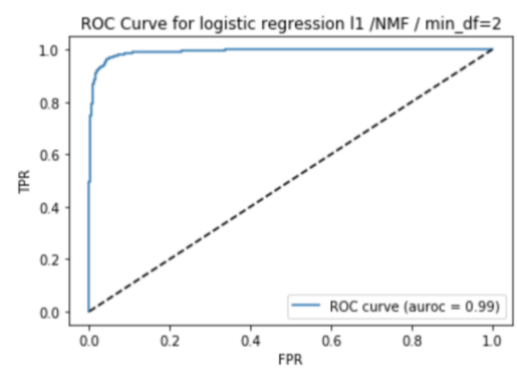

Result for logreg_L2 / LSI /min_df=5

confusion_matrix:

[[1492 68]

[ 30 1560]]

accuracy = 0.968889, precision = 0.958231, recall = 0.981132



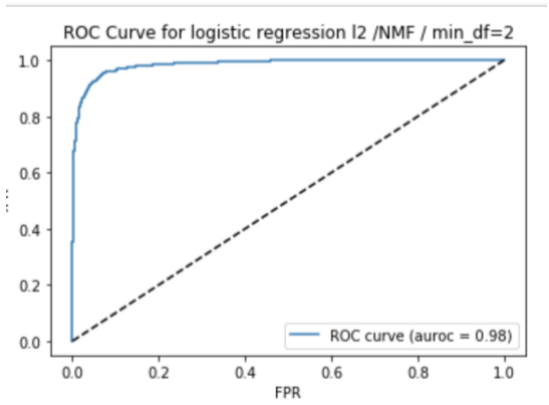
Result for logreg_L1 / NMF /min_df=2

confusion_matrix:

[[1485 75]

[ 55 1535]]

accuracy = 0.958730, precision = 0.953416, recall = 0.953459




Result for logreg_L2 / NMF /min_df=2

confusion_matrix:

[[1450 110]

[ 74 1516]]

accuracy = 0.941587, precision = 0.932349, recall = 0.953459

In the case of LSI, we can see that L1 regularization gives us slightly bette r result for the dataset the dataset.

However, when we try to use NMF for both L1 and L2. L1 still give a rela tively good result, however, the accuracy, precision of L2 drops significantl y. Therefore, in this case L1 is better.

For the rest of this part, we are going to use different c values to analysis L1 and L2 logistic regression.

For L1 analysis with different c values:

Result for c = 0.001

confusion_matrix:

[[1560 0]
[1590 0]]
accuracy = 0.495238, precision = 0.000000, recall = 0.000000
Result for c = 0.01
confusion_matrix:
[[1478 82]
[ 238 1352]]
accuracy = 0.898413, precision = 0.942817, recall = 0.850314
Result for c = 0.1
confusion_matrix:
[[1464 96]
[ 46 1544]]
accuracy = 0.954921, precision = 0.941463, recall = 0.971069
Result for c = 1
confusion_matrix:
[[1495 65]
[ 33 1557]]
accuracy = 0.968889, precision = 0.959926, recall = 0.979245
Result for c = 10
confusion_matrix:
[[1506 54]
[ 30 1560]]
accuracy = 0.973333, precision = 0.966543, recall = 0.981132
Result for c = 100
confusion_matrix:
[[1510 50]
[ 32 1558]]
accuracy = 0.973968, precision = 0.968905, recall = 0.979874
Result for c = 1000
confusion_matrix:
[[1510 50]
[ 32 1558]]
accuracy = 0.973968, precision = 0.968905, recall = 0.979874

For L2 analysis with different c values:

Result for c = 0.001

confusion_matrix:

[[ 604  956]

 [   0 1590]]

accuracy = 0.696508, precision = 0.624509, recall = 1.000000

Result for c = 0.01

confusion_matrix:

[[1393  167]

 [  10 1580]]

accuracy = 0.943810, precision = 0.904408, recall = 0.993711

Result for c = 0.1

confusion_matrix:

[[1482   78]

 [  34 1556]]

accuracy = 0.964444, precision = 0.952264, recall = 0.978616

Result for c = 1

confusion_matrix:

[[1492   68]

 [  35 1555]]

accuracy = 0.967302, precision = 0.958102, recall = 0.977987

Result for c = 10

confusion_matrix:

[[1503   57]

 [  26 1564]]

accuracy = 0.973651, precision = 0.964837, recall = 0.983648

Result for c = 100

confusion_matrix:

[[1508   52]

 [  29 1561]]

accuracy = 0.974286, precision = 0.967762, recall = 0.981761

Result for c = 1000

confusion_matrix:
[[1511 49]
[ 31 1559]]
accuracy = 0.974603, precision = 0.969527, recall = 0.980503

We also listed a table of accuracy from different C values.

| C | L1(Accuracy) | L2(Accuracy) |
|---|---|---|
| 0.001 | 0.495238 | 0.696508 |
| 0.01 | 0.898413 | 0.943810 |
| 0.1 | 0.95492 | 0.964444 |
| 1 | 0.968889 | 0.967302 |
| 10 | 0.973333 | 0.973651 |
| 100 | 0.973968 | 0.974286 |
| 1000 | 0.973968 | 0.974603 |

From the results, we can see that accuracy score increases with the increasing in c value. However, after a c-value reaches to a certain extent, the result does not improve in a significant amount of percentage. Optimization of the result is around c>=10.

## 11 Question i2

We are asked to perform multiclass classification in this part. We started with Gaussian naïve bayes. With Naïve bayes we are only interested in NMF. And the result is listed below. MultiClass classification means a single estimator is able to handle more than two classes of input and perform classification tasks of more than two classes.

GaussianNB multiclassification we are used is to perform Gaussian classification toward four different input data we have. By using a Gaussian classification, we

are able to get the estimation of the result.

Result for GaussianNB /NMF /min_df=2:
confusion_matrix:
[[291 40 53 8]
[ 76 245 61 3]
[ 53 23 301 13]
[ 2 3 0 393]]
accuracy = 0.785942, precision = 0.786945, recall = 0.785942

We could find that NMF of the GaussianNB is giving us a comparatively low accuracy, precision and recall score. The confusion matrix is not that bad. We are ignoring LSI since this is a naïve bayes classifier.

OneVsOne Classifier is actually fitting one classifier per class pair. It compares each class with the rest to get the result. It is usually slower comparing to one-vs-the-rest classifier due to the complexity as O(n^2).
Then we perform OnevsOne classifier with the combination of different situations of LSI or NMF. The results are shown below.

Result for OnevsOne_LSI / min_df=2:
confusion_matrix:
[[311 53 27 1]
[ 41 318 26 0]
[ 24 20 344 2]
[ 3 1 1 393]]
accuracy = 0.872843, precision = 0.872881, recall = 0.872834
Result for OnevsOne_NMF / min_df=2:
confusion_matrix:
[[312 49 31 0]
[ 57 297 28 3]
[ 27 16 345 2]
[ 4 3 3 388]]

accuracy = 0.857508, precision = 0.857866, recall = 0.857508

One-vs-all, this strategy consists in fitting one classifier per class. For each classifier, the class is fitted against all the other classes. In addition to its computational efficiency (only n_classes classifiers are needed), one advantage of this approach is its interpretability. Since each class is represented by one and one classifier only. We also performed OnevsRest classifier with different situations of LSI or NMF. The results are shown below.

Result for OnevsRest_LSI / min_df=2:
confusion_matrix:
[[310 61 21 1]
[ 30 331 24 0]
[ 21 18 349 2]
[ 4 1 1 392]]
accuracy = 0.883067, precision = 0.884060, recall = 0.883067

Result for OnevsRest_NMF / min_df=2:
confusion_matrix:
[[310 50 30 2]
[ 48 306 27 4]
[ 27 17 345 1]
[ 3 1 3 391]]
accuracy = 0.863898, precision = 0.863526, recall = 0.863898

In general, both OneVsOneClassifier and OneVsRestClassifier give us good result. In both case, LSI works better than NMF, however, not significantly. Both the classifiers gave better results than Gaussian naïve Bayes. Therefore, LSI and OnevsOneClassifier and OnveVsRestClassifier are preferred more to perform multiclass classification tasks.