ECE 219 Project 1: Classification Analysis on Textual Data

Chuang Yu (305629107), Hongyi Chen (705186099), Zichun Chai (505625207)

# Question 1:

In the first question of the project, we first divided 20,000 documents into 20 different groups. To see if the documents were distributed evenly, we plotted a histogram to visualize the distribution.

The data set was already balanced (especially for the categories that we would mainly work on), and therefore we did not need to balance by modifying the penalty function or down-sampling the majority class.

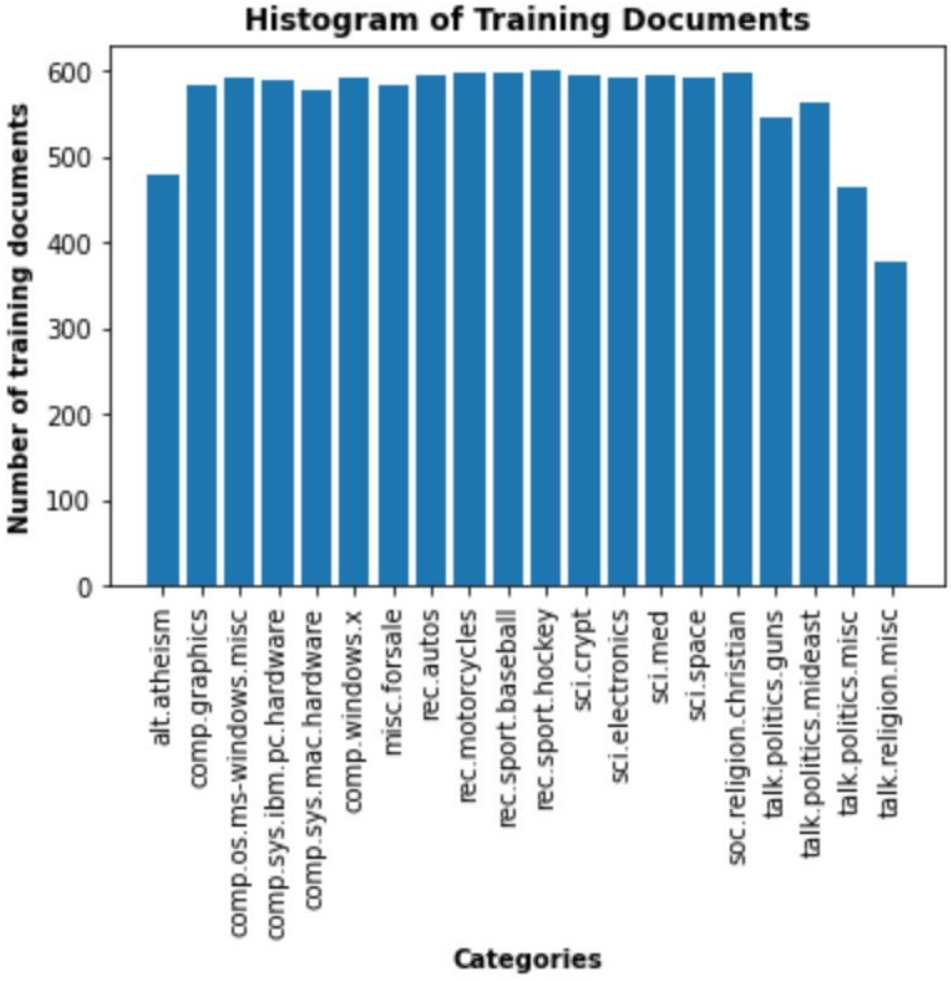



Figure 1: Histogram of the distributed documents

From the histogram given above, the documents were distributed into 20 groups evenly with minor deviations.

# Question 2:

In this question, we first had a given list of categories ['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'] with a random state of 42. We treated the documents as a set of words and chose our own stopwords. Document representation was very important in classifying a corpus of text since a good representation would not only retain enough information but also enable us to perform classification. We decided to use the "Bag of Words" representation in which a corpus of text can be summarized into a term-document matrix whose entries are some statistic of the terms.

With the instructions given by the project manual, we performed lemmatization with nltk.wordnet and pos_tag to extract what we wanted from the larger set. Lemmatization is a process of finding the root word with linguistics rules (with the use of regexes). Then we used the "english" stopwords and min_df = 3 in the CountVectorizer, and excluded all numerical values and symbols. The result of the train and test TF-IDF matrices were given below:

| | |
|---|---|
| Train | (4732, 16292) |
| Test | (3150, 16292) |

Table 1: Shape of the TF-IDF matrices of the train and test subsets

From the table we can see that there were 4732 documents in the train dataset and 3150 documents in the test dataset with 16292 words(features) extracted from both of them.

# Question 3:

As stated in the project manual, a high dimensionality can be harmful for performing large-scale data manipulation, which is sometimes referred to as "The Curse of Dimensionality". And since

the document-term TF-IDF matrix is sparse and low-rank, we could transform the features into a lower dimensional space. In this question, we aimed to decrease the dimensionality with two methods. We first used `TruncatedSVD` function from `sklearn.decomposition` to perform LSI (Latent Semantic Indexing) with k=50, random_state = 42. Then we applied NMF also with k=50, random_state = 42 to compare the Frobenius Norm of these two methods. The results are shown below:

| | |
|---|---|
| Train (LSI & NMF) | (4732, 50) |
| Test (LSI & NMF) | (3150, 50) |

Table 2: Shape of the TF-IDF matrices of the train and test subsets

| | |
|---|---|
| $\|\|X - U_k \Sigma_k V_k^T\|\|_F^2$ (LSI) | 4107.971183106478 |
| $\|\|X - WH\|\|_F^2$ (NMF) | 4144.733226773768 |

Table 3: Frobenius Norm comparison with different dimensionality reduction methods

From the table given above, we can see that the dimensions of our matrix were greatly reduced and LSI had a slightly lower Frobenius Norm. This difference in error can be explained by the constraints on the resulting matrices of NMF in which both W and H must be non-negative, and $\|\|X - U_k \Sigma_k V_k^T\|\|_F^2$ provides the best k rank approximation of X.

# Question 4:

After we had successfully reduced the dimension of our matrix with both LSI and NMF, we were ready to begin the classification process by training different classifiers with our training dataset and test those trained classifiers with the testing dataset later. In this question, we first trained two SVMs with a hard ($\gamma = 1000$) margin and a soft ($\gamma = 0.0001$) margin. In these extreme cases, we compared them with the ROC curve and confusion matrices of them. To quantify the differences between them, we also provided the accuracy, recall, precision and F-1 score of both cases.

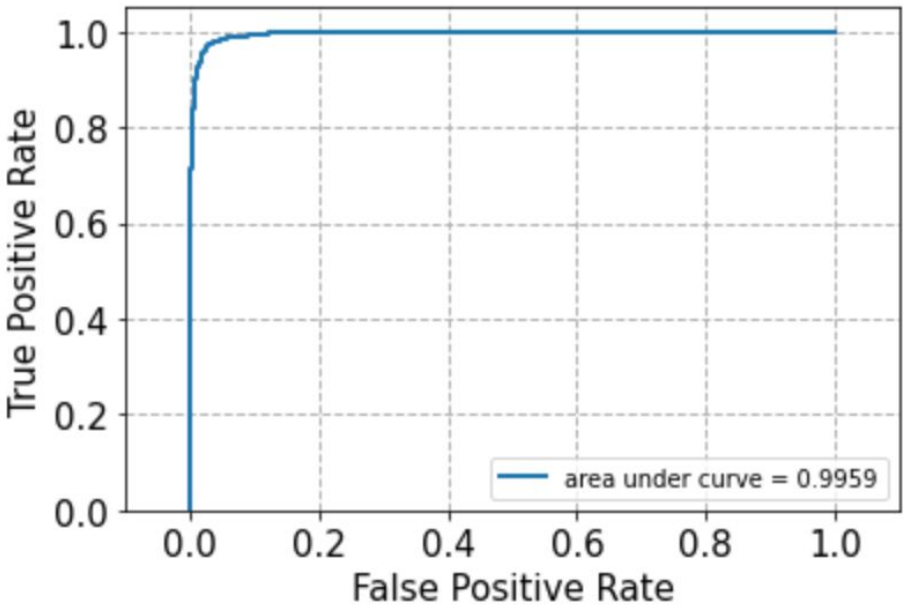



Figure 2: ROC curve of hard margin SVM

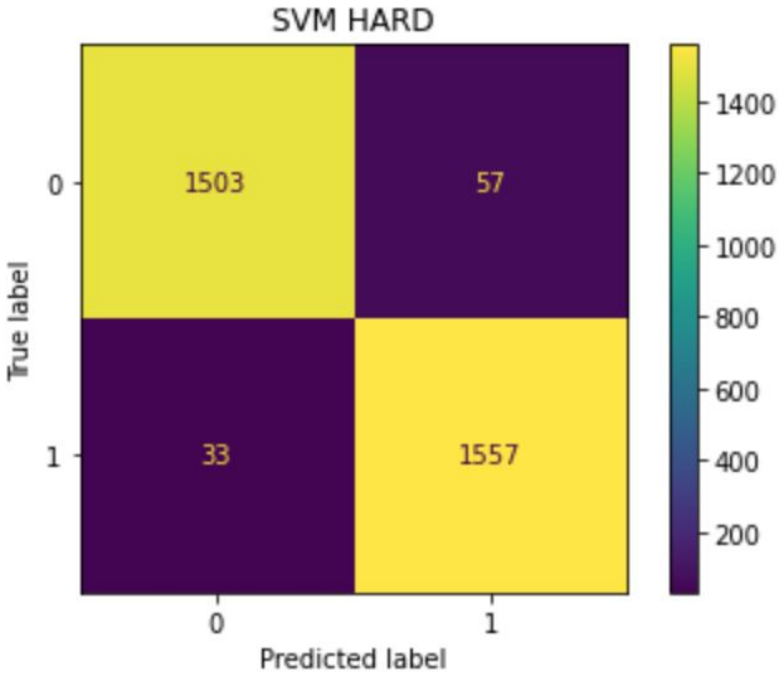



Figure 3: Confusion matrix of hard margin SVM

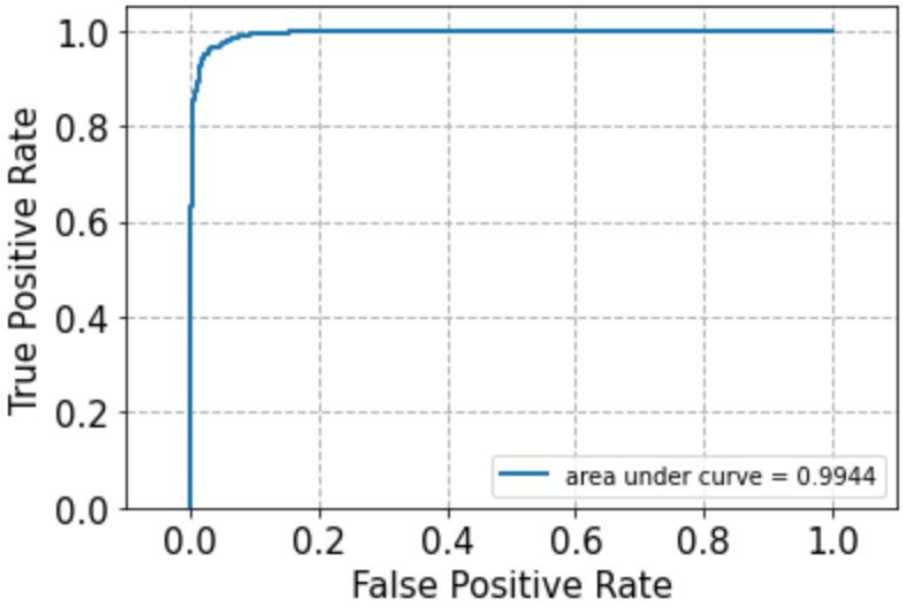



Figure 4: ROC curve of soft margin SVM

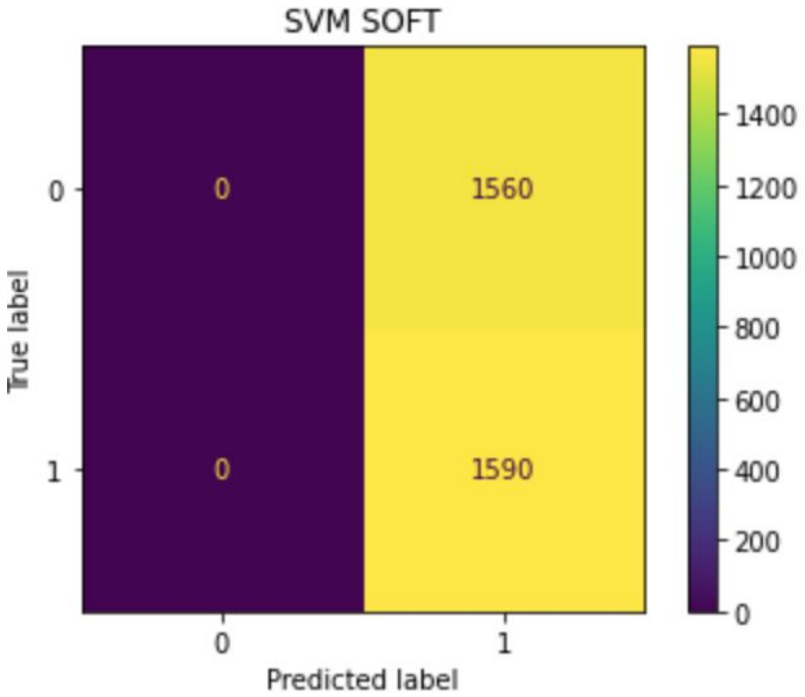



Figure 5: Confusion matrix of soft margin SVM

| | $\gamma = 1000$ (hard margin) | $\gamma = 0.0001$ (soft margin) |
|---|---|---|
| Accuracy | 0.9714285714285714 | 0.5047619047619047 |
| Recall | 0.9792452830188679 | 1 |
| Precision | 0.9646840148698885 | 0.5047619047619047 |
| F-1 score | 0.9719101123595505 | 0.6708860759493671 |

Table 4: Data of hard and soft margin SVM

From the given plots and table, we can see that hard margins have a better and more stable performance given by its higher value in both accuracy and precision. From the ROC we could also notice a minor difference that soft margin has a lower curvature than hard margin. The ROC curve of soft margin looked OK and reasonable, but not as good as that of hard margin because its accuracy and precision were comparatively lower, whereas the Recall value was almost 1 which was very high. This property did not conflict with other metrics since the soft method might not find the optimal solution eventually due to the fact that it is not strict or rigid about the misclassified cases while the hard method is very stringent.

In the next part of this question we applied a 5-fold cross-validation to find the best $\gamma$ in the range of [0.001, 0.01, 0.1, 1, 10, 100, 1000] by using SVC and GridSearchCV. We wrote a helper function that would generate the parameters for us and another helper function that helped us find the best mean_test_score value and its corresponding $\gamma$. With the same procedures performed for each different $\gamma$, we found that the best $\gamma$ that gave us the optimal mean_test_score value in this range was 10. The ROC curve, confusion matrix, accuracy, recall, precision and F-1 score are provided for this best case. From the reported values, we could clearly see that this case had a greater performance than both hard margin and soft margin cases.

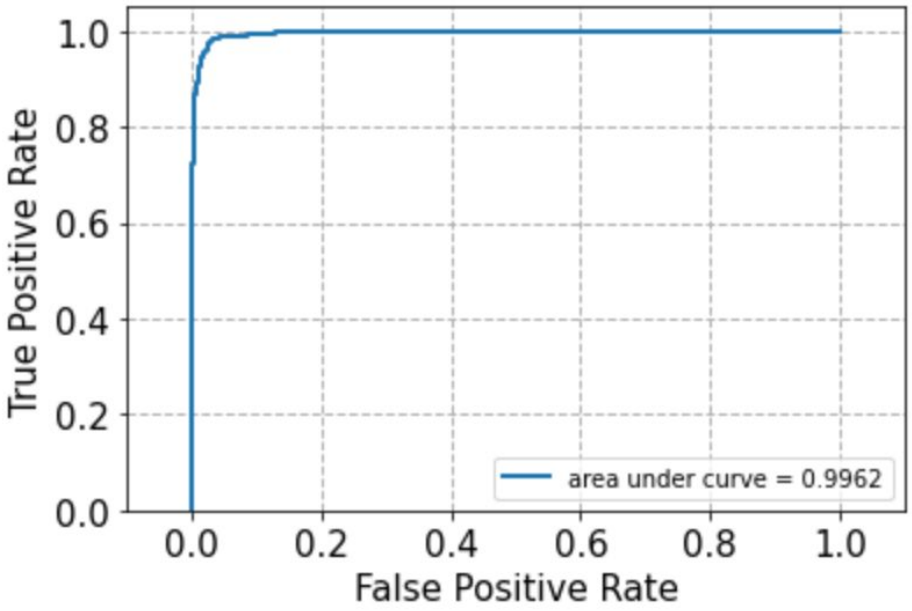



Figure 6: ROC curve of γ=10

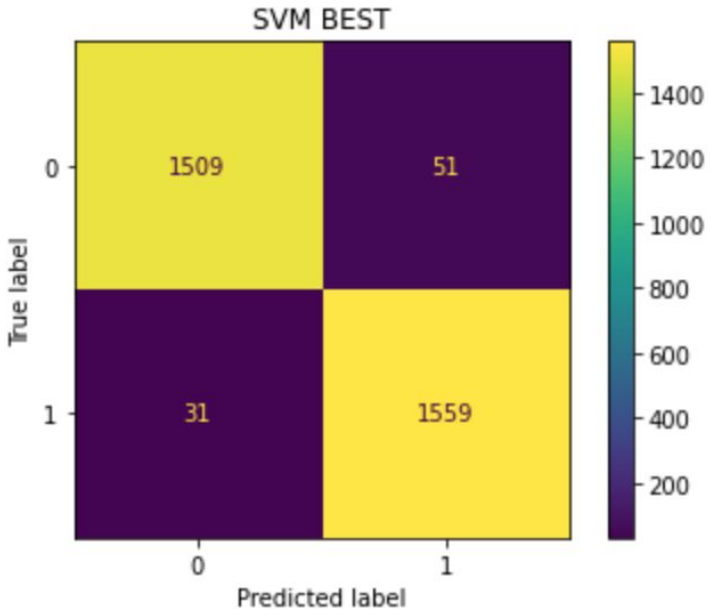


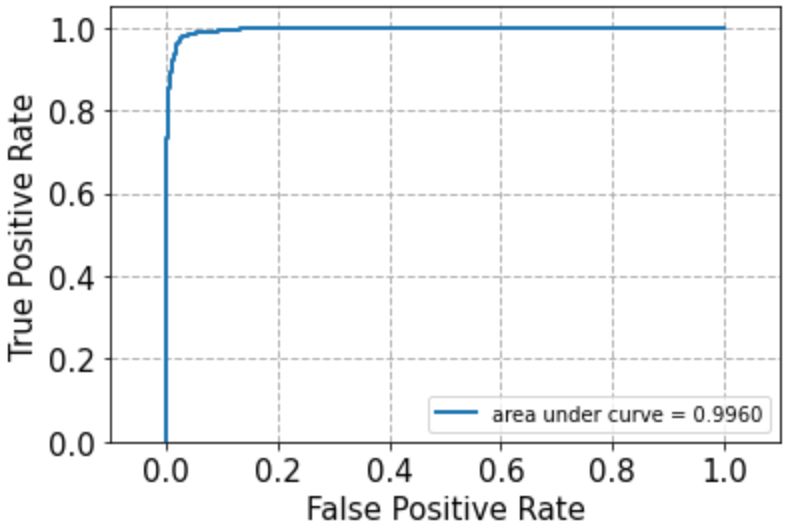

Figure 7: Confusion matrix of γ=10

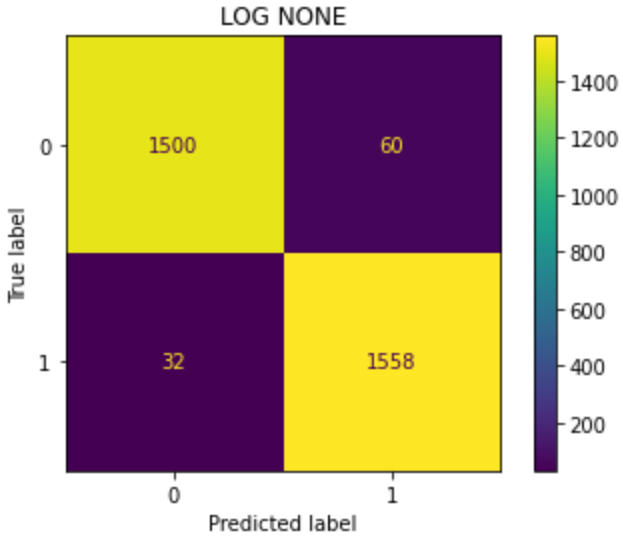
| | |
|---|---|
| Accuracy | 0.9739682539682539 |
| Recall | 0.980503144654088 |
| Precision | 0.9683229813664597 |
| F-1 score | 0.974375 |

Table 5: The best case SVM data when γ=10

# Question 5:

Similar to the last question, we still worked on the same SVM but with both L1 and L2 regularization from functions imported from sklearn.linear model.LogisticRegression, and also 5-fold cross-validation to find the value in the range of [0.001, 0.01, 0.1, 1, 10, 100, 1000] by using SVC and GridSearchCV. With the similar techniques applied with two regularizations, the results for w/o regularization, w/L1 regularization, and w/L2 regularization are provided below:



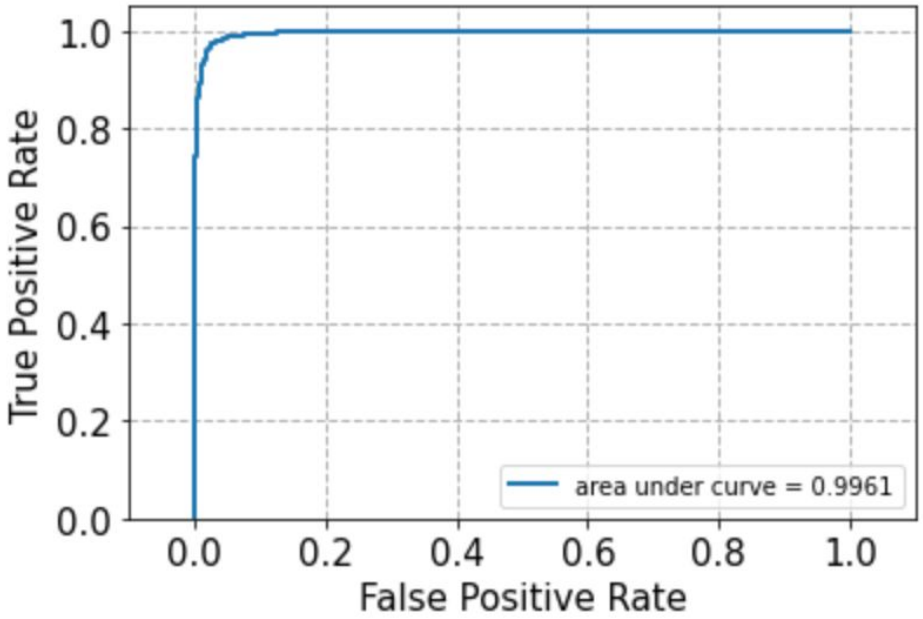



Figure 8: ROC curve of γ=10 with L1 regularization

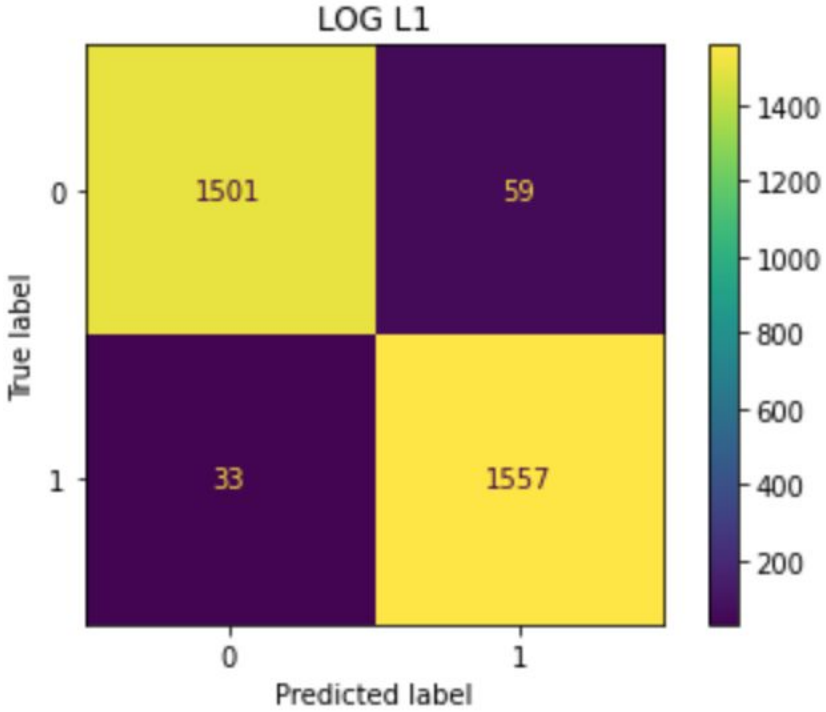



Figure 9: Confusion matrix of γ=10 with L1 regularization

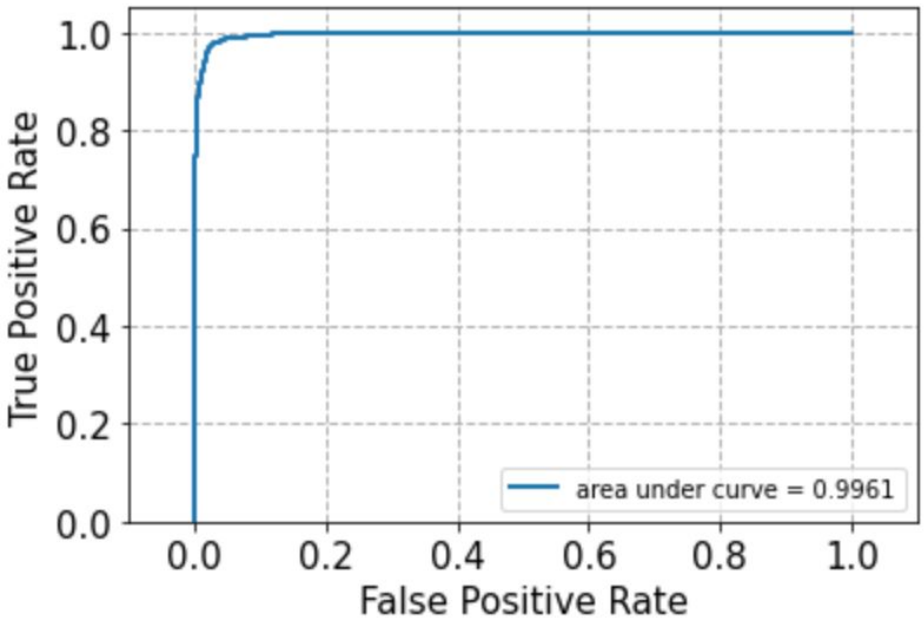



Figure 10: ROC curve of γ=10 with L2 regularization

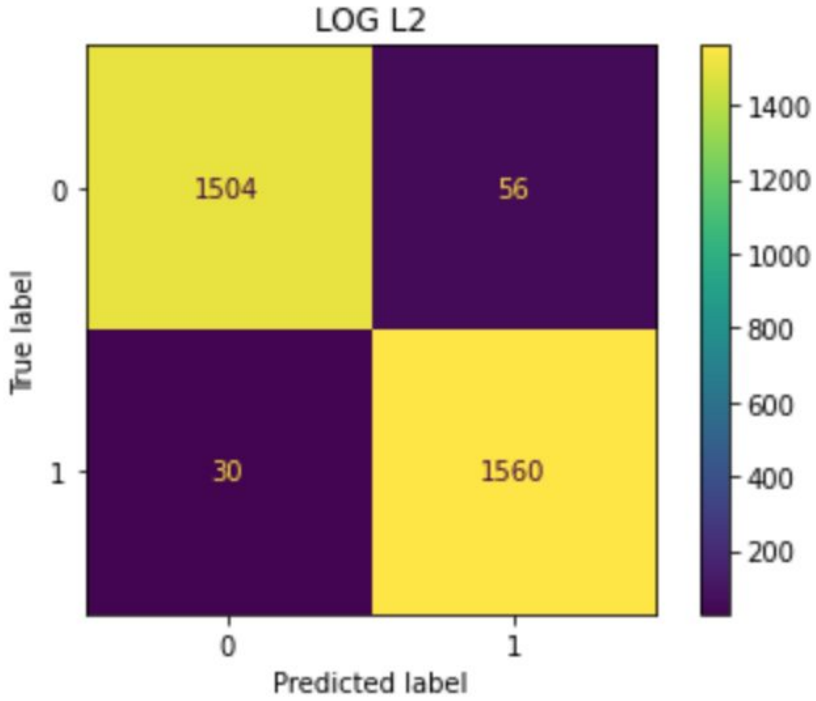



Figure 11: Confusion matrix of γ=10 with L2 regularization

| | W/O regularization | L1 regularization | L2 regularization |
|---|---|---|---|
| Accuracy | 0.9707936507936508 | 0.9707936507936508 | 0.9726984126984127 |
| Recall | 0.979874213836478 | 0.9792452830188679 | 0.9811320754716981 |
| Precision | 0.9629171817058096 | 0.963490099009901 | 0.9653465346534653 |
| F-1 score | 0.9713216957605986 | 0.9713038053649408 | 0.9731752963194011 |

Table 6: The best case SVM data when $\gamma$=10 with different regularizations

From the figures and data given above, we can conclude that with regularizations, the performance of SVM is improved, and even better with higher levels of regularization. C=10 for L1 regularization and C = 100 for L2 regularization gave us the best performance. Despite the similarities of the coefficients, we can see a slight improvement on each sector with regularization. In most cases, the larger the regularization parameter, the less the error. Moreover, the learned coefficients were quite different since the L1 case contained mostly smaller numbers and several 0's while the L2 case did not. This difference in learned coefficients could affect our selection of regularization when training different dataset. If we want to catch all the features of a dataset, we could use L2.

Both logistic regression and SVM are trying to classify data points using a linear decision boundary, but SVM tries to find the best margin that separates the classes while logistic regression does not since it has different decision boundaries and weights that are close to the optimal point. Their performance could differ when we are trying to classify different sizes of data.

# Question 6:

For this question, we trained and tested a GaussianNB classifier and performed the same procedures like the previous questions. Naive Bayes classifiers use the assumption that features are statistically independent of each other when conditioned by the class the data point belongs to, to simplify the calculation for the Maximum A Posteriori (MAP) estimation of the labels. Here are the results we obtained:

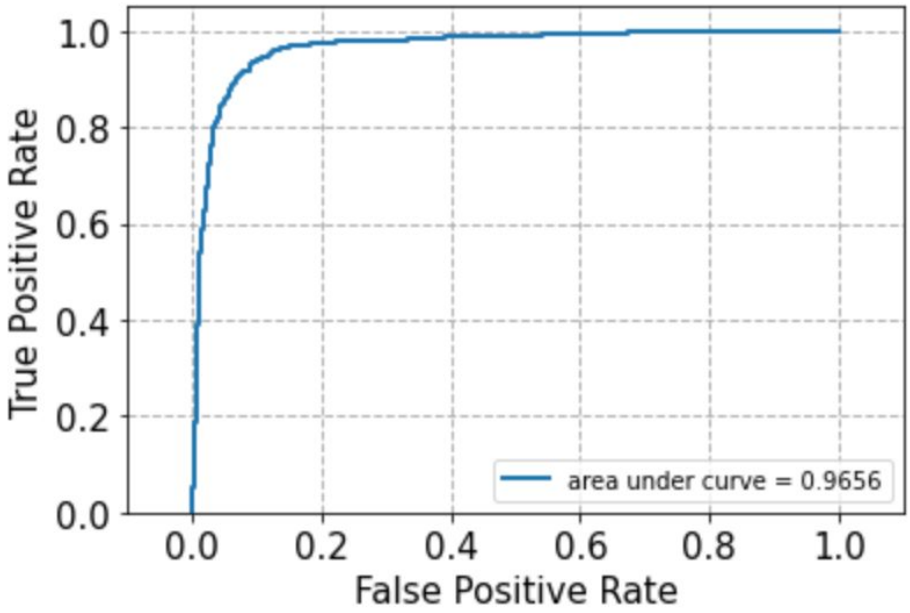



Figure 12: ROC curve GaussianNB classifier

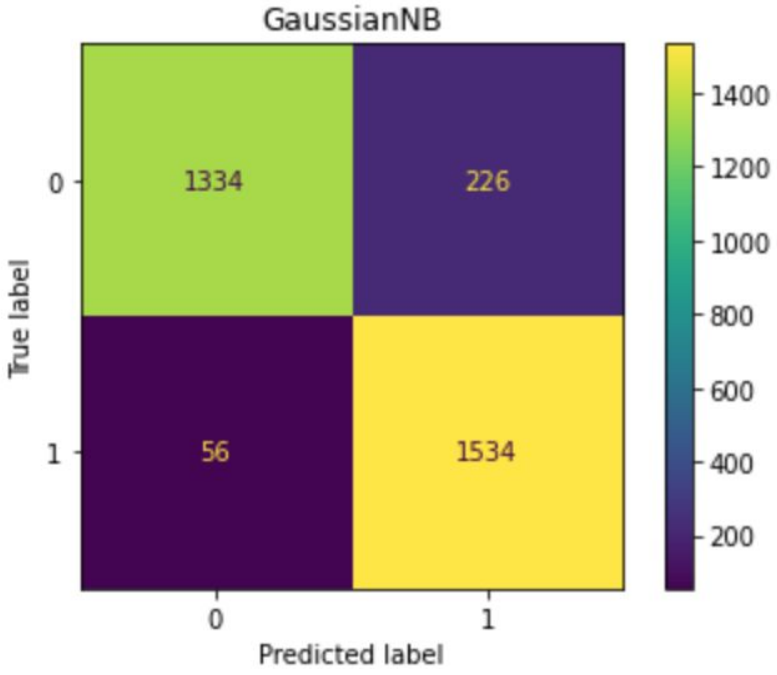



Figure 13: Confusion matrix of GaussianNB classifier

| | |
|---|---|
| Accuracy | 0.9104761904761904 |
| Recall | 0.9647798742138365 |
| Precision | 0.8715909090909091 |

| | |
|---|---|
| F-1 score | 0.915820895522388 |

Table 7: Data of GaussianNB classifier

# Question 7:

Since we have gone through the complete process of training and testing a classier. However, there are lots of parameters that we can tune. In this part, we ne-tune the parameters by using pipeline and 5-fold cross-validation in loading data, feature extraction, dimensionality reduction, classification process. We utilized sklearn.pipeline for pipelining and SVC and GridSearchCV for cross validation. The total 64 results are shown below:

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_clf | param_reduce_dim | param_reduce_dim__n_components | param_vect | params | split |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 5.208929 | 0.200096 | 0.233085 | 0.010124 | SVC(C=10, break_ties=False, cache_size=200, cl... | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': SVC(C=10, break_ties=False, cache_size... | |
| 1 | 55.807906 | 0.714026 | 13.156616 | 0.643642 | SVC(C=10, break_ties=False, cache_size=200, cl... | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': SVC(C=10, break_ties=False, cache_size... | |
| 2 | 4.361330 | 0.114264 | 0.226974 | 0.006284 | SVC(C=10, break_ties=False, cache_size=200, cl... | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': SVC(C=10, break_ties=False, cache_size... | |
| 3 | 54.942048 | 0.657343 | 13.353558 | 1.033389 | SVC(C=10, break_ties=False, cache_size=200, cl... | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': SVC(C=10, break_ties=False, cache_size... | |
| 4 | 24.139803 | 7.155204 | 0.468712 | 0.020934 | SVC(C=10, break_ties=False, cache_size=200, cl... | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': SVC(C=10, break_ties=False, cache_size... | |
| 5 | 25.953358 | 4.061540 | 13.220048 | 0.650754 | SVC(C=10, break_ties=False, cache_size=200, cl... | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': SVC(C=10, break_ties=False, cache_size... | |
| 6 | 20.588964 | 5.560501 | 0.414500 | 0.013876 | SVC(C=10, break_ties=False, cache_size=200, cl... | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': SVC(C=10, break_ties=False, cache_size... | |
| 7 | 21.241998 | 4.093725 | 13.106034 | 0.661635 | SVC(C=10, break_ties=False, cache_size=200, cl... | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': SVC(C=10, break_ties=False, cache_size... | |
| 8 | 0.533412 | 0.011205 | 0.204196 | 0.005979 | LogisticRegression(C=10, class_weight=None, du... | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': LogisticRegression(C=10, class_weight=... | |
| 9 | 0.469575 | 0.014049 | 12.982197 | 0.684633 | LogisticRegression(C=10, class_weight=None, du... | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': LogisticRegression(C=10, class_weight=... | |
| 10 | 0.506460 | 0.018176 | 0.198121 | 0.009061 | LogisticRegression(C=10, class_weight=None, du... | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': LogisticRegression(C=10, class_weight=... | |
| 11 | 0.432256 | 0.023343 | 13.054764 | 0.623190 | LogisticRegression(C=10, | TruncatedSVD(algorithm='randomized', | 50 | CountVectorizer(analyzer= | {'clf': LogisticRegression(C=10, | |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 20 | 0.593535 | 0.038660 | 0.389192 | 0.013191 | LogisticRegression(C=100, class_weight=None, d... | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': LogisticRegression(C=100, class_weight... |
| 21 | 0.508513 | 0.027382 | 13.271662 | 0.636103 | LogisticRegression(C=100, class_weight=None, d... | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': LogisticRegression(C=100, class_weight... |
| 22 | 0.537675 | 0.040960 | 0.367989 | 0.015265 | LogisticRegression(C=100, class_weight=None, d... | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': LogisticRegression(C=100, class_weight... |
| 23 | 0.416739 | 0.019498 | 13.203128 | 0.644062 | LogisticRegression(C=100, class_weight=None, d... | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': LogisticRegression(C=100, class_weight... |
| 24 | 0.437482 | 0.009574 | 0.203344 | 0.006200 | GaussianNB(priors=None, var_smoothing=1e-09) | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': GaussianNB(priors=None, var_smoothing=... |
| 25 | 0.365890 | 0.003695 | 13.040621 | 0.709506 | GaussianNB(priors=None, var_smoothing=1e-09) | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': GaussianNB(priors=None, var_smoothing=... |
| 26 | 0.377390 | 0.005269 | 0.198020 | 0.013971 | GaussianNB(priors=None, var_smoothing=1e-09) | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': GaussianNB(priors=None, var_smoothing=... |
| 27 | 0.327360 | 0.009044 | 13.053939 | 0.655998 | GaussianNB(priors=None, var_smoothing=1e-09) | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': GaussianNB(priors=None, var_smoothing=... |
| 28 | 0.450478 | 0.016290 | 0.346249 | 0.010294 | GaussianNB(priors=None, var_smoothing=1e-09) | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': GaussianNB(priors=None, var_smoothing=... |
| 29 | 0.398575 | 0.004935 | 13.184778 | 0.675879 | GaussianNB(priors=None, var_smoothing=1e-09) | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': GaussianNB(priors=None, var_smoothing=... |
| 30 | 0.406059 | 0.006196 | 0.316972 | 0.011764 | GaussianNB(priors=None, var_smoothing=1e-09) | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': GaussianNB(priors=None, var_smoothing=... |
| 31 | 0.361483 | 0.005985 | 13.224795 | 0.656313 | GaussianNB(priors=None, var_smoothing=1e-09) | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': GaussianNB(priors=None, var_smoothing=... |

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_clf | param_reduce_dim | param_reduce_dim__n_components | param_vect | params | split |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 4.212575 | 0.078046 | 0.185702 | 0.008794 | SVC(C=10, break_ties=False, cache_size=200, cl... | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': SVC(C=10, break_ties=False, cache_size... | |
| 1 | 45.232122 | 0.788550 | 10.621867 | 0.601374 | SVC(C=10, break_ties=False, cache_size=200, cl... | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': SVC(C=10, break_ties=False, cache_size... | |
| 2 | 3.553069 | 0.102465 | 0.184819 | 0.009103 | SVC(C=10, break_ties=False, cache_size=200, cl... | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': SVC(C=10, break_ties=False, cache_size... | |
| 3 | 44.225284 | 0.654440 | 10.455020 | 0.686493 | SVC(C=10, break_ties=False, cache_size=200, cl... | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': SVC(C=10, break_ties=False, cache_size... | |
| 4 | 29.113269 | 9.808678 | 0.358711 | 0.018527 | SVC(C=10, break_ties=False, cache_size=200, cl... | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': SVC(C=10, break_ties=False, cache_size... | |
| 5 | 27.298498 | 4.741831 | 10.594124 | 0.658409 | SVC(C=10, break_ties=False, cache_size=200, cl... | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': SVC(C=10, break_ties=False, cache_size... | |
| 6 | 20.173503 | 5.928646 | 0.330520 | 0.011354 | SVC(C=10, break_ties=False, cache_size=200, cl... | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': SVC(C=10, break_ties=False, cache_size... | |
| 7 | 18.068133 | 3.936925 | 10.592226 | 0.737965 | SVC(C=10, break_ties=False, cache_size=200, cl... | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': SVC(C=10, break_ties=False, cache_size... | |
| 8 | 0.447571 | 0.023891 | 0.162126 | 0.014428 | LogisticRegression(C=10, class_weight=None, du... | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': LogisticRegression(C=10, class_weight=... | |
| 9 | 0.397908 | 0.011296 | 10.460743 | 0.684003 | LogisticRegression(C=10, class_weight=None, du... | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': LogisticRegression(C=10, class_weight=... | |
| 10 | 0.406453 | 0.007410 | 0.154056 | 0.009788 | LogisticRegression(C=10, class_weight=None, du... | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': LogisticRegression(C=10, class_weight=... | |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 20 | 0.506351 | 0.026829 | 0.320037 | 0.019545 | LogisticRegression(C=100, class_weight=None, d... | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': LogisticRegression(C=100, class_weight... |
| 21 | 0.437134 | 0.020804 | 10.544322 | 0.650203 | LogisticRegression(C=100, class_weight=None, d... | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': LogisticRegression(C=100, class_weight... |
| 22 | 0.433547 | 0.024560 | 0.290450 | 0.014152 | LogisticRegression(C=100, class_weight=None, d... | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': LogisticRegression(C=100, class_weight... |
| 23 | 0.394180 | 0.035304 | 10.493838 | 0.692278 | LogisticRegression(C=100, class_weight=None, d... | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': LogisticRegression(C=100, class_weight... |
| 24 | 0.380919 | 0.012151 | 0.151960 | 0.009807 | GaussianNB(priors=None, var_smoothing=1e-09) | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': GaussianNB(priors=None, var_smoothing=... |
| 25 | 0.317280 | 0.010876 | 10.426629 | 0.668695 | GaussianNB(priors=None, var_smoothing=1e-09) | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': GaussianNB(priors=None, var_smoothing=... |
| 26 | 0.321491 | 0.010464 | 0.146782 | 0.006587 | GaussianNB(priors=None, var_smoothing=1e-09) | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': GaussianNB(priors=None, var_smoothing=... |
| 27 | 0.283707 | 0.008031 | 10.367405 | 0.697428 | GaussianNB(priors=None, var_smoothing=1e-09) | TruncatedSVD(algorithm='randomized', n_compone... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': GaussianNB(priors=None, var_smoothing=... |
| 28 | 0.396691 | 0.021719 | 0.258022 | 0.016719 | GaussianNB(priors=None, var_smoothing=1e-09) | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': GaussianNB(priors=None, var_smoothing=... |
| 29 | 0.352208 | 0.009568 | 10.427198 | 0.709761 | GaussianNB(priors=None, var_smoothing=1e-09) | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': GaussianNB(priors=None, var_smoothing=... |
| 30 | 0.357267 | 0.008784 | 0.239902 | 0.014579 | GaussianNB(priors=None, var_smoothing=1e-09) | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer='word', binary=False,... | {'clf': GaussianNB(priors=None, var_smoothing=... |
| 31 | 0.315508 | 0.006373 | 10.458577 | 0.651150 | GaussianNB(priors=None, var_smoothing=1e-09) | NMF(alpha=0.0, beta_loss='frobenius', init=Non... | 50 | CountVectorizer(analyzer=<function stem_rmv_pu... | {'clf': GaussianNB(priors=None, var_smoothing=... |

We compared the values of mean_test_score's of the tables above, and we can tell that the best result is 0.975698. Therefore, the corresponding best parameters are: NOT remove "headers" and "footers", min_df = 5, use lemmatization, LSI, LogisticRegression with L1 regularization(C = 10). For more detailed information, please refer to the tables in our ipynb file.

# Question 8:

(a) Compared to probabilities themselves, the ratio of co-occurrence probabilities,the ratio is better able to distinguish relevant words (solid and gas) from irrelevant words (water and fashion) and it is also better able to discriminate between the two relevant words.

(b) No, GLoVE embeddings will return different vectors for the word running when training the embedding. Because GLoVE embeddings distinguish between words. The relevancy between word running and in and relevance between word running and for is different. If we use a pre-trained embedding, it will return the same vector for the word running.

(c) I will expect that ||GLoVE["queen"] - GLoVE["king"] - GLoVE["wife"] + GLoVE["husband"]||$_2$ should be the largest one. ||GLoVE["queen"] - GLoVE["king"]||$_2$ should be larger than the ||GLoVE["wife"] - GLoVE["husband"]||$_2$ since the co-occurrence probability of wife and husband is larger than queen and king.

(d) I would choose the lemmatization method. Because stemming reduces the word-form to stems and leads to erroneousness. And lemmatization will consider the whole context and lemmatize the word to its meaningful base form. For example, if we stem "caring", we will get "car" while lemmatization gives us "care".

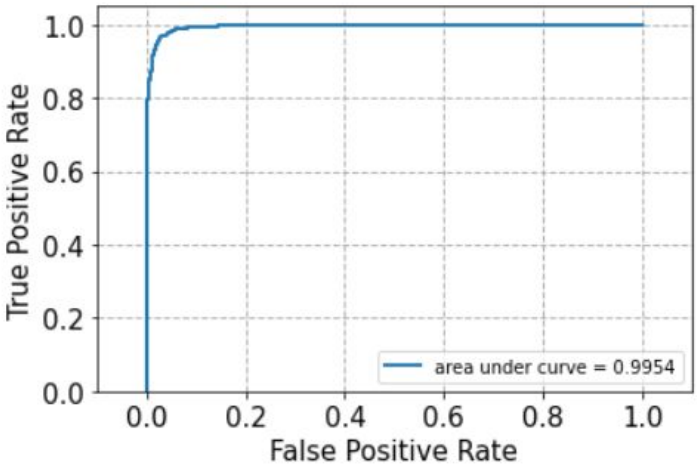
# Question 9:

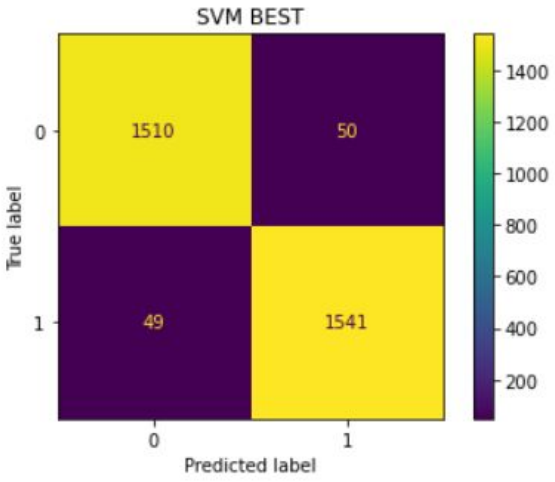
(a) I first wrote a basic data cleaning function to remove numbers and punctuations, convert characters to lowercase, and substitute multiple white spaces to single whitespace. And then we use the method TweetTokenizer() to tokenize the text. Also, we have two other functions to filter out stop words and remove the words that are not in the pretrained embedded dictionaries. In the project statement it mentioned that we can use some important topical words such as "Keywords: …" and " Subject: …". However, with a deeper look inside the data, a lot of subject details are useless words or phrases.
After applying the strategies above, we cleaned our train and test data. And we iterate words of our data and find the corresponding vector inside the embeddings. Sum them up and get the average embedding of all words in each document.

(b) We tried different classifiers and parameters. SVM gives a very decent result, with lambda = 100, we get the accuracy at 96.86%.

SVM BEST:

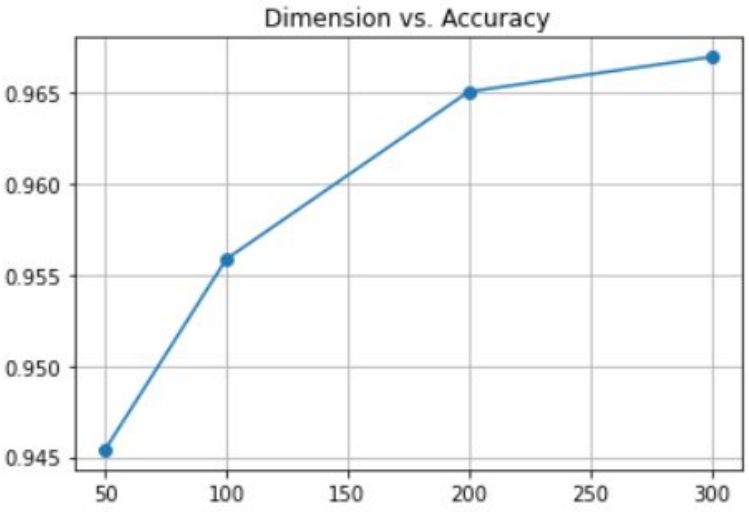
```
svm_best accuracy: 0.9685714285714285
svm_best recall: 0.9691823899371069
svm_best precision: 0.9685732243871779
svm_best F-1 score: 0.9688777114115058
svm_best confusion_matrix:
 [[1510   50]
 [  49 1541]]
```



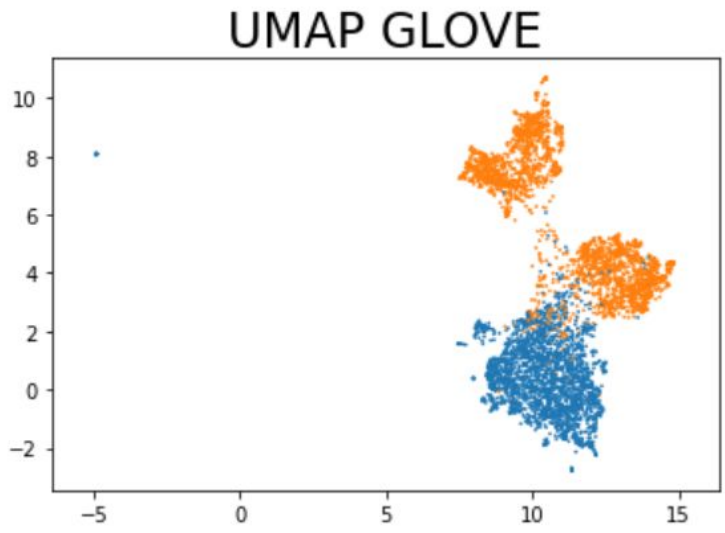
# Question 10:

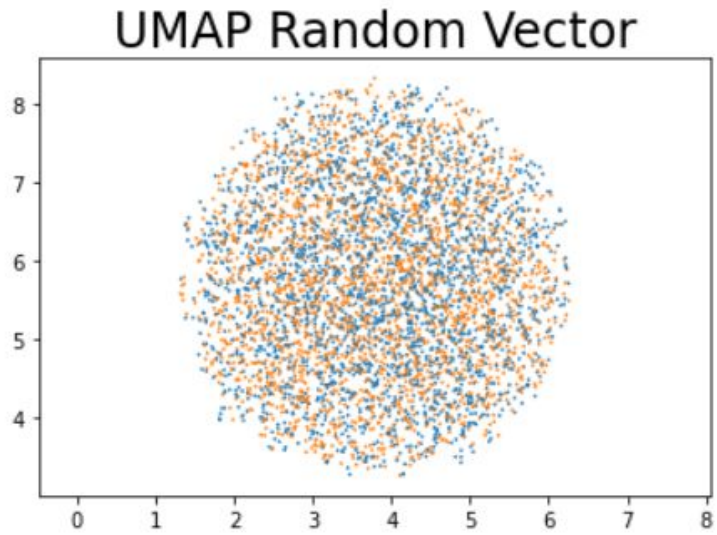
From the plot we can see that as the dimension size of Global embedding increases, we get a better accuracy, which matches with our expectation. Also, the accuracy is not increasing that much as dimension increasing. The accuracy of 300 dimensions is only around 2% larger than the 50 dimensions.




# Question 11:

We can see that for the GLoVE embedding case, the two clusters are divided and only have a very small overlapping region. For the randomized vectors case, we cannot find any single cluster and all the data points are overlapping and we cannot classify them.


UMAP GLOVE
10
8
6
4
2
0
−2
−5
0
5
10
15


UMAP Random Vector
8
7
6
5
4
0
1
2
3
4
5
6
7
8

# Question 12:

In this part we compared Naive Bayes classifier, One Vs One, and One Vs Rest with all of them implemented with TF-IDF, LSI and NMF. Below will show and compare these nine outcomes with both confusion matrices and specific data.

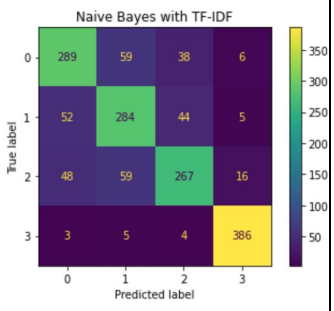
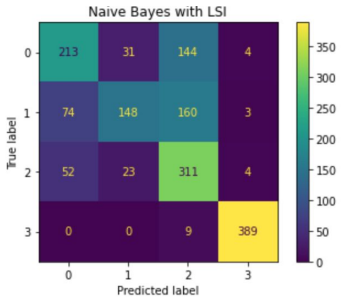
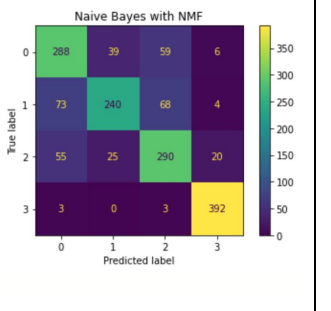
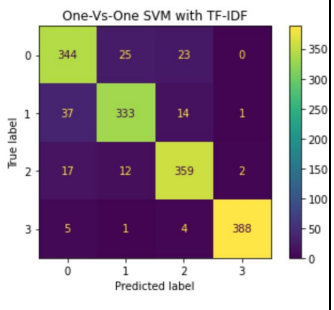
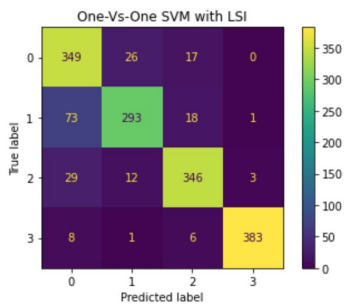
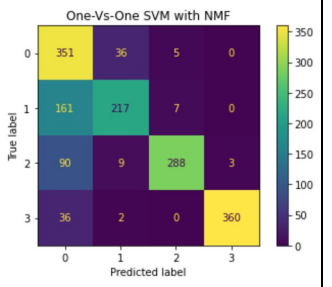
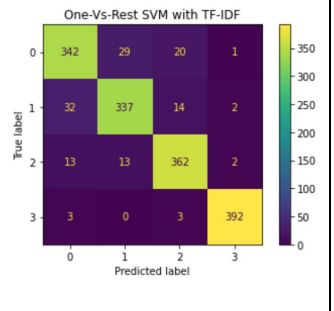
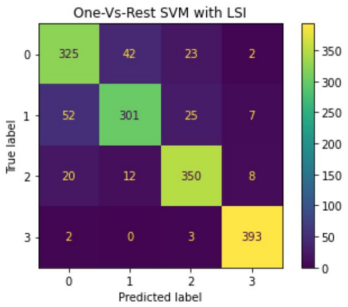
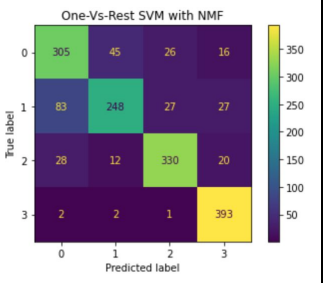
| | TF-IDF | LSI | NMF |
|---|---|---|---|
| Naive Bayes |  |  |  |
| One Vs One |  |  |  |
| One Vs Rest |  |  |  |

Table 8: Confusion matrices of the nine outcomes

| | | Accuracy | Recall | Precision | F-1 Score |
|---|---|---|---|---|---|
| Naive Bayes | TF-IDF | 0.7833865814696486 | 0.7823429666170155 | 0.7815080576903742 | 0.7812589441566716 |

| | | | | | |
|---|---|---|---|---|---|
| | LSI | 0.6779552715654952 | 0.675651440865906 | 0.7079723218237413 | 0.6688429634576198 |
| | NMF | 0.7731629392971247 | 0.7716462169082414 | 0.7740526656538869 | 0.7697607627743176 |
| One Vs One | TF-IDF | 0.9099041533546326 | 0.9094683194288362 | 0.910249876151614 | 0.9096863482198181 |
| | LSI | 0.876038338658147 | 0.8752090321140931 | 0.8816499086481204 | 0.8759767166354894 |
| | NMF | 0.7769968051118211 | 0.7755071696071337 | 0.8309654934929229 | 0.7827951594587443 |
| One Vs Rest | TF-IDF | 0.9156549520766774 | 0.9152258515593046 | 0.9151947117395871 | 0.9151650047016007 |
| | LSI | 0.8747603833865815 | 0.8739432244591971 | 0.8734445513184663 | 0.8732465425019968 |
| | NMF | 0.8153354632587859 | 0.8139520251822836 | 0.81467494165251 | 0.81073432777199 |

Table 9: Numerical data of the nine outcomes