EE 219 Project 1 Classification Analysis on Textual Data

Zhilai Shen, Yufei Hu, Zheang Huai, and Tianyi Liu

January 30, 2019

1 Getting familiar with the dataset

Question 1:

After loading the 20 categories from trianing set, we are able to view the historgram which shows the distribution of number of training documents in these 20 categories. The graph is shown as following, and from the historgram, the number of training documents in each categories is almost the same.

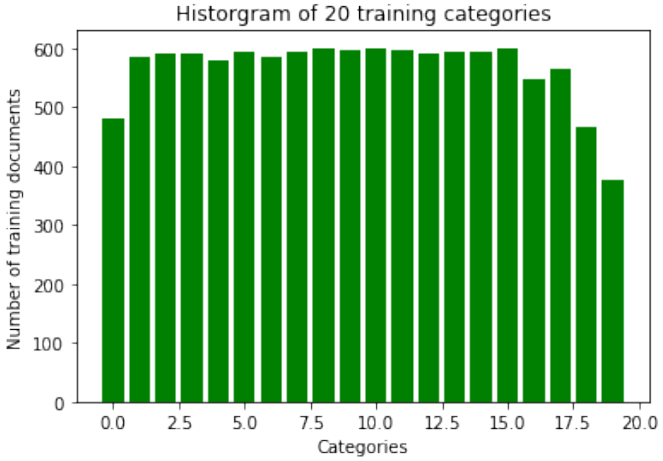

Figure 1: Historgram of 20 Categories

2 Binary Classification

2.1 Feature Extraction

Question 2:

Since we are only interested in 8 categories, while loading the data, "category" is a list containing these eight targeted categories. The stopwords are set to "english" as required, and one regular expression is used to get rid of numeric terms. Min_df is set to be 3. The lemmatization and pos_tag are also applied to combined term words with same root word.

After the above process, the shape of the TF-IDF matrices of the train subsets is $(4732, 16292)$ and that of rge test subsets is $(3150, 16292)$.

2.2 Dimensionality Reduction

Question 3:

Once we have the TF-IDF matrix, dimension reduction needs to be performed on the TF-IDF matrices by 1) LSI and 2) NMF. *TruncatedSVD()* and *NMF* modules in *scikit learn* are used and number of components is set to be 50. Data is then loaded for further use.

The square of Frobenius distance between original training data and reconstructed training data using LSI and NMF are 4103.50 and 4143.26 respectively. LSI seems to be slightly better than NMF for performing dimensionality reduction in this case.

To explain the possible reason behind this observation, differences between LSI and NMF have to be analyzed. NMF split a matrix R into the product of two matrices W and H. Here R is usually very sparse. In such cases NMF works better as the missing-values assumption is inbuilt to the algorithm. In case of LSI, it does not assume anything about sparsity of the original data. And LSI usually works better when the input matrix is not too sparse. LSI results are more deterministic compared to that of NMF. In our case, the possible reason for the observation might be due to the TF-IDF matrix is not too sparse.

2.3 Classification Algorithms

2.3.1 SVM

Question 4:

Table 1: Performances for hard margin SVM (LSI, γ=1000)

Accuracy	0.9714
Recall	0.9792
Precision	0.9647
F1 Score	0.9719

A hard margin and a soft margin SVM classifiers using linear kernel are trained based on training data from LSI. Their ROC curve are shown in 2 and 3 respectively. Other performances can be found in 1, 2, 3, 4.

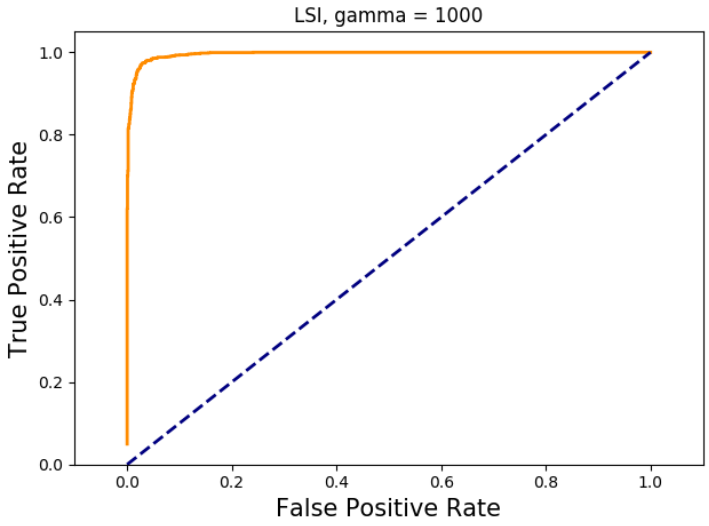

Figure 2: ROC for hard margin SVM (LSI, γ=1000)

Table 2: Performances for hard margin SVM (LSI, γ=1000)

	Computer tech (predict)	Recreation (predict)
Computer tech (true)	1503	57
Recreation (true)	33	1557

Based on the results, we have observed that hard margin classifier performs much better than soft margin classifier in terms of all metrics. It appears that all instances in the test set are predicted to be recreation-related when using the soft margin classifier. Typically, a hard margin SVM works better when the data is more linearly separable. Also, a too high penalty parameter C could lead to overfitting problem while a too low C could cause underfitting problem. In our case, the instances are more difficult to fit considering their scales and dimensions (50 is still quite large to a classification problem), therefore we need an SVM classifier with a higher C to deal with such dataset. This is the reason why hard margin SVM classifier outperforms soft margin SVM classifier.

Soft margin SVM classifier may have underfitted based on its confusion matrix listed in 4. Therefore, its ROC curve looks not so good which is also shown in 3. There is a sharper turning at the top-left corner in 3 comparing to 2, the reason is because soft margin SVM classifier either has a high true positive rate with a low false positive rate or a low true positive rate with a high false positive rate. There does not exist a sweetspot that makes both true positive rate and false positive rate reach a good balance. Thus, no conflict is found between ROC and other metrics for the soft margin SVM classifier.

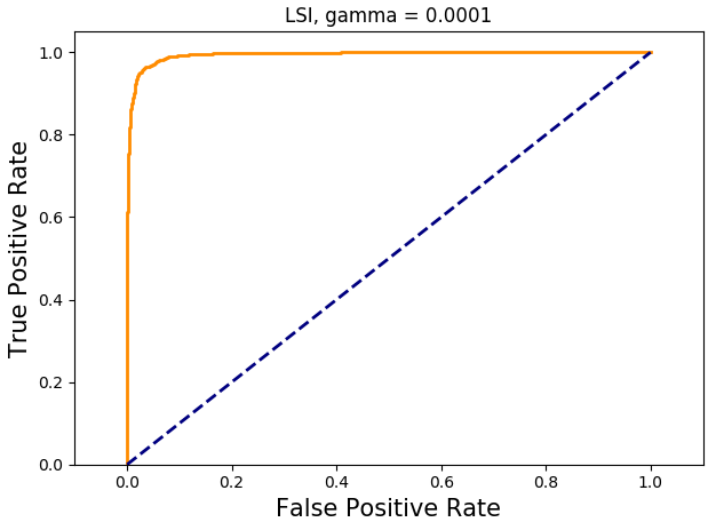

Figure 3: ROC for soft margin SVM (LSI, γ=0.0001)

Table 3: Performances for soft margin SVM (LSI, γ=0.0001)

Accuracy	0.5048
Recall	1.0000
Precision	0.5048
F1 Score	0.6709

After applying 5-fold cross validation, we found that the best value of the penalty parameter γ for linear SVM classifier is 10. We then perform same evaluation. ROC curve, confusion matrix and other metrics can be found in 4, 5 and 6.

2.3.2 Logistic Regression

Question 5:

We trained a logistic classifier without regularization. The ROC curve is shown in Figure 5. The other classification measure results are presented in Table 7.

Now by applying 5-fold cross-validation on the training data just like previously discussed, we found that the best regularization strength is 10 for L1 regularization and 100 for l2 regularization. The ROC curve of those logistic classifiers with regularization are shown in Figure 6 and Figure 7. The other classification measure results are presented in Table 9 and Table 10.

By comparing the performance results of those three logistic classifiers: without regu-

Table 4: Performances for soft margin SVM (LSI, γ=0.0001)

	Computer tech (predict)	Recreation (predict)
Computer tech (true)	0	1560
Recreation (true)	0	1590

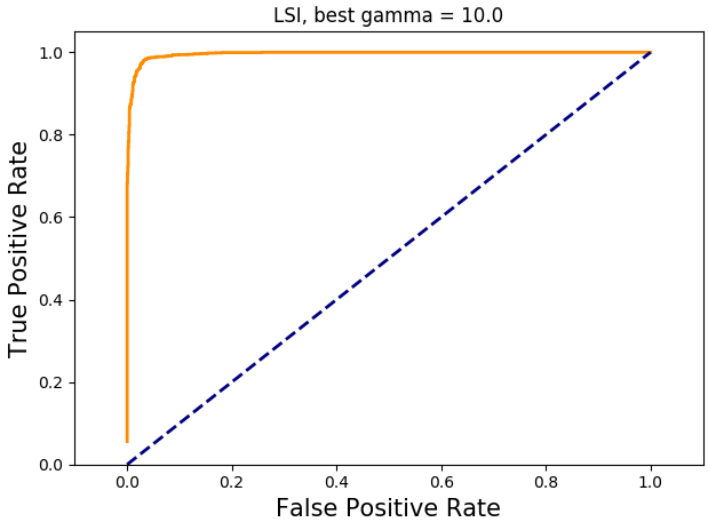

Figure 4: ROC for best SVM (LSI, γ=10)

larization, with L1 regularization and L2 regularization, we can conclude that the logistic classifier with L2 regularization does the best job.

Like Owen Zheng said, if you are using regression without regularization, you have to be very special. Technically speaking, regularization would constrain learning a way more complex model, so as to avoid the risk of overfitting. In term of coefficient, regularization is kind of a form of regression that would shrink the coefficient estimates towards zero. That's why adding regularization would facilitate generalization ability of the classifier and improve the performance when applying test data. For example, if there is noise in the training data, then the estimated coefficients won't generalize well to the future test data. This is where regularization comes in and regularizes these learned estimates towards zero. However it's also a trade off to choose a great and appropriate regularization strength. Increasing regularization parameter results in less overfitting bus also greater bias, which still would decrease the performance of the classifier.

When it comes to choosing a type of regularization, the differences of L1-norm and L2-norm as a loss function can be promptly summarized as Table 8. Intuitively, the key

Table 5: Performances for best SVM (LSI, γ=10)

Accuracy	0.9740
Recall	0.9805
Precision	0.9683
F1 Score	0.9744

Table 6: Performances for best SVM (LSI, γ=10)

	Computer tech (predict)	Recreation (predict)
Computer tech (true)	1509	51
Recreation (true)	31	1559

difference between these two regularization methods is that L1 can be used for noisy data, which is very appealing and on the other hand L2 has a closed form solution in an optimization setting.

Even though logistic regression and linear SVM both are using a linear decision boundary, the concept or motivation behind the algorithm is totally different. Logistic regression focuses on maximizing the probability of the data. The farther the data lies from the separating hyperplane on the correct side, the happier it is. On the other hand, an SVM just tries to find the hyperplane that maximizes the distance of the closet points to the margin. If a point is not a support vector, it doesn't really matter. Therefore, depending on the problem and data, their performance would differ.

2.3.3 Naive Bayes

Question 6:

This time, we trained a naive Bayes classifier. The ROC curve is shown in Figure 8. The other classification measure results are presented in Table 11.

2.3.4 Grid Search of Parameters

Question 7:

We do grid search with 5-fold cross-validation for different classifiers with different parameters and compare their test accuracy. The results of removing "headers" and "footers" are shown in Table 12 and the results of not removing "headers" and "footers" are shown in Table 13. Due to the width of the table, it can't be shown completely. The complete table is in the excel document named "q7".

From the two tables, we can get the following analysis: Min_df does not have much influence on mean_test_score, because the mean_test_scores when we choose Min_df=3

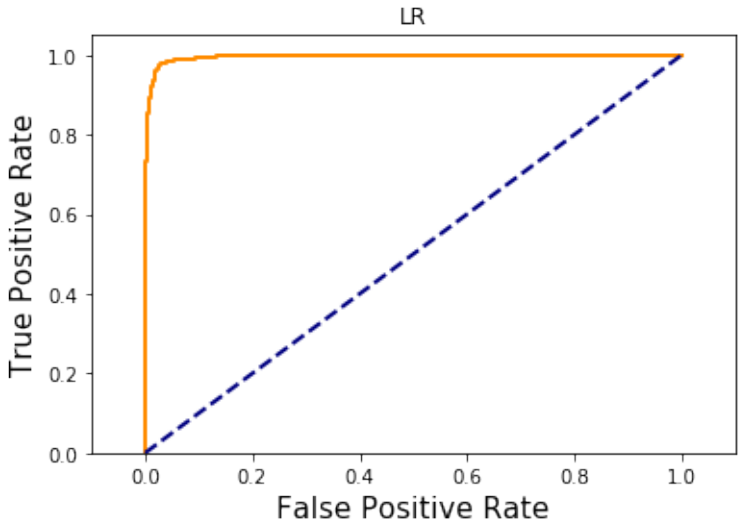

Figure 5: ROC curve of logistic classifier without regularization

Table 7: Classification quanlity of logistic regression without regularization

Confusion Matrix	[1500 60 ; 32 1558]
Accuracy	0.9708
Recall	0.9799
Precision	0.9629
F1 Score	0.9713

or Min_df=5 are very close. Lemmatization has much influence on mean_test_score. Mean_test_score improves significantly when we use lemmatization than not. In most cases, mean_test_score is higher when we use LSI to reduce dimensionality than using NMF. But if we choose GaussianNB classifier and use lemmatization, NMF is better than LSI. As for different classifiers, the performance of GaussianNB is the worst and the performance of the other three classifiers is very close. And the performance improves a little when we do not remove "headers" and "footers" than we do.

The best combination is this: The classifier is SVM or Logistic Regression with L1 or L2 regularization(because their performances are really close to each other). Do not remove "headers" and "footers". Use lemmatization. Use LSI to do dimensionality reduction. And min_df=3(But in some cases min_df=5 has better performance).

Table 8: Comparison between L1 and L2 regularization

	L1 regularization	L2 regularization
Robustness	robust	not very robust
Stability	unstable solution	stable solution
Uniqueness	possibly multiple solutions	always one solution

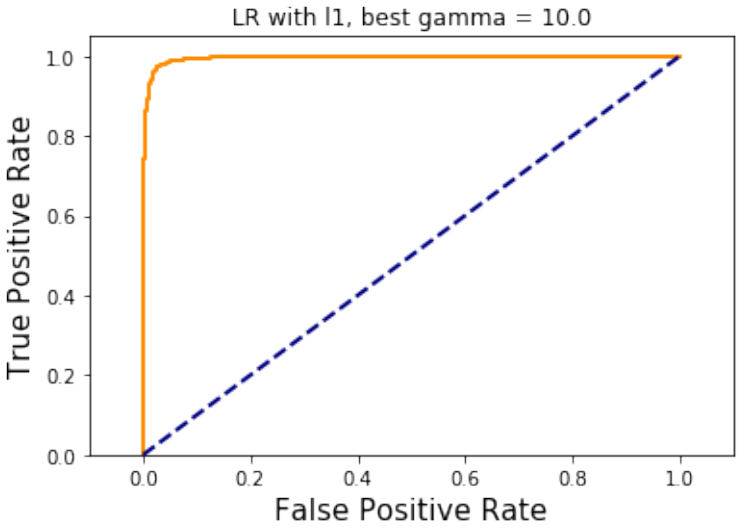

Figure 6: ROC curve of logistic classifier with regularization L1

3 Multiclass Classification

Question 8:

The results of different multiclass classification are shown in Table 14,Table 15,Table 16,Table 17,Table 18,Table 19.

From the results, we can get the following conclusions: For multiclass classification task, SVM has better performance than Naïve Bayes. And OneVsOne SVM and OneVsRest SVM have similar classification performances. As for dimension reduction methods in this question, LSI has better performance than NMF when we use SVM, but has worse performance than NMF when we use Naïve Bayes.

Table 9: Classification quanlity of logistic regression with regularization L1

Confusion Matrix	[1501 59 ; 33 1557]
Accuracy	0.9708
Recall	0.9792
Precision	0.9635
F1 Score	0.9713

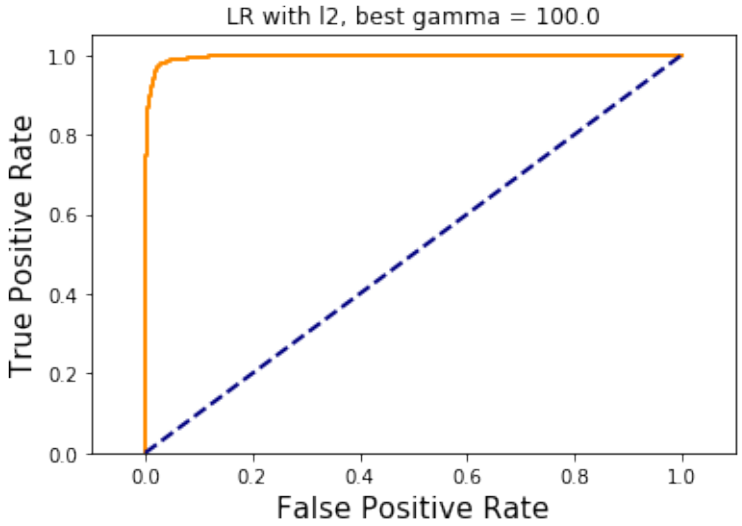

Figure 7: ROC curve of logistic classifier with regularization L2

Table 10: Classification quanlity of logistic regression with regularization L2

Confusion Matrix	[1505 66 ; 30 1560]
Accuracy	0.9727
Recall	0.9911
Precision	0.9653
F1 Score	0.9732

Table 11: Classification quanlity of naive Bayes classifier

Confusion Matrix	[1334 226 ; 56 1534]
Accuracy	0.9105
Recall	0.9648
Precision	0.8716
F1 Score	0.9158

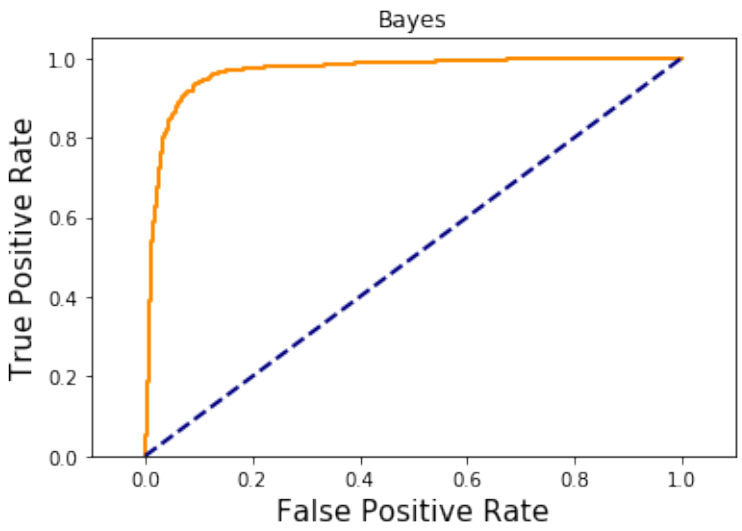

Figure 8: ROC curve of naive Bayes classifier

Table 12: Removing "headers" and "footers"

	param clf	param clf C	param reduce dim	param
0	LinearSVC	10	TruncatedSVD(algorithm='randomized', n compone...	<function stem rmv pu
1	LinearSVC	10	TruncatedSVD(algorithm='randomized', n compone...	<function stem rmv pu
2	LinearSVC	10	TruncatedSVD(algorithm='randomized', n compone...	<function dumb ste
3	LinearSVC	10	TruncatedSVD(algorithm='randomized', n compone...	<function dumb ste
4	LinearSVC	10	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function stem rmv pu
5	LinearSVC	10	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function stem rmv pu
6	LinearSVC	10	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function dumb ste
7	LinearSVC	10	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function dumb ste
8	GaussianNB	NaN	TruncatedSVD(algorithm='randomized', n compone...	<function stem rmv pu
9	GaussianNB	NaN	TruncatedSVD(algorithm='randomized', n compone...	<function stem rmv pu
10	GaussianNB	NaN	TruncatedSVD(algorithm='randomized', n compone...	<function dumb ste
11	GaussianNB	NaN	TruncatedSVD(algorithm='randomized', n compone...	<function dumb ste
12	GaussianNB	NaN	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function stem rmv pu
13	GaussianNB	NaN	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function stem rmv pu
14	GaussianNB	NaN	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function dumb ste
15	GaussianNB	NaN	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function dumb ste
16	LogisticRegression	10	TruncatedSVD(algorithm='randomized', n compone...	<function stem rmv pu
17	LogisticRegression	10	TruncatedSVD(algorithm='randomized', n compone...	<function stem rmv pu
18	LogisticRegression	10	TruncatedSVD(algorithm='randomized', n compone...	<function dumb ste
19	LogisticRegression	10	TruncatedSVD(algorithm='randomized', n compone...	<function dumb ste
20	LogisticRegression	10	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function stem rmv pu
21	LogisticRegression	10	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function stem rmv pu
22	LogisticRegression	10	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function dumb ste
23	LogisticRegression	10	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function dumb ste
24	LogisticRegression	100	TruncatedSVD(algorithm='randomized', n compone...	<function stem rmv pu
25	LogisticRegression	100	TruncatedSVD(algorithm='randomized', n compone...	<function stem rmv pu
26	LogisticRegression	100	TruncatedSVD(algorithm='randomized', n compone...	<function dumb ste
27	LogisticRegression	100	TruncatedSVD(algorithm='randomized', n compone...	<function dumb ste
28	LogisticRegression	100	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function stem rmv pu
29	LogisticRegression	100	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function stem rmv pu
30	LogisticRegression	100	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function dumb ste
31	LogisticRegression	100	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function dumb ste

Table 13: Not removing "headers" and "footers"

	param clf	param clf C	param reduce dim	param
0	LinearSVC	10	TruncatedSVD(algorithm='randomized', n compone...	<function stem rmv pu
1	LinearSVC	10	TruncatedSVD(algorithm='randomized', n compone...	<function stem rmv pu
2	LinearSVC	10	TruncatedSVD(algorithm='randomized', n compone...	<function dumb ste
3	LinearSVC	10	TruncatedSVD(algorithm='randomized', n compone...	<function dumb ste
4	LinearSVC	10	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function stem rmv pu
5	LinearSVC	10	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function stem rmv pu
6	LinearSVC	10	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function dumb ste
7	LinearSVC	10	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function dumb ste
8	GaussianNB	NaN	TruncatedSVD(algorithm='randomized', n compone...	<function stem rmv pu
9	GaussianNB	NaN	TruncatedSVD(algorithm='randomized', n compone...	<function stem rmv pu
10	GaussianNB	NaN	TruncatedSVD(algorithm='randomized', n compone...	<function dumb ste
11	GaussianNB	NaN	TruncatedSVD(algorithm='randomized', n compone...	<function dumb ste
12	GaussianNB	NaN	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function stem rmv pu
13	GaussianNB	NaN	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function stem rmv pu
14	GaussianNB	NaN	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function dumb ste
15	GaussianNB	NaN	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function dumb ste
16	LogisticRegression	10	TruncatedSVD(algorithm='randomized', n compone...	<function stem rmv pu
17	LogisticRegression	10	TruncatedSVD(algorithm='randomized', n compone...	<function stem rmv pu
18	LogisticRegression	10	TruncatedSVD(algorithm='randomized', n compone...	<function dumb ste
19	LogisticRegression	10	TruncatedSVD(algorithm='randomized', n compone...	<function dumb ste
20	LogisticRegression	10	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function stem rmv pu
21	LogisticRegression	10	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function stem rmv pu
22	LogisticRegression	10	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function dumb ste
23	LogisticRegression	10	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function dumb ste
24	LogisticRegression	100	TruncatedSVD(algorithm='randomized', n compone...	<function stem rmv pu
25	LogisticRegression	100	TruncatedSVD(algorithm='randomized', n compone...	<function stem rmv pu
26	LogisticRegression	100	TruncatedSVD(algorithm='randomized', n compone...	<function dumb ste
27	LogisticRegression	100	TruncatedSVD(algorithm='randomized', n compone...	<function dumb ste
28	LogisticRegression	100	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function stem rmv pu
29	LogisticRegression	100	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function stem rmv pu
30	LogisticRegression	100	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function dumb ste
31	LogisticRegression	100	NMF(alpha=0.0, beta loss='frobenius', init=Non...	<function dumb ste

Table 14: Use LSI to reduce dimension, GaussianNB classification

Confusion Matrix	[[226 42 122 2] [107 148 128 2] [43 47 298 2] [0 1 15 382]]
Accuracy	0.6735
Recall	0.6735
Precision	0.6858
F1 Score	0.6673

Table 15: Use LSI to reduce dimension, OneVsOneSVM classification

Confusion Matrix	[[330 40 22 0] [42 322 21 0] [26 20 343 1] [5 1 4 388]]
Accuracy	0.8837
Recall	0.8837
Precision	0.8848
F1 Score	0.8842

Table 16: Use LSI to reduce dimension, OneVsRestSVM classification

Confusion Matrix	[[317 47 26 2] [34 330 20 1] [17 21 348 4] [2 0 2 394]]
Accuracy	0.8875
Recall	0.8875
Precision	0.8874
F1 Score	0.8872

Table 17: Use NMF to reduce dimension, GaussianNB classification

Confusion Matrix	[[275 53 56 8] [116 221 44 4] [48 36 290 16] [1 1 5 391]]
Accuracy	0.7521
Recall	0.7521
Precision	0.7516
F1 Score	0.7494

Table 18: Use NMF to reduce dimension, OneVsOneSVM classification

Confusion Matrix	[[303 62 27 0] [70 291 23 1] [37 14 337 2] [4 1 4 389]]
Accuracy	0.8435
Recall	0.8435
Precision	0.8450
F1 Score	0.8440

Table 19: Use NMF to reduce dimension, OneVsRestSVM classification

Confusion Matrix	[[294 70 28 0] [66 293 22 4] [33 15 338 4] [3 2 2 391]]
Accuracy	0.8409
Recall	0.8409
Precision	0.8408
F1 Score	0.8409