Qiong Hu (405065032)
Daniel Ahn (504817439)
Riyya Hari Iyer (305427411)
Juan Piao (304959335)

Project 1 Report

Classification Analysis on Textual Data

ECE 219
Data Mining
January 20, 2021

Getting familiar with the dataset

QUESTION 1:

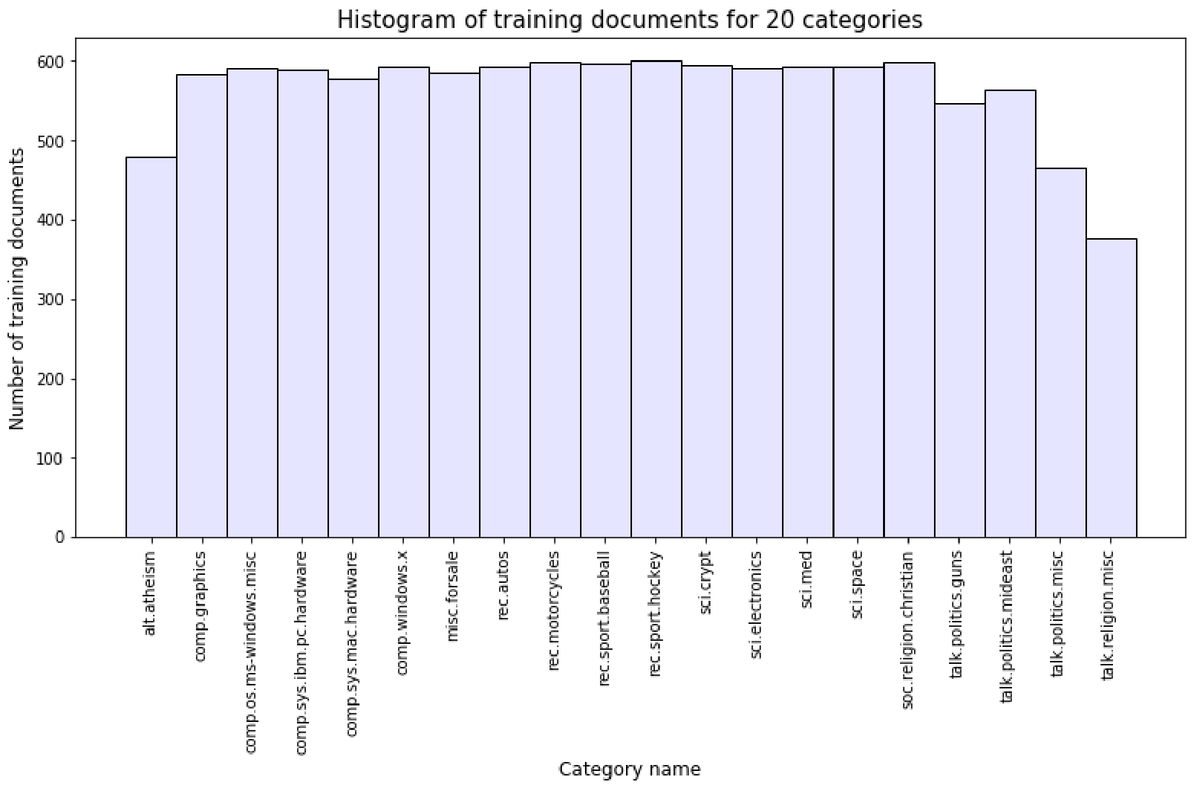

Figure 1: Histogram of training documents for each of the 20 categories

We checked that the average number of the training documents are 565.7, and the std is 56.78. By looking at the histogram in Fig. 1, we noticed that apart from 'alt.atheism', 'talk.politics.misc', and 'talk.religion.misc', all the other categories have more than 500 documents and are quite evenly distributed.

Binary Classification

1 Feature Extraction

QUESTION 2: By observing the raw dataset, we notice that there are many non-alphabetic terms including numbers, email addresses, and other meaningless characters, which are not directly related to the topic of the textual data. Therefore, we exclude all the non-alphabetic terms when extracting features. The resulting shape of the TF-IDF matrices of the train and test subsets are as follows:

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

Dataset	Document Number	Feature Number
Train	4732	12601
Test	3150	12601

2 Dimensionality Reduction

QUESTION 3: We use function “TruncatedSVD” and “NMF” from library “sklearn.decomposition” to perform LSI and NMF. Both methods reduce the feature dimensionality of train set and test test to 50 terms.

Table 2: Frobenius Norm of residual between original data and reconstruction

Method	Train Set	Test Set
LSI	64.19	53.40
NMF	64.45	53.54

We notice the result with NMF is larger than LSI in both train set and test set. The reason might be that, the factorization in NMF is not unique, so it’s possible for NMF to get a local minimum solution, while LSI searches for the global minimum solution without constraints of number of topics. Therefore, LSI has a better performance here.

3 Classification Algorithms

QUESTION 4: We use function “LinearSVC” from library “”sklearn.svm” to train hard margin and soft margin linear SVMs. And the evaluation results are as follows:

Table 3: Evaluation of Linear SVMs

Classifier	Accuracy	F-1 Score	Recall	Precision
Hard Margin SVM ($\gamma = 1000$)	0.9749	0.9754	0.9854	0.9666
Soft Margin SVM ($\gamma = 0.0001$)	0.7314	0.7899	1.0000	0.6527
Best SVM ($\gamma = 10$)	0.9768	0.9772	0.9855	0.9691

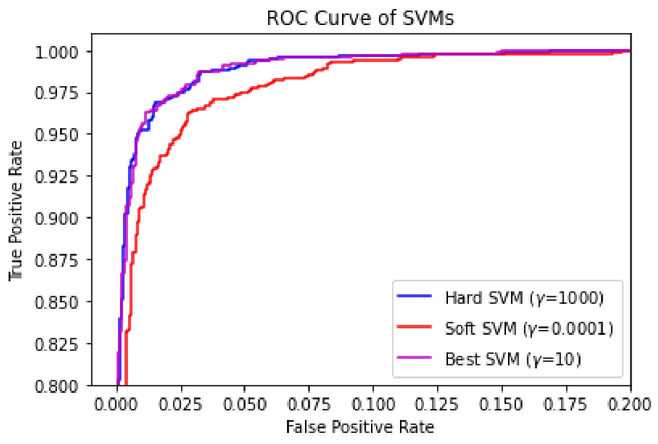

Figure 2: ROC Curves of SVMs

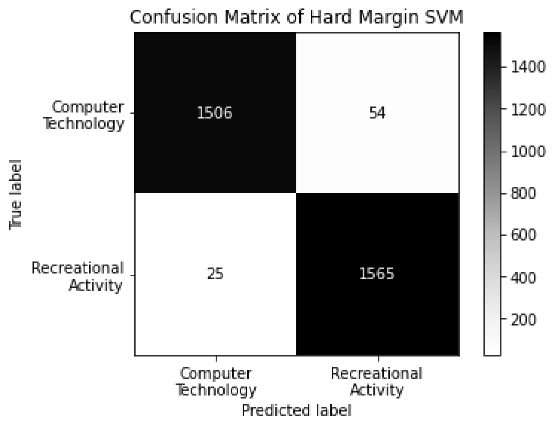

Figure 3: Hard Margin SVM (γ=1000)

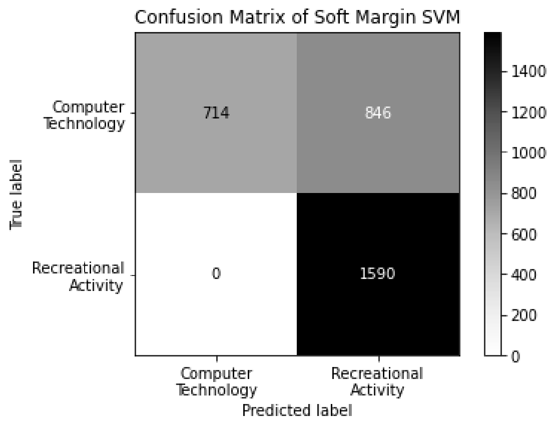

Figure 4: Soft Margin SVM (γ=0.0001)

We notice that hard margin SVM has a better performance than soft margin SVM in all the metrics, as well as ROC curve, according to Table 3 and Fig. 2-4. The reason is that, when the γ is really small, the optimization objective almost becomes to minimize $||\mathbf{w}||_2^2$ only, and therefore, the penalty for misclassification of individual points is very low, resulting in ill-fitting classification results.

Table 4: Cross-validation for best Linear SVM Parameter γ

k	-3	-2	-1	0	1	2	3
γ	10^{-3}	10^{-2}	10^{-1}	1	10	10^2	10^3
Average Score	0.9368	0.9637	0.9706	0.9755	**0.9784**	0.9765	0.9668

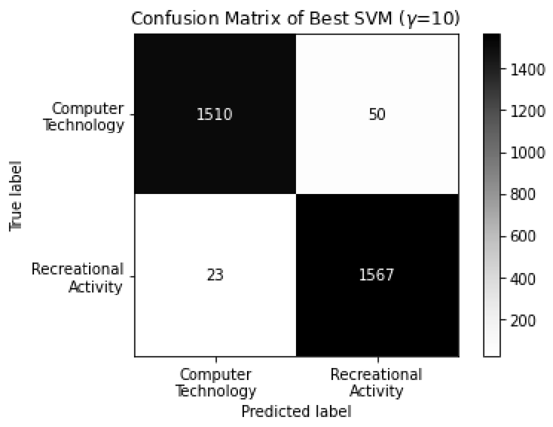

Figure 5: Best SVM (γ=10)

By using 5-fold cross-validation (as shown in Table 4), we find the best value of the parameter $\gamma = 10$, the ROC curve is plotted in Fig. 2, the confusion matrix is reported in Fig. 5, and all the evaluation metrics are reported in Table 3. We noticed that the overall performance of the best SVM is very close to that of the hard margin SVM.

QUESTION 5: We use function "LogisticRegression" from library "sklearn.linear_model" to train logistic classifiers. And we set "penalty" term of the function to "none", "l1" and "l2", respectively, to find out the effects of different regularization term.

By using 5-fold cross-validation, we found the best regularization strength for both L1 and L2 regularization is $k = 3$, meaning $C = 10^3$. All the evaluation results are reported in Table 5 and Fig. 6-10.

Table 5: Evaluation of Logistic Classifiers

Classifier	Accuracy	F-1 Score	Recall	Precision
w/o regularization	0.9749	0.9753	0.9836	0.9672
w/ L1 regularization ($C = 1000$)	0.9749	0.9754	0.9836	0.9672
w/ L2 regularization ($C = 1000$)	0.9743	0.9747	0.9830	0.9666

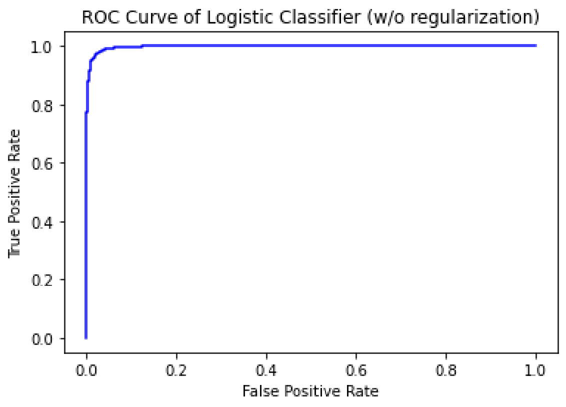

Figure 6: Without Regularization

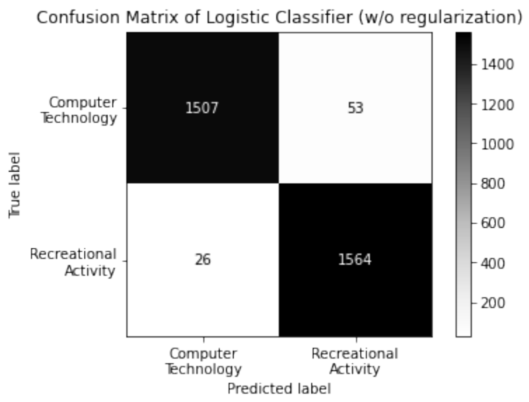

Figure 7: Without Regularization

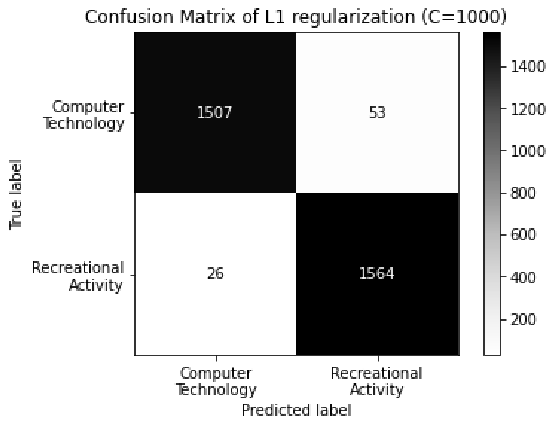

Figure 8: L1 Regularization

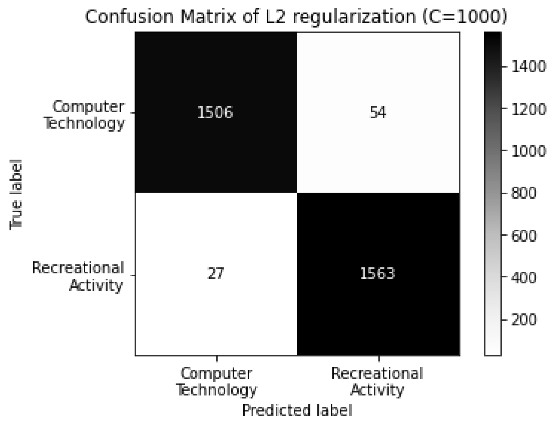

Figure 9: L2 Regularization

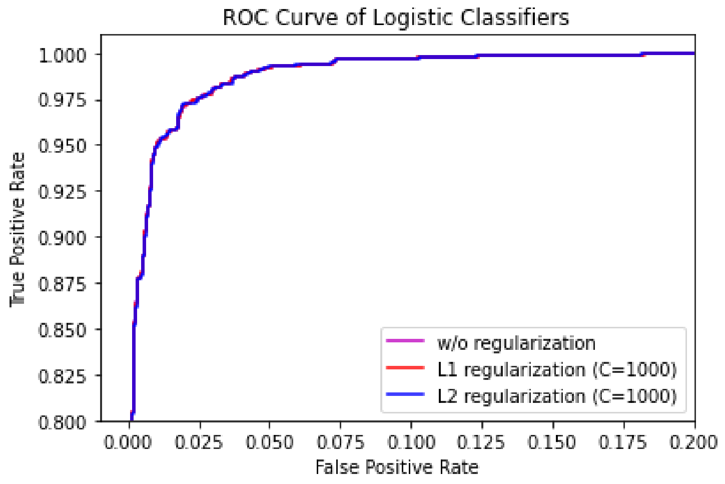

Figure 10: ROC Curves of Logistic Classifiers

- By observing the performance results and ROC curves, we found that three logistic classifiers only show negligible difference.

- Regularization (especially L2) affects test error in that it reduces it. This is primarily because it modifies the model in such a manner that it reduces the overfitting of the model to the training dataset, thereby making it capable of gauging the underlying pattern in the dataset which makes it a better fit for the testing dataset and results in higher test accuracy.

- Regularization decreases the coefficient estimates to zero and penalizes the high coefficients[3].

  Finally L1 or Lasso regularizations adds the absolute value of the coeffient as the magnitude for penalization while the L2 or ridge regularization adds the square magnitude of coeffient term[4].

  L1 creates sparsity in solutions and makes it robust to outliers[5] while L2 helps reduce test error and overfitting.

- SVM or Support Vector Machine (for 2 classes) creates a boundary where the accuracy with which a data point can be classified as being either one of the two classes is very low. So as you move away from this boundary line (surrounded by boundary regions on either side), you have a greater accuracy at classifying the data point as the class whose side it is on.

  Logistic Regression or LR is about probability of 0 or 1, i.e. either an outcome would successfully occur or it won't.

  SVM tries to make a boundary such that the separation between the two classes is as wide as possible (to include as much certainty as possible and rule it out)[6]. Decision boundary for LR is halfway where the event (data point) any or may not occur. This may be weighted as well[7].

  Although both have given a similar performance, SVM usually works slightly better than LR since it's more suited for semi-structured data in comparison to LR. Furthermore SVM has a decreased chance of overfitting while LR is more vulnerable to it[8].

QUESTION 6: We use function "GaussianNB" from library "sklearn.naive_bayes" to train Naïve Bayes classifier. And the results are as below:

Table 6: Evaluation of Naïve Bayes classifier

Classifier	Accuracy	F-1 Score	Recall	Precision
Naïve Bayes classifier	0.8435	0.8629	0.9755	0.7736

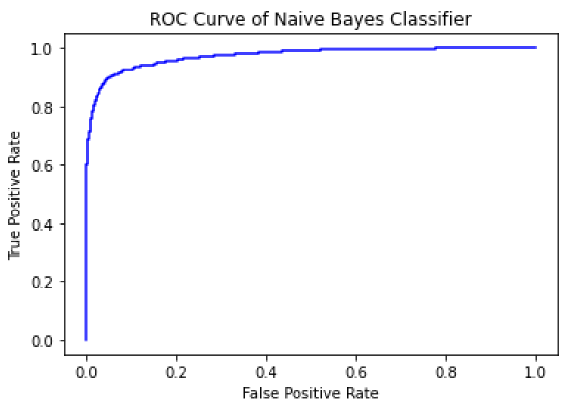

Figure 11: ROC Curve

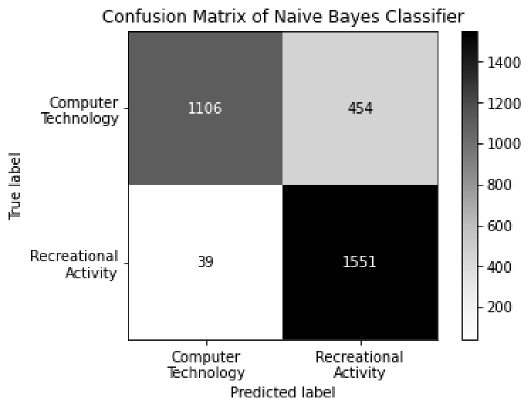

Figure 12: Confusion Matrix

QUESTION 7: We used function "Pipeline" from library "sklearn.pipeline" to do a grid search of parameters.

The pipeline is as follows: vectorizer → Tfidf Transformer → Dimension reduction → Classifier.

Table 7: Test Accuracy in Grid Search (Loading Data w/ "headers" "footers")

	min_df=3 LSI	min_df=5 LSI	min_df=3 NMF	min_df=5 NMF
Linear SVM	0.974851	0.974852	0.965133	0.966612
L1 Regularization	**0.976965**	0.976543	0.971472	0.972739
L2 Regularization	0.976753	0.976543	0.971472	0.972528
Naïve Bayes	0.861789	0.867283	0.954351	0.956675

Table 8: Test Accuracy in Grid Search (Loading Data w/o "headers" "footers")

	min_df=3 LSI	min_df=5 LSI	min_df=3 NMF	min_df=5 NMF
Linear SVM	0.960481	0.959847	0.943785	0.944209
L1 Regularization	0.967033	0.964919	0.958578	0.956255
L2 Regularization	**0.967455**	0.966187	0.958367	0.956467
Naive Bayes	0.829036	0.836646	0.937658	0.936180

We ran the pipeline twice (with and without headers and footers in the loading data), and present the results in Table 7-8, and found two best models in each circumstance. The two best combinations and their performance on test dataset are presented in Table 9 below, along with the ROC curves and confusion matrices in Fig. 13-16.

Table 9: Evaluation of Best Combinations

	Best Model 1	Best Model 2
Loading Data	w/ “headers” “footers”	w/o ”headers” “footers”
Feature Extraction	min_df = 3	min_df = 3
Dim Reduction	LSI	LSI
Classifier	L1 regularization	L2 regularization
Accuracy	0.9749	0.9660
F-1 Score	0.9754	0.9664
Recall	0.9843	0.9692
Precision	0.9666	0.9637

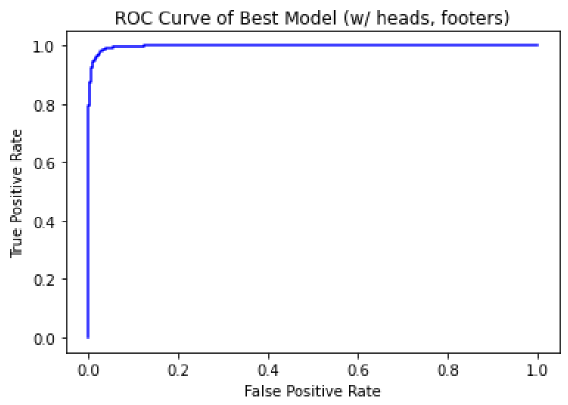

Figure 13: Best Model 1

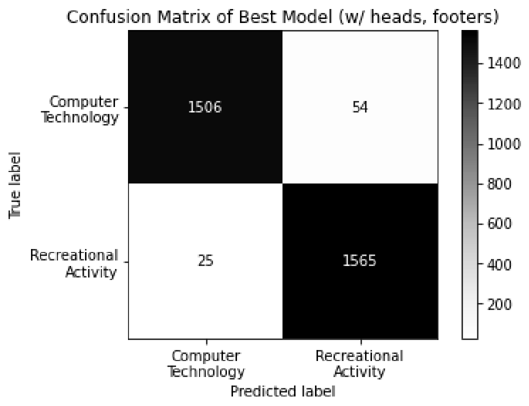

Figure 14: Best Model 1

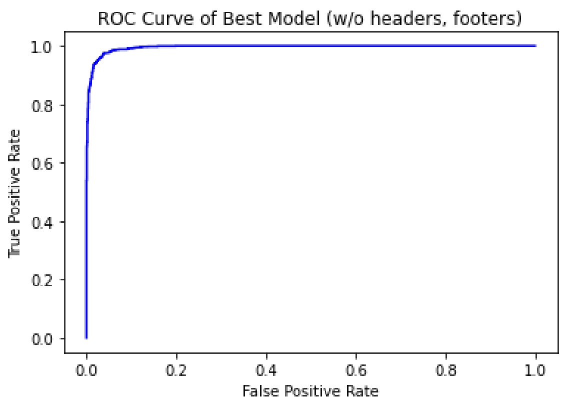

Figure 15: Best Model 2

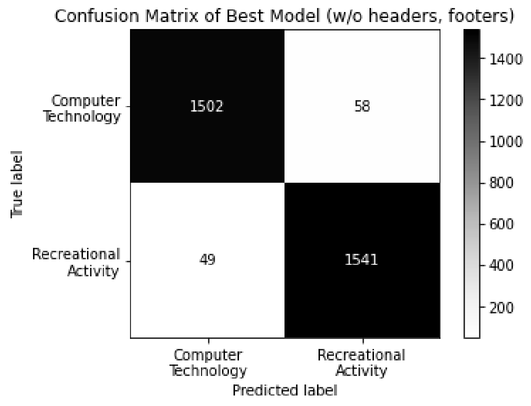

Figure 16: Best Model 2

From the above results, we notice that, whether or not the initial dataset load headers and footers, the best parameter for feature extraction is always min_df=3, and the best dimensionality reduction methods is always LSI.

Overall, the best combination is loading data with headers and footers, using min_df=3 for feature extraction, LSI for dimensionality reduction and logistic classifier with L1 regularization (strength $C = 1000$).

Word Embedding

QUESTION 8: About GLoVE embeddings:

(a) GLoVE is trained on co-occurrence probabilities rather than the probabilities itself because this is Unsupervised learning where the model has to decipher the pattern of the data by itself instead of an available labelled dataset. In such a case, it relies on the overall context of the word rather than just the word itself. GLoVE even demonstarted and proved that certain aspects of meaning could be directly extracted form co-occurrence probabilities. It studies the relationship between words by studying the ratio of their co-occurence probabilities with various probe words.

(b) GLoVE would likely return different vector for the two sentences "James is running in the park" and "James is running for the presidency" for the word running. This is because the GLoVE model takes the whole context into account and not just the word itself. Furthermore, unlike other NLP models, GLoVE takes the whole context into account, and not just that for the neighbouring words, resulting in higher accuracies. Thus for the first sentence, GLoVE associates running with the word park, while for the latter, GLoVE associates running with the word "Presidency".

(c) Ideally, the formula $w_d = w_b - w_a + w_c \Rightarrow w_d - w_b + w_a - w_c = 0$ should have been used and should have worked.

$\| GLoVE["queen"] - GLoVE["king"] - GLoVE["wife"] + GLoVE["husband"] \|_2 = 0$

$\| GLoVE["queen"] - GLoVE["king"] \|_2 = \| GLoVE["wife"] - GLoVE["husband"] \|_2$

since husband-wife and king-queen are embedded as analogies in GLoVE.

However, the values were found to be as follows (as per the code):

$\| GLoVE["queen"] - GLoVE["king"] \|_2 = 5.966258$

$\| GLoVE["wife"] - GLoVE["husband"] \|_2 = 3.1520464$

Apparently, while these two sets of words seem like analogies, the L_2 norm of the difference seemed to render different values. Consequently you expect the value of the first equation to be 2.8142116. However, the value of $\| GLoVE["queen"] - GLoVE["king"] - GLoVE["wife"] + GLoVE["husband"] \|_2$ was found to be equal to 6.1650367.

(d) We should use Lemmatization and not Stemming in case of GLoVE. While both remove inflectional forms of a word, Stemming truncates a word while Lemmatization returns 1 word out of the related set of words depending on the whole context, i.e. whether a verb was used or a noun (Democratic vs Democracy) as per [1]. Since GLoVE relies on the whole context surrounding a word and returns the word vector, Lemmatization of a word would work better before mapping it to its GLoVE embedding.

QUESTION 9:

(a) Describe a feature engineering process that uses GLoVE word embeddings to represent each document.

1) Preprocessing data : tokenize and lemmatize each document in the text.
2) Prepare GLoVE embeddings dictionary : load in the GloVe embedding and appending them to a dictionary.
3) Obtain GLoVE embeddings matrix : creating an embedding matrix for each word in each document in the training set. Obtain the embedding vector for each word from the embedding dictionary. For words not found in the embedding dictionary, we will have a matrix representation with all zeros.
4) Aggregate these words into a single vector: Use the column mean of the embedding matrix to obtain a vector to represent each document.

(b) Select a classifier model, train and evaluate it with your GLoVE-based feature.

We chose an SVM with $\gamma = 1$ as our classifier model.

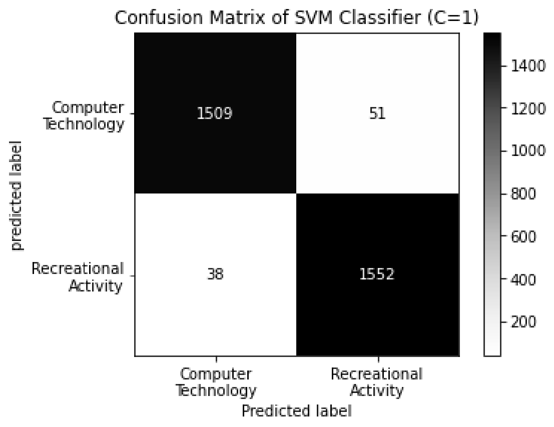

Figure 17: Confsion Matrix of SVM

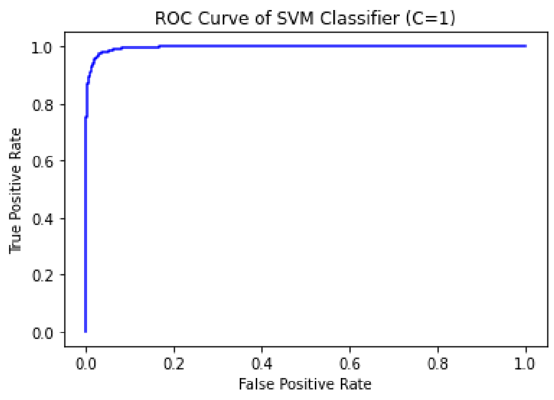

Figure 18: ROC of SVM

QUESTION 10:

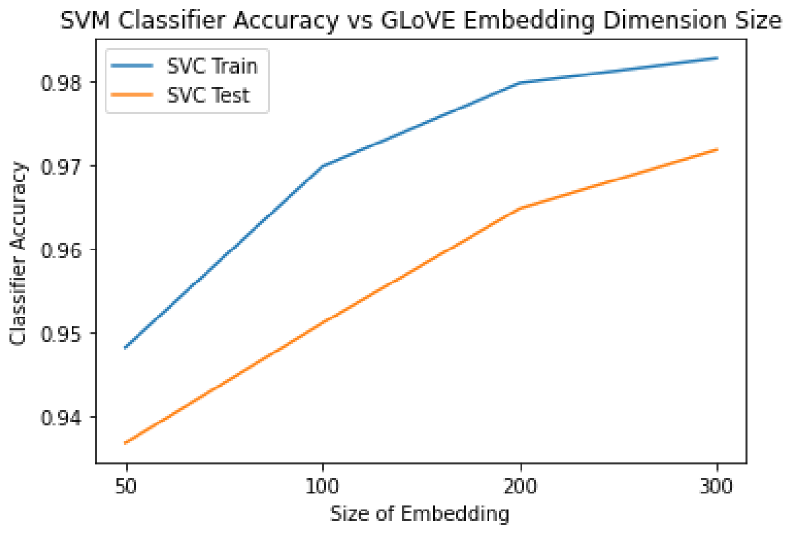

Figure 19: SVM Classifier (C=1) Train and Test Accuracy vs GLoVE Embedding Dimension Size

In Fig. 19 we can see the performance of our classifier as a function of the GLoVE Embedding Dimension size. As the size of the embedding dimension increases, the accuracies improves. This is expected as a larger dimension size means that the embedding would contain more information that would allow the classifier to make better predictions.

QUESTION 11:

In Fig. 20, we plot a projection of normalized GLoVE embeddings onto a 2D plane and observe two distinct clusters. These clusters correspond to the labels used in the original binary classification problem. The formation of these clusters indicate that the information content of the GLoVE embeddings can be used to meaningfully distinguish different classes of text. In Fig. 21, we plot a projection of normalized random vectors onto a 2D plane and observe no clusters. This is expected because the vectors and random and should have no correlation to any of the labels.

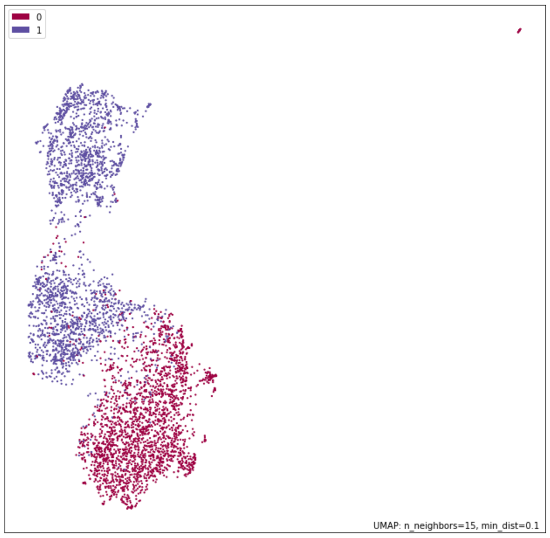

Figure 20: Projection of GLoVE Embeddings

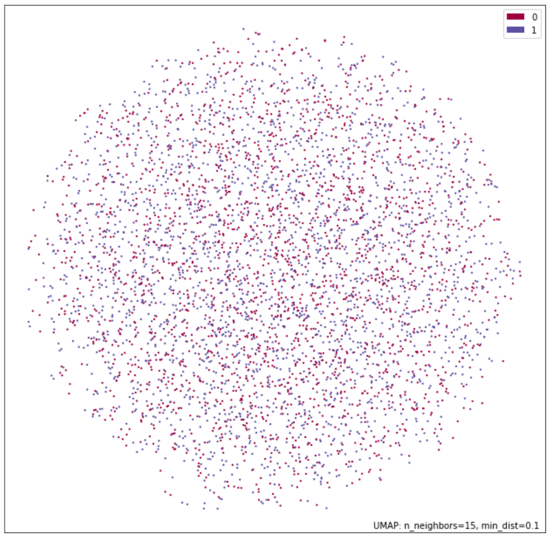

Figure 21: Projection of Normalized random Vectors

Multiclass Classification

QUESTION 12:

Naive Bayes theorem as well as Multiclass and One vs One and One vs the Rest SVM were used for classification and Confusion matrices were also built. One vs the rest SVM was found having the best accuracy as well as other parameters such as recall precision and F1 score.

The categories used were: comp.sys.ibm.pc.hardware, comp.sys.mac.hardware, misc.forsale, soc.religion.christian

Table 10: Estimation of Multiclass Classifiers

Classifier	Accuracy	F-1 Score	Recall	Precision
Naïve Bayes classification	0.8076	0.81	0.82	0.81
One vs One SVM classification	0.8830	0.88	0.89	0.88
One vs the rest SVM classification	0.8856	0.89	0.89	0.89

Clearly, One vs the rest performed the best as compared to the others. One vs the rest performs just slightly better than One vs One. This is because in One vs One you have the classification analysis of one category w.r.t. just one another and it doesn't completely account for the fact that during testing phase the model would have to assign one category to the data, one out of all the present categories in the dataset while doing a 1-on-1 analysis. Conversely in One vs the rest, the category in question's classification is evaluated against the presence of the other categories present in the dataset. Hence the resulting model can effectively classify the data and assign the correct category to the data by differentiating against all the present categories thus resulting in higher accuracy than One vs One.

Naive Bayes' classifier performs the least well because this model requires independence of the predictors which isn't the case in most scenarios and that hinders the performance of the classifier[2].

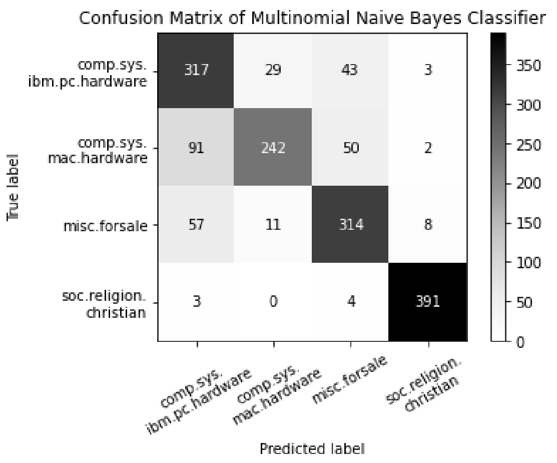

Figure 22: Confusion Matrix of Multinomial Naive Bayes

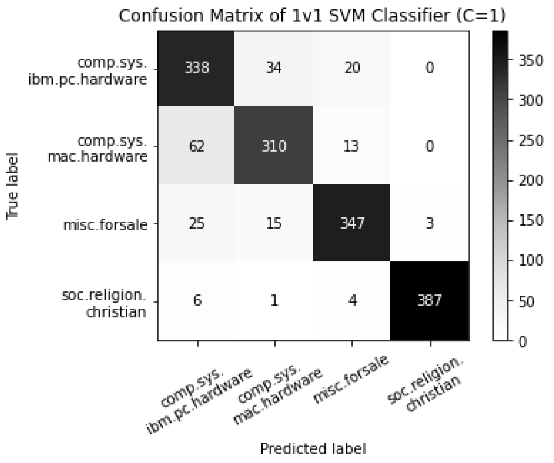

Figure 23: Confusion Matrix of SVM (One vs One)

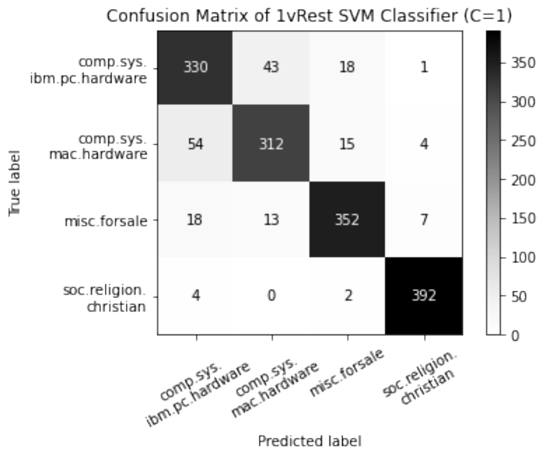

Figure 24: Confusion Matrix of SVM (One vs Rest)

Reference

1. https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html
2. https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c
3. https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a
4. https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c
5. https://towardsdatascience.com/the-game-of-regularization-91442b3be862
6. https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989
7. https://datascience.stackexchange.com/questions/49573/how-to-plot-logistic-regression-decision-boundary
8. https://medium.com/axum-labs/logistic-regression-vs-support-vector-machines-svm-c335610a3d16