EE219 Project 1
Classification Analysis on Textual Data

Winter 2019

Tiantong Wang	105229836	wangtiantong@ucla.edu
Yuqi Zhou	905231915	yuqi0207@ucla.edu
Ray Lin	705224483	Ray199246@yahoo.com
Miaoqing Chen	505219312	mickchen666@g.ucla.edu

Question 1 Data Import

In this project, we work with “20Newsgroups” dataset, which consists of nearly 20000 newsgroups posts partitioned on 20 topics evenly. In this project, we specifically choose 8 classes among 20. For these 8 topics, we can further divide them into two categories, computer technology and Recreational Activity, as shown in table 1.

Table 1 Two-well separated classes

Computer Technology	**Recreational Activity**
comp.graphics	rec.autos
comp.os.ms-windows.misc	rec.motorcycles
comp.sys.ibm.pc.hardware	rec.sport.basebal
comp.sys.mac.hardware	rec.sport.hockey

First, we count the number of posts in each group on training dataset to see its distribution. The result is shown in Figure 1. From the histogram figure, we find that all posts are evenly distributed in these 8 categories, indicating that the dataset is well-balanced that can avoid negative influence of the effectiveness of the classifier we need to train later brought by imbalance. In the following processing and training, we can use this dataset safely.

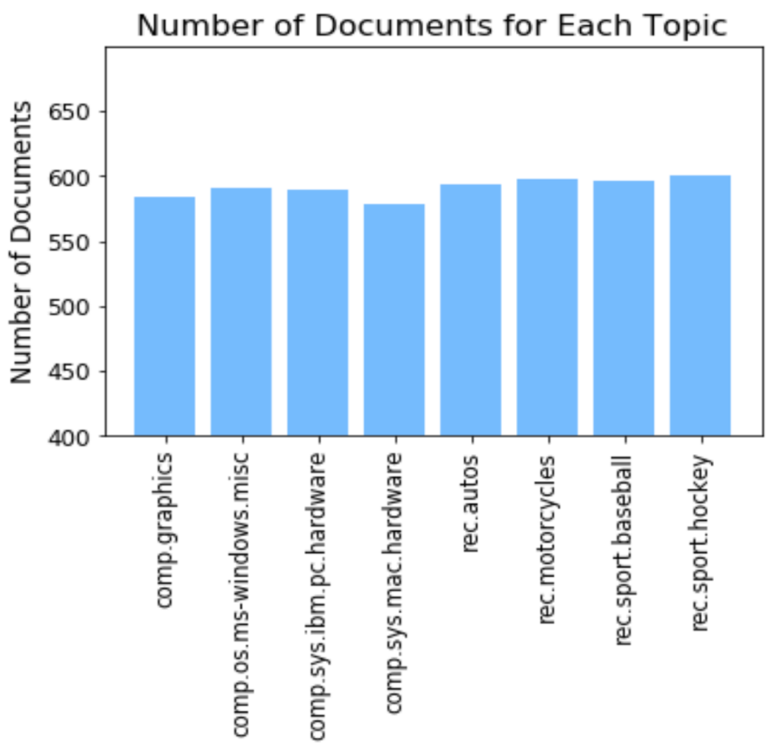


Figure 1 Distribution of documents

Question 2 Feature Extraction

First, we tokenized each document and then remove stopwords and terms that are numbers. Then we perform lemmatization to reduce inflectional forms and sometimes derivationally related forms of a word to linguistically valid lemmas [1] so that words sharing the same basic forms will be merged as one word. Here we first applied class "CountVectorizer" to generate the matrix containing count of the appearance (term frequency) of each word. Then we applied class "TfidfTransformer" to get TF-IDF matrix to represent the weight of each word in the documents. For this question, we choose 'min_df' as 3, which means ignoring terms that appear in less than 3 documents. After specifying these parameters, we learn the vocabulary dictionary based on training dataset and return count matrix and transform a count matrix to a normalized TF-IDF representation [2]. Then we extract features by applying the models mentioned above for test datasets. The shape of TF-IDF matrices of train and test subsets are shown in table 2.

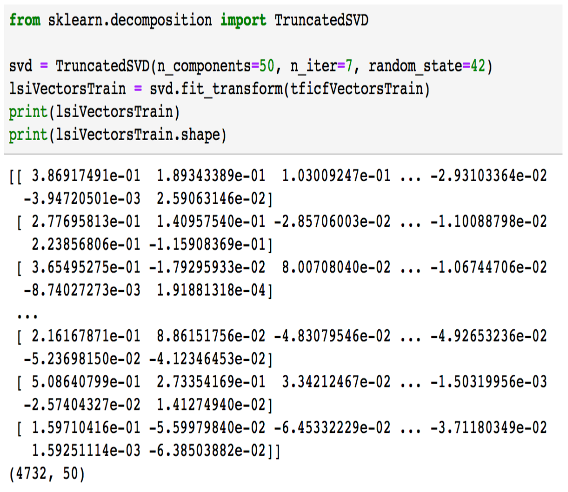
Table 2 Shape of TF-IDF matrices

	Training Dataset	**Test Dataset**
Row	4732	3150
Column	16600	16600

Question 3 Reduce Demension

The TF-IDF matrices generated from the step of feature extraction are sparse and of high dimensionality. Also, the classifiers fail to be well-trained with such high dimensional data. Therefore, we reduce the dimension of TF-IDF matrices for training dataset and test dataset. Here are two possible approaches, Latent Semantic Indexing (LSI) and Non-Negative Matrix Factorization (NMF). In this project, we aim to reduce the dimension of 50 (k=50). The shape of TF-IDF matrix for training dataset is (4730,50), and the shape of test TF-IDF matrix is (3150,50).

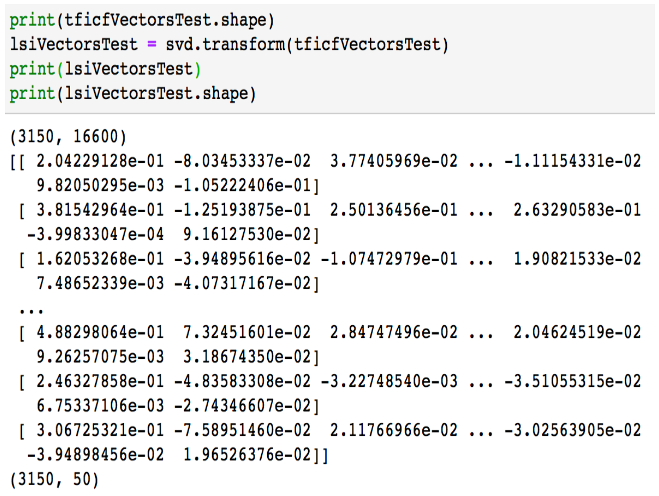
First, we use LSI based on Singular Value Decomposition. The intuition is to sort the singular values in descending order and then choose first k elements. In this way, the dimension is reduced to k. We fit on training TF-IDF matrix to get transformed matrix of U, V and Σ, and then multiple TF-IDF matrix of test dataset to V to reduce the dimension to k. Figure 2 shows the steps and results for training dataset and test dataset.



```
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=50, n_iter=7, random_state=42)
lsiVectorsTrain = svd.fit_transform(tficfVectorsTrain)
print(lsiVectorsTrain)
print(lsiVectorsTrain.shape)
```

```
[[ 3.86917491e-01  1.89343389e-01  1.03009247e-01 ... -2.93103364e-02
  -3.94720501e-03  2.59063146e-02]
 [ 2.77695813e-01  1.40957540e-01 -2.85706003e-02 ... -1.10088798e-02
   2.23856806e-01 -1.15908369e-01]
 [ 3.65495275e-01 -1.79295933e-02  8.00708040e-02 ... -1.06744706e-02
  -8.74027273e-03  1.91881318e-04]
 ...
 [ 2.16167871e-01  8.86151756e-02 -4.83079546e-02 ... -4.92653236e-02
  -5.23698150e-02 -4.12346453e-02]
 [ 5.08640799e-01  2.73354169e-01  3.34212467e-02 ... -1.50319956e-03
  -2.57404327e-02  1.41274940e-02]
 [ 1.59710416e-01 -5.59979840e-02 -6.45332229e-02 ... -3.71180349e-02
   1.59251114e-03 -6.38503882e-02]]
(4732, 50)
```

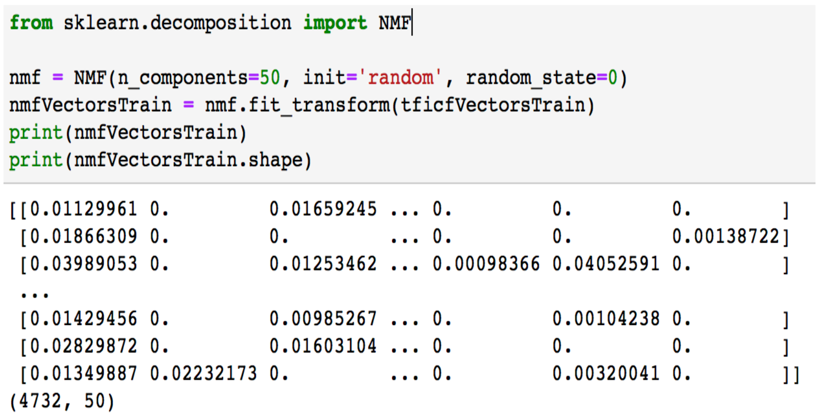


```
print(tficfVectorsTest.shape)
lsiVectorsTest = svd.transform(tficfVectorsTest)
print(lsiVectorsTest)
print(lsiVectorsTest.shape)
```

```
(3150, 16600)
[[ 2.04229128e-01 -8.03453337e-02  3.77405969e-02 ... -1.11154331e-02
   9.82050295e-03 -1.05222406e-01]
 [ 3.81542964e-01 -1.25193875e-01  2.50136456e-01 ...  2.63290583e-01
  -3.99833047e-04  9.16127530e-02]
 [ 1.62053268e-01 -3.94895616e-02 -1.07472979e-01 ...  1.90821533e-02
   7.48652339e-03 -4.07317167e-02]
 ...
 [ 4.88298064e-01  7.32451601e-02  2.84747496e-02 ...  2.04624519e-02
   9.26257075e-03  3.18674350e-02]
 [ 2.46327858e-01 -4.83583308e-02 -3.22748540e-03 ... -3.51055315e-02
   6.75337106e-03 -2.74346607e-02]
 [ 3.06725321e-01 -7.58951460e-02  2.11766966e-02 ... -3.02563905e-02
  -3.94898456e-02  1.96526376e-02]]
(3150, 50)
```

Figure 2 LSI Results (left: training right: test)

Second, we apply NMF which finds two non-negative matrix, W and H, and minimize the Frobenius norm of the difference of original matrix and WH. Matrix H is learned when applying NMF model to training TF-IDF matrix and then use it for testing TF-IDF matrix. Figure 3 shows the steps and results for training dataset and Figure 4 is for test dataset.

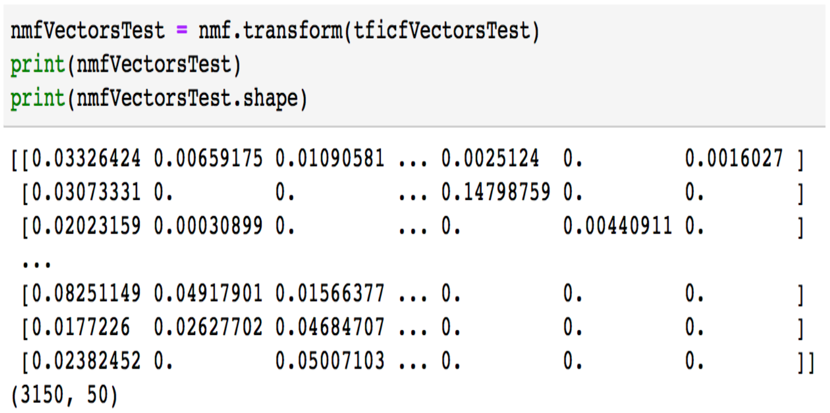


```
from sklearn.decomposition import NMF

nmf = NMF(n_components=50, init='random', random_state=0)
nmfVectorsTrain = nmf.fit_transform(tficfVectorsTrain)
print(nmfVectorsTrain)
print(nmfVectorsTrain.shape)
```

```
[[0.01129961 0.         0.01659245 ... 0.         0.         0.        ]
 [0.01866309 0.         0.         ... 0.         0.         0.00138722]
 [0.03989053 0.         0.01253462 ... 0.00098366 0.04052591 0.        ]
 ...
 [0.01429456 0.         0.00985267 ... 0.         0.00104238 0.        ]
 [0.02829872 0.         0.01603104 ... 0.         0.         0.        ]
 [0.01349887 0.02232173 0.         ... 0.         0.00320041 0.        ]]
(4732, 50)
```

Figure 3 NMF Result for training dataset

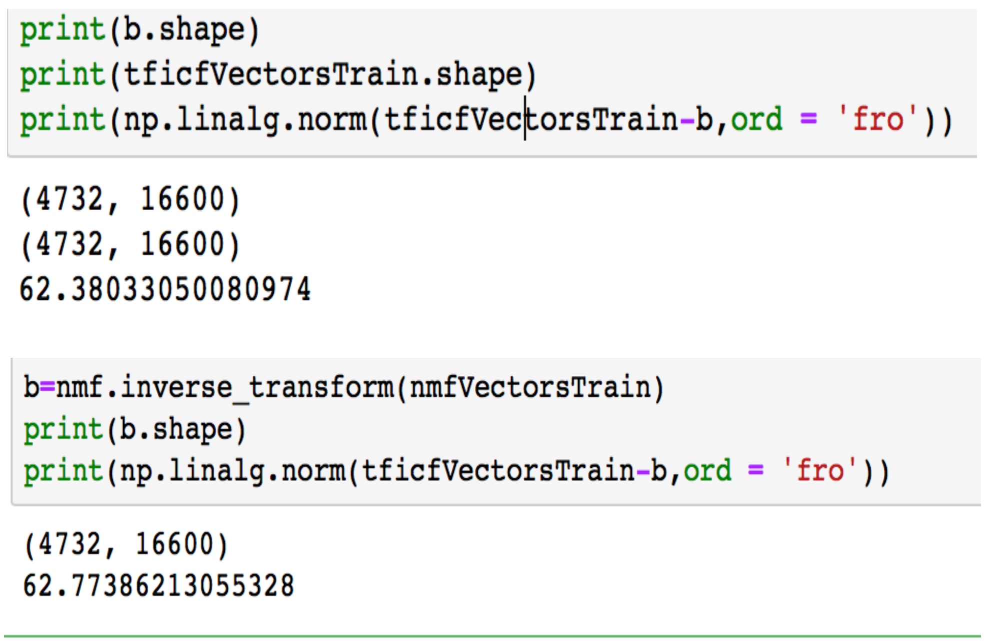


```
nmfVectorsTest = nmf.transform(tficfVectorsTest)
print(nmfVectorsTest)
print(nmfVectorsTest.shape)
```

```
[[0.03326424 0.00659175 0.01090581 ... 0.0025124  0.         0.0016027 ]
 [0.03073331 0.         0.         ... 0.14798759 0.         0.        ]
 [0.02023159 0.00030899 0.         ... 0.         0.00440911 0.        ]
 ...
 [0.08251149 0.04917901 0.01566377 ... 0.         0.         0.        ]
 [0.0177226  0.02627702 0.04684707 ... 0.         0.         0.        ]
 [0.02382452 0.         0.05007103 ... 0.         0.         0.        ]]
(3150, 50)
```

Figure 4 NMF Result for test dataset

Then we compare the performance of two approaches by comparing $||X - U_k \Sigma_k V_k^T||_F^2$ in LSI and $||X - WH||_F^2$ in NMF. We first do inverse transform to convert the new matrix with low dimension to its original dimension. And then calculate the F norm of the difference of original matrix and new-transformed matrix of same dimension. The process and results are shown in figure 5.



```
print(b.shape)
print(tficfVectorsTrain.shape)
print(np.linalg.norm(tficfVectorsTrain-b,ord = 'fro'))
```

```
(4732, 16600)
(4732, 16600)
62.38033050080974
```

```
b=nmf.inverse_transform(nmfVectorsTrain)
print(b.shape)
print(np.linalg.norm(tficfVectorsTrain-b,ord = 'fro'))
```

```
(4732, 16600)
62.77386213055328
```

Figure 5 Comparison between LSI and NMF

Therefore, the values are 62.38 for LSI and 62.77 for NMF and the latter one is larger. This indicates that LSI get better performance than NMF. This is because in LSI we do SVD, $X = U\Sigma V^T$, which is strictly equivalent. However, in NMF X is just approximate to WH, which decreases the precision. Though NMF is a little worse than LSI, this approximation saves much time we need in the process of reduction of dimension that NMF works faster than LSI.

Question 4 SVM

Here we use SVM (Support Vector Machine) to do binary classification. SVM constructs margins that can separate data points in different classes. In this project, we choose to use linear SVM, which is effective and efficient both. There are two kinds of SVM classified by the value of parameter γ , hard margin SVM (γ>>1) and soft margin SVM (γ<<1) . This parameter indicates the tradeoff between minimizing regulation, equivalent to maximizing margin between two different classes, and minimizing loss function on training data. Hard margin SVM strictly separate data but it is likely to cause overfitting. Soft margin SVM allows tolerance in misclassification. The results for hard margin SVM are shown in table 3 and figure 6. Table 4 and Figure 7 show the results for soft margin SVM.

Table 3 Hard Margin SVM Results ($\gamma = 1000$)

Parameters	Accuracy	Precision	Recall	F1_Score	Confusion Matrix

min_df = 3, LSI	0.927619	0.928953	0.927365	0.930104	[[1405 155] [73 1517]]

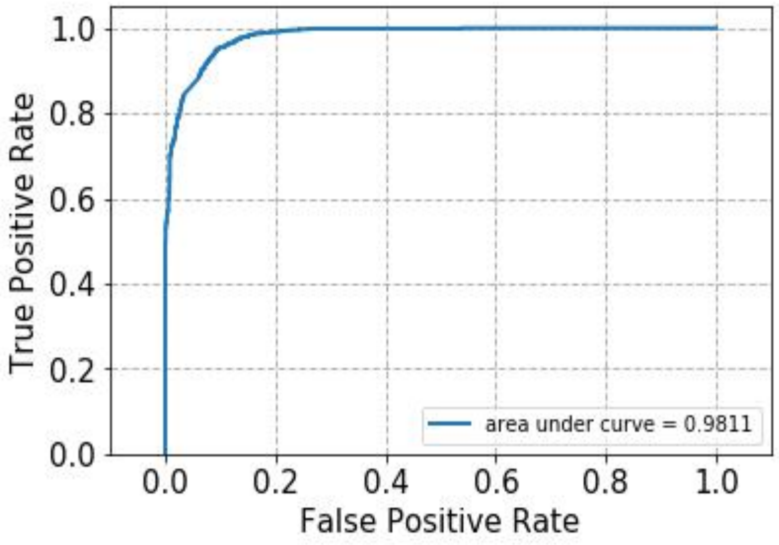


Figure 6 ROC Plot of Hard Margin SVM

Table 4 Soft Margin SVM Results $\gamma = 0.0001$

Parameters	Accuracy	Precision	Recall	F1_Score	Confusion Matrix
min_df = 3, LSI	0.504762	0.252381	0.500000	0.670886	[[0 1560] [0 1590]]

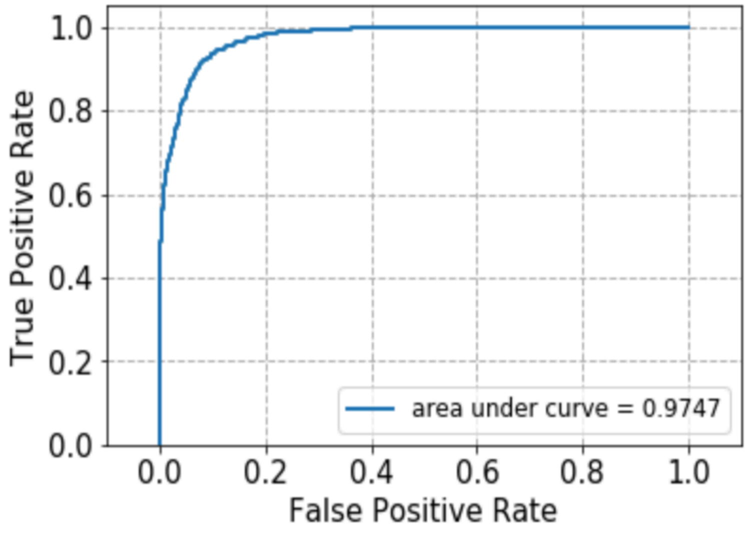


Figure 6 ROC Plot of Soft Margin SVM

From these two tables, we can conclude that hard margin SVM shows better performance over soft margin SVM. For Soft Margin SVM, true positive and false negative of the confusion matrix are 0. The precision, recall, and accuracy are far lower than what we expected to be a good classifier. It is due to the tolerance we allowed by setting parameter γ to 0.0001, which may lead to underfitting. However, the ROC curve of soft margin SVM looks good. The value of AUC is larger than 0.5 and is close to 1.0 indicating it is good, which is conflict to other metrices.

By using 5-fold cross validation with changing hyperparameter $\gamma = 10^k$ from k = -3 to 3 with step size of 1, the best parameter is k = 1.
The cross-validation scores for SVM with different k value are listed below:

Table 5 Cross-validation scores for SVM

k value	-3	-2	-1	0	1	2	3
score	0.33156	0.33586	0.90025	0. 92394	0. 92802	0. 92782	0. 92782

From the above chart, we can see that the SVM goes through underfitting to overfitting with increasing γ. SVM perform the best when k =1 ($\gamma = 10^1$). The performance of SVM with $\gamma = 10^1$ and its ROC curve are listed below:

Table 6 SVM Results $\gamma = 10$

Parameters	Accuracy	Precision	Recall	F1_Score	Confusion Matrix
min_df = 3, LSI	0. 924444	0. 926370	0. 924135	0. 927439	[[1391 169] [69 1521]]

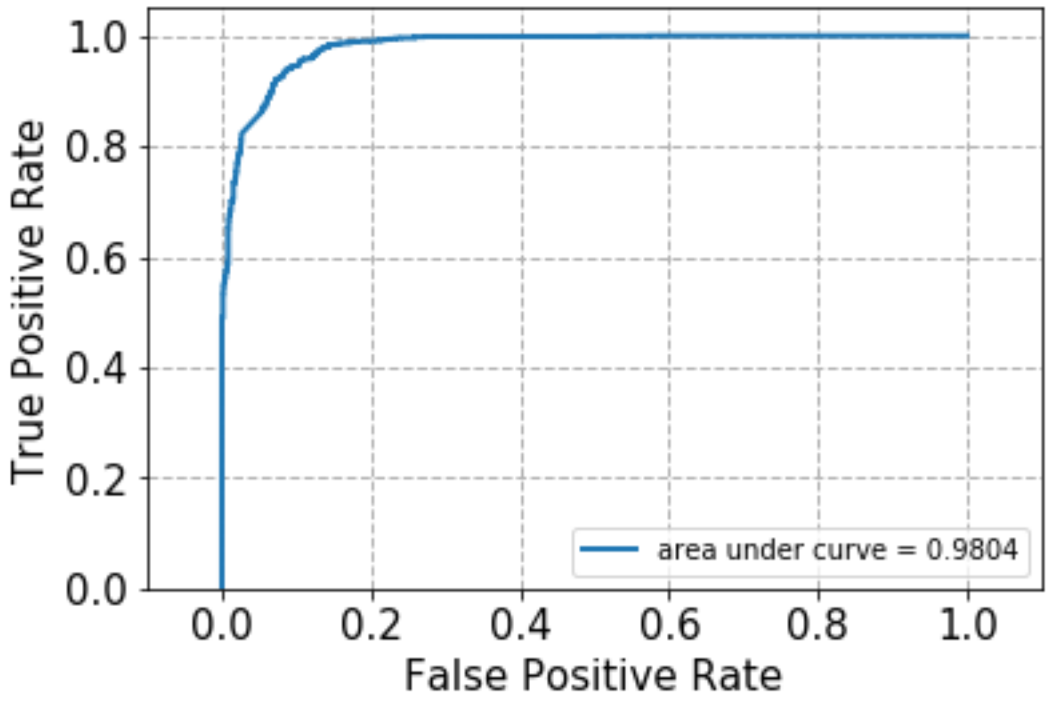


Figure 7 ROC Plot of SVM $\gamma = 10$

Question 5 Logistic Classifier

Part1: Logistic Classifier without Regularization

Here we use Logistic Regression (LR) to do binary classification. We use sklearn.linear_model.LogisiticRegression classifier to perform logistic regression. Since this classifier has an inverse penalty parameter, C, that will regularize the classification. In order to perform LR without regularization, we set C to a large number 10^6 to minimize the effect of regularization.

Table 7 LR Classification Results without Regularization

Parameters	Accuracy	Precision	Recall	F1_Score	Confusion Matrix
min_df = 3, LSI	0. 926667	0. 927386	0. 926482	0. 928682	[[1415 145] [86 1504]]

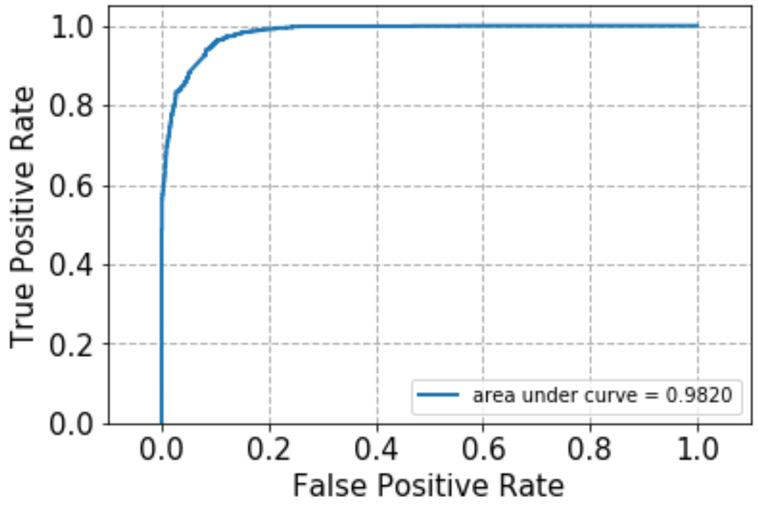


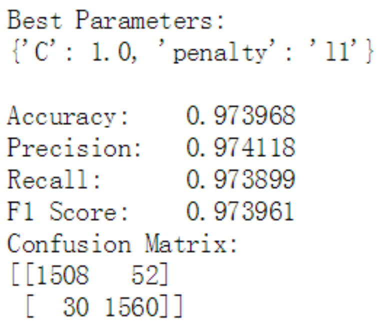
Figure 8 ROC Curve of LR Classifier without regularization

Part 2: Logistic Classifier with Regularization

When using logistic classifier with regularization, we use 5-fold cross-validation on the dimension-reduced-by-LSI training data (min_df = 3), trying to find the best regularization strength C in the set of C = [0.001 0.01 0.1 1 10 100 1000] for L1 regularization and L2 regularization.

We use the grid search for L1 regularization and L2 regularization respectively in order to find the best regularization strength.

When min_df = 3, penalty = 'L1':



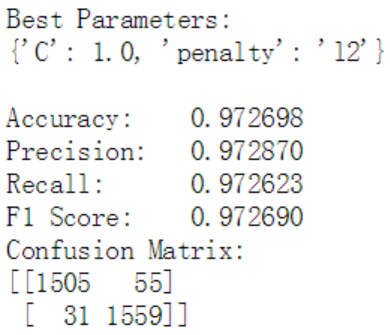
```
Best Parameters:
{'C': 1.0, 'penalty': 'l1'}

Accuracy:    0.973968
Precision:   0.974118
Recall:      0.973899
F1 Score:    0.973961
Confusion Matrix:
[[1508   52]
 [  30 1560]]
```

Figure 9 Result of LR When min_df = 3, penalty = 'L1'

As we can see, when using L1 regularization, the best regularization strength is 1.

When min_df = 3, penalty = 'L2':



```
Best Parameters:
{'C': 1.0, 'penalty': 'l2'}

Accuracy:    0.972698
Precision:   0.972870
Recall:      0.972623
F1 Score:    0.972690
Confusion Matrix:
[[1505   55]
 [  31 1559]]
```

Figure 10 Result of LR When min_df = 3, penalty = 'L2'

As we can see, when using L2 regularization, the best regularization strength is 1.
In comparison with logistic regression without regularization, we can derive the table as below:

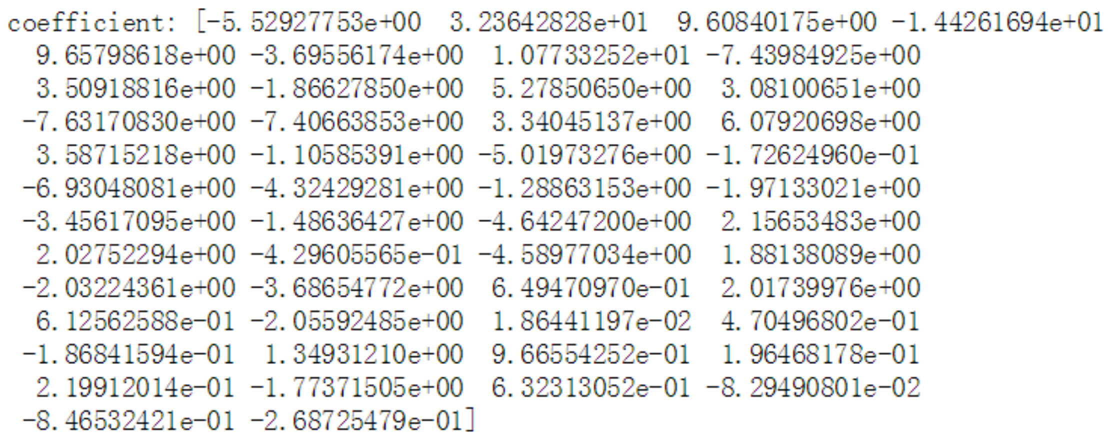
Table 8 Performance of Three Logistic Classifiers

Logistic Classifier	Accuracy	Precision	Recall	F1 Score
Without Regularization	0.977143	0.977175	0.977117	0.977140
L1 regularization	0.973968	0.974118	0.973899	0.973961
L2 regularization	0.972698	0.972870	0.972623	0.972690

Theoretically, a regression model that uses L1 regularization technique is called Lasso Regression and model which uses L2 is called Ridge Regression. Lasso regression (Least Absolute Shrinkage and Selection Operator) adds "absolute value of magnitude" of coefficient as penalty term to the loss function, whereas Ridge regression adds "squared magnitude" of coefficient as penalty term to the loss function. The main difference between the two techniques is that Lasso regression shrinks the less important feature's coefficient to zero thus, removing some feature altogether. So, this works well for feature selection in case we have a huge number of features, and both of the techniques can fix the problem of overfitting.

In order to compare the difference of coefficients among logistic regression without regularization, with L1 regularization, and with L2 regularization, we derive the coefficients of the three scenarios as follows:

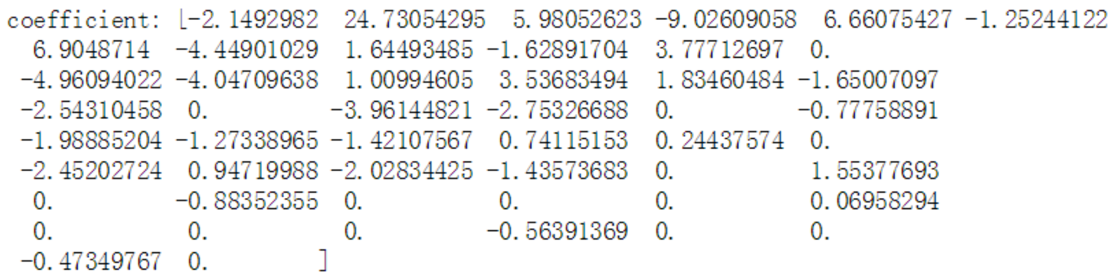
Coefficients without Regularization (C = 10^8):



```
coefficient: [-5.52927753e+00  3.23642828e+01  9.60840175e+00 -1.44261694e+01
  9.65798618e+00 -3.69556174e+00  1.07733252e+01 -7.43984925e+00
  3.50918816e+00 -1.86627850e+00  5.27850650e+00  3.08100651e+00
 -7.63170830e+00 -7.40663853e+00  3.34045137e+00  6.07920698e+00
  3.58715218e+00 -1.10585391e+00 -5.01973276e+00 -1.72624960e-01
 -6.93048081e+00 -4.32429281e+00 -1.28863153e+00 -1.97133021e+00
 -3.45617095e+00 -1.48636427e+00 -4.64247200e+00  2.15653483e+00
  2.02752294e+00 -4.29605565e-01 -4.58977034e+00  1.88138089e+00
 -2.03224361e+00 -3.68654772e+00  6.49470970e-01  2.01739976e+00
  6.12562588e-01 -2.05592485e+00  1.86441197e-02  4.70496802e-01
 -1.86841594e-01  1.34931210e+00  9.66554252e-01  1.96468178e-01
  2.19912014e-01 -1.77371505e+00  6.32313052e-01 -8.29490801e-02
 -8.46532421e-01 -2.68725479e-01]
```

Figure 11 Coefficients without Regularization

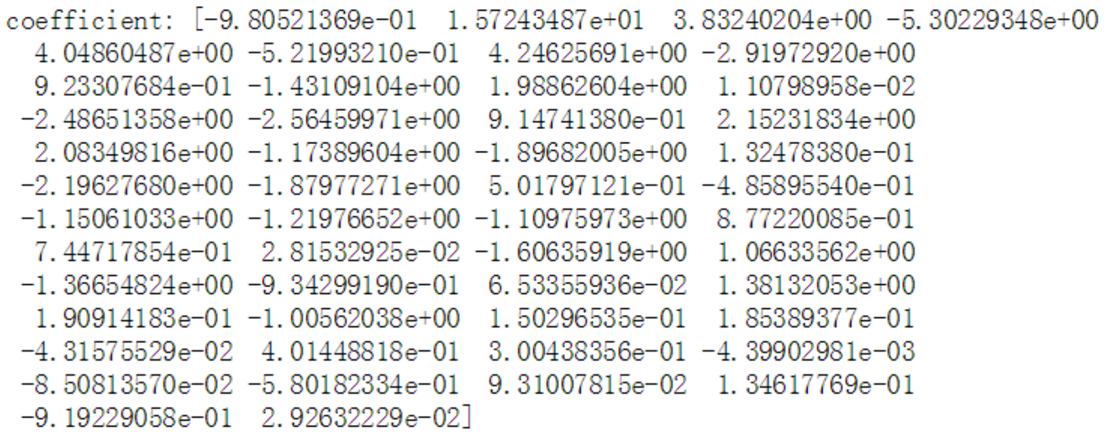
Coefficients with L1 Regularization (C = 1.0):



```
coefficient: [-2.1492982  24.73054295  5.98052623 -9.02609058  6.66075427 -1.25244122
  6.9048714  -4.44901029  1.64493485 -1.62891704  3.77712697  0.
 -4.96094022 -4.04709638  1.00994605  3.53683494  1.83460484 -1.65007097
 -2.54310458  0.          -3.96144821 -2.75326688  0.          -0.77758891
 -1.98885204 -1.27338965 -1.42107567  0.74115153  0.24437574  0.
 -2.45202724  0.94719988 -2.02834425 -1.43573683  0.           1.55377693
  0.          -0.88352355  0.           0.           0.           0.06958294
  0.           0.           0.          -0.56391369  0.           0.
 -0.47349767  0.         ]
```

Figure 12 Coefficients with L1 Regularization (C = 1.0)

Coefficients with L2 Regularization (C = 1.0):



```
coefficient: [-9.80521369e-01  1.57243487e+01  3.83240204e+00 -5.30229348e+00
  4.04860487e+00 -5.21993210e-01  4.24625691e+00 -2.91972920e+00
  9.23307684e-01 -1.43109104e+00  1.98862604e+00  1.10798958e-02
 -2.48651358e+00 -2.56459971e+00  9.14741380e-01  2.15231834e+00
  2.08349816e+00 -1.17389604e+00 -1.89682005e+00  1.32478380e-01
 -2.19627680e+00 -1.87977271e+00  5.01797121e-01 -4.85895540e-01
 -1.15061033e+00 -1.21976652e+00 -1.10975973e+00  8.77220085e-01
  7.44717854e-01  2.81532925e-02 -1.60635919e+00  1.06633562e+00
 -1.36654824e+00 -9.34299190e-01  6.53355936e-02  1.38132053e+00
  1.90914183e-01 -1.00562038e+00  1.50296535e-01  1.85389377e-01
 -4.31575529e-02  4.01448818e-01  3.00438356e-01 -4.39902981e-03
 -8.50813570e-02 -5.80182334e-01  9.31007815e-02  1.34617769e-01
 -9.19229058e-01  2.92632229e-02]
```

Figure 13 Coefficients with L2 Regularization (C = 1.0)

As we can see above, the coefficients of logistic regression without regularization have larger absolute value than those of logistic regression with L2 regularization, whereas some of the coefficients of logistic regression with L1 regularization are zeros, or we can say the coefficients

of logistic regression with L1 regularization are “sparser”, which reduces the number of the features.
Both logistic regression and linear SVM are trying to classify data points using a linear decision boundary. Although logistic regression and SVM with linear Kernel generally perform comparably in practice, they still have different aspects. The objective of the support vector machine algorithm is to find the hyperplane that has the maximum margin in an N-dimensional space (N—the number of features) that distinctly classifies the data points, or in other words, it tries to minimize the hinge loss. However, in the logistic regression, we try to maximize the likelihood that a random data point gets classified correctly, and we use the sigmoid function to compress the probability into the range of [0, 1], and when the value of sigmoid function is greater than some threshold, the input will be labeled as 1, otherwise it will be labeled as 0, so in the logistic regression process, we aim to minimize the logistic loss.

Since the logistic loss diverges faster than hinge loss, in general, it will be more sensitive to outliers. Furthermore, logistic loss does not go to zero even if the point is classified sufficiently confidently, this might lead to minor degradation in accuracy.

Question 6 Naïve Bayes Gaussian

In this part, we applied Naïve Bayes classifier to perform the binary classification. Naïve Bayes’s principle is based on the Maximum A Posteriori (MAP) assumption that features are independent from each other. The algorithm estimates the maximum likelihood probability based on a given features set. And for this part, we applied GaussianNB model and the results are shown as the following.

Table 9 Performance of Three Logistic Classifiers

Parameters	Accuracy	Precision	Recall	F1_Score	Confusion Matrix
min_df = 3, LSI	0. 906984	0. 927044	0. 892792	0. 909595	[[1383 177] [116 1474]]

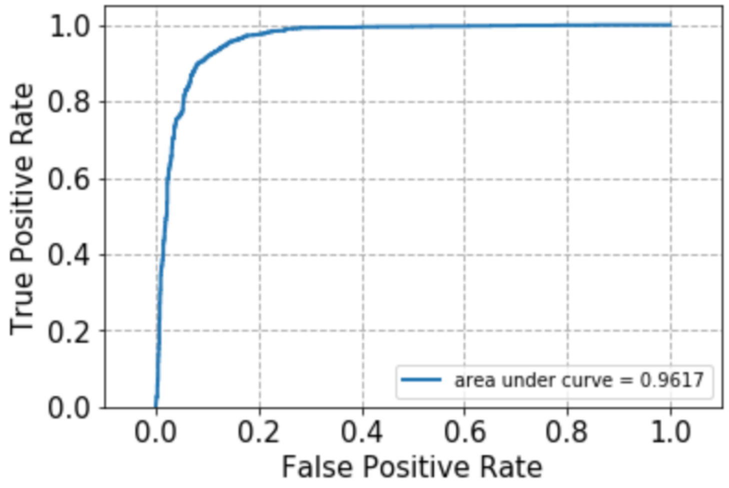
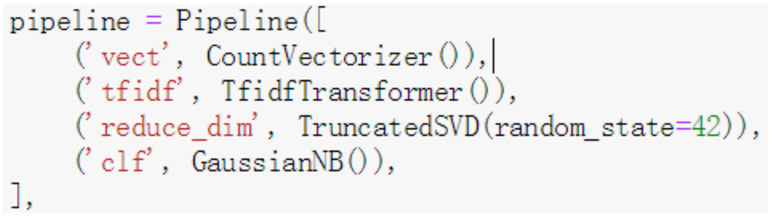


Figure 14 ROC Plot of Naïve Bayes Gaussian

Question 7 Pipeline and Grid Search

There are standard workflows in applied machine learning. Standard because they overcome common problems like data leakage in the test harness. Python scikit-learn provides a Pipeline utility to help automate machine learning workflows. Pipelines work by allowing for a linear sequence of data transforms to be chained together culminating in a modeling process that can be evaluated.

We created a pipeline that can perform feature extraction, dimensionality reduction and classification.



```
pipeline = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('reduce_dim', TruncatedSVD(random_state=42)),
    ('clf', GaussianNB()),
],
```

Figure 15 Pipeline for Grid Search

Furthermore, we implemented grid search on the constructed pipeline to find the best set of parameters that achieves greatest test accuracy.

When loading the data with chopping off 'headers' and 'footers', we performed the grid search on the parameter:
min_df = [3, 5],
reduce_dim = [TruncatedSVD(n_components=50), NMF(n_components=50)],
clf = [
LinearSVC(), C = [0.1, 1, 10]
GaussianNB(),
LogisticRegression(), C = [0.1, 1, 10], penalty = ['l1', 'l2']
]

Then we derive the best parameters:
min_df = 3,
reduce_dim = TruncatedSVD(n_components=50)
clf = LogisticRegression() with C = 10, penalty = 'l1'
The accuracy is 0.9653423499577346.

When loading the data without chopping off 'headers' and 'footers', we performed the grid search on the parameter:
min_df = [3, 5],
reduce_dim = [TruncatedSVD(n_components=50), NMF(n_components=50)],
clf = [
LinearSVC(), C = [0.1, 1, 10]
GaussianNB(),
LogisticRegression(), C = [0.1, 1, 10], penalty = ['l1', 'l2']
]

Then we derive the best parameters:
min_df = 3,
reduce_dim = TruncatedSVD(n_components=50)
clf = LinearSVC() with C = 10.
The accuracy is 0.9723161453930684.

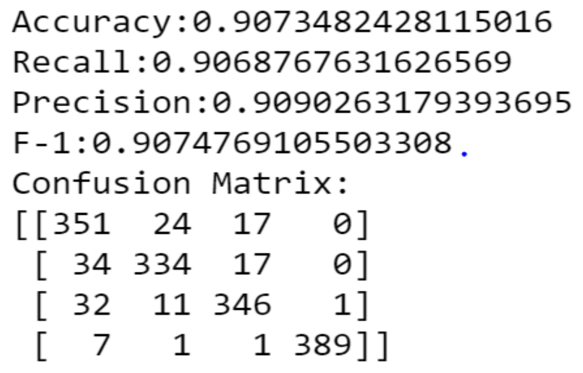
In comparison with chopping off the 'headers' and 'footers' when loading the data, we can see that we can achieve better accuracy when we do not chop of the 'headers' and 'footers' and the best parameters are:
min_df = 3,
reduce_dim = TruncatedSVD(n_components=50)
clf = LinearSVC() with C = 10.

Question 8 Multiclass Classification

In previous parts, we constructed and evaluated classifies for binary classification. And for this part, we applied the Naïve Bayes algorithm and SVM for multiclass classifications.

Question 8 part1: Naïve Bayes for Multiclass

We applied Naïve Bayes algorithm for binary classification, and the result is satisfying. And Naïve Bayes algorithm can get maximum likelihood computation for given data, regardless of the classes number of given data, so we can directly use the Naïve Bayes algorithm to build a multiclass classifier. In the previous binary classification, we used the GaussinNB model and it worked well for binary classification but it didn't work as well for multiclass so we chose the MultinomialNB model. MultinomialNB implements the naive Bayes algorithm for multinomially distributed data which is suitable for multiclass classifications. And the results are shown as the following figure.



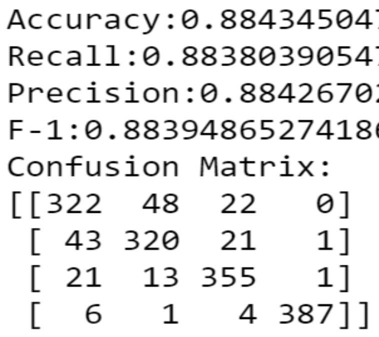
```
Accuracy:0.9073482428115016
Recall:0.9068767631626569
Precision:0.9090263179393695
F-1:0.9074769105503308.
Confusion Matrix:
[[351  24  17   0]
 [ 34 334  17   0]
 [ 32  11 346   1]
 [  7   1   1 389]]
```

Figure 16 Results of MultinomialNB Model for Multiclass Classification

From the above figure, we can see that for multiclass, the MultinomialNB model worked greatly since the accuracy and other parameters are all above 0.9.

Question 8 part2: SVM – One Vs One

The SVM cannot perform the multiclass classification inherently, which is different with Naïve Bayes algorithm. The idea for SVM is to maximum the margin so the basic SVM cannot handle multiclass classification. [3] Firstly, we applied one vs one of SVM, which can be considered like that we applied a 'binary classifier' for each pair of classes and derive decision boundaries of two classes. Therefore, if the problem has N classes, we OVO will be composed N(N-1)/2 times. And the results are shown as following.



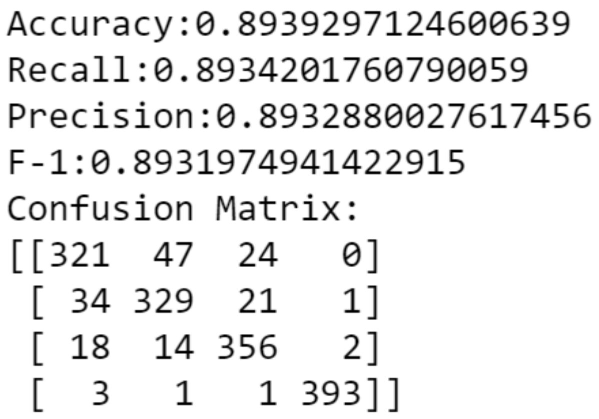
```
Accuracy:0.8843450479233227
Recall:0.8838039054747597
Precision:0.8842670238616814
F-1:0.8839486527418602
Confusion Matrix:
[[322  48  22   0]
 [ 43 320  21   1]
 [ 21  13 355   1]
 [  6   1   4 387]]
```

Figure 17 Results of SVM-One Vs One for Multiclass Classification

As you can see from the figure, the accuracy is nearly0 .9 and SVM One VS One works well for multiclass classification. We assigned the penalty parameter C to be 10, which is better than C=1.0 and C larger than 10 doesn't improve the results.

Question 8 part3: SVM – One Vs Rest

Another natural way if One Vs Rest (OVR). OVR computes a classifier for each class, and it assign a label as positive and left as negative. And in this way, the unbalanced number of each class should be handled. In this way, we can reduce the number of classifiers to N which is much smaller than One Vs One. And the base classifiers are also required to give real-value confidence scores for their decisions. The results are shown as the following,



```
Accuracy:0.8939297124600639
Recall:0.8934201760790059
Precision:0.8932880027617456
F-1:0.8931974941422915
Confusion Matrix:
[[321  47  24   0]
 [ 34 329  21   1]
 [ 18  14 356   2]
 [  3   1   1 393]]
```

Figure 18 Results of SVM – One Vs Rest for Multiclass Classification

The results are great as the accuracy and other parameters are close to 0.9.

Reference

[1] https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html
[2] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html#sklearn.feature_extraction.text.TfidfTransformer
[3] https://en.wikipedia.org/wiki/Multiclass_classification#Naive_Bayes