ECE 219 Project 1

Tanner Dulay, Loïc Maxwell, Zoë Tucker, Alex Wasdahl

January 20, 2021

1 Question 1

To begin, we need to determine whether the data in our dataset is evenly distributed. To check this, we plot a bar chart showing the number of documents with each available label in the "test" subset.

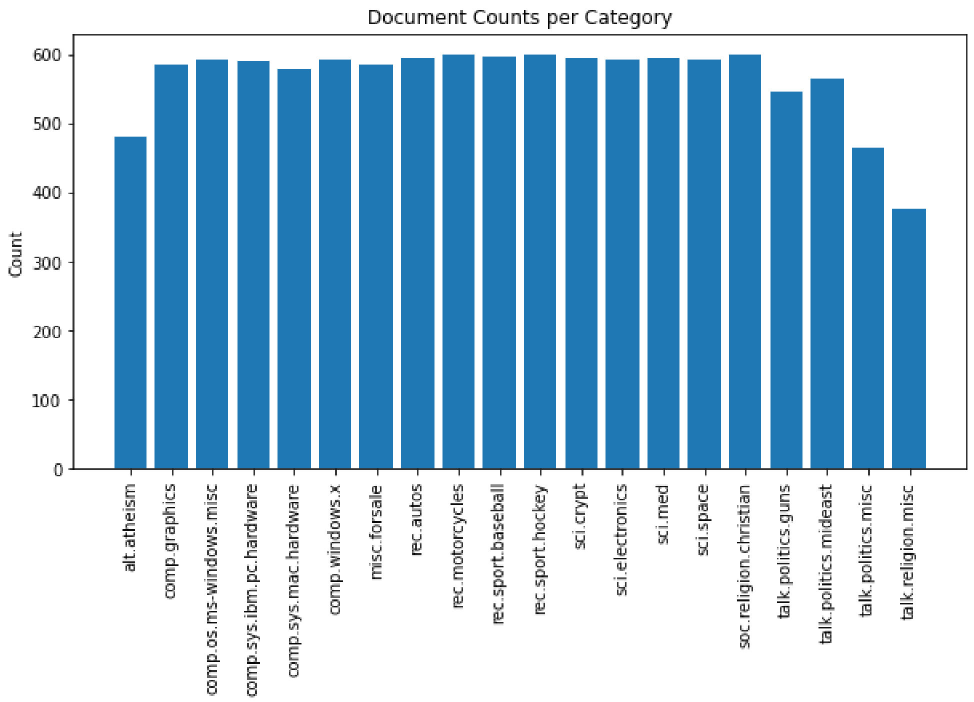

Figure 1: Histogram of the data

As shown in Figure 1, the data is balanced, especially considering only the labels identified as "Computer Technology" or "Recreational Activity" as defined in the project statement.

2 Question 2

Next, we need to make a representation of our dataset to use in our classification algorithms using `CountVectorizer`. Excluding "`english`" stopwords, numerals, and punctuation, lemmatizing with `nltk.wordnet.WordNetLemmatizer` and `pos_tag`, and using `min_df` = 3, we create TF-IDF matrices for the test and train data subsets.

The resulting training data matrix is 4732×16466, and the test data matrix is 3150×16466.

3 Question 3

Before we apply classification algorithms, we reduce the dimensionality of the data to map each document to a 50-dimensional vector. We attempted this with two methods: LSI and NMF.

For LSI, the total (Frobenius) error is 4099.6. For NMF, it is 4141.4. As the error for the LSI-reduced matrices is lower, LSI is the better method for our case and we use the LSI matrices in subsequent questions. The reduced error for LSI may be due to the fact that NMF is forced to use non-negative matrices in its decomposition, limiting the range of potential solutions.

4 Question 4

In this section, we use linear Support Vector Machines (SVMs) as our first classification algorithm. We begin by training a "hard margin" SVM, with tradeoff parameter $\gamma = 1000$ representing a high penalty for individual misclassifications, and a "soft margin" SVM with $\gamma = 0.0001$ representing a higher prioritization of separation of most data points.

The below table summarizes the performance statistics for each of the SVMs, and the confusion matrices and ROC curves are given in Figures 2, 3, 4, and 5.

SVM	Accuracy	Recall	Precision	F-1 Score
Soft Margin	0.6711	0.6679	0.8027	0.6286
Hard Margin	0.9721	0.9720	0.9722	0.9721

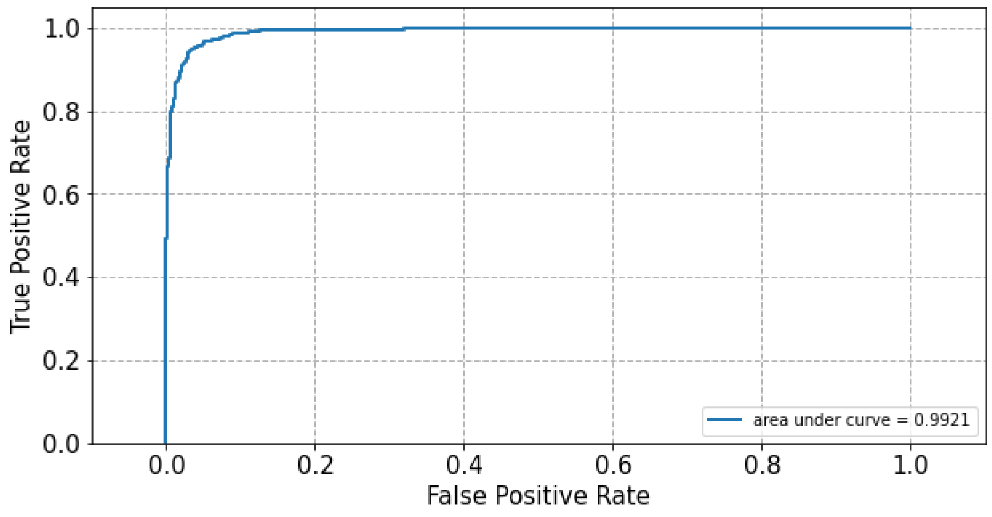

Figure 2: ROC for Soft Margin SVM

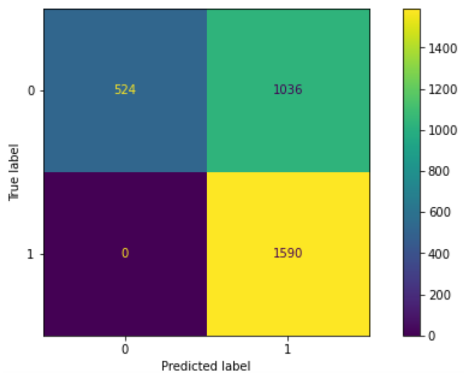

Figure 3: Confusion Matrix for Soft Margin SVM

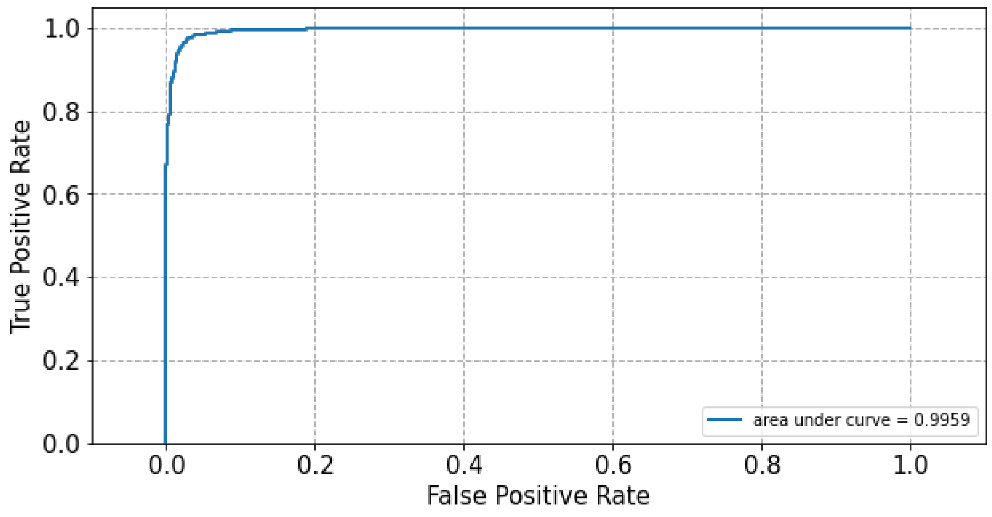

Figure 4: ROC for Hard Margin SVM

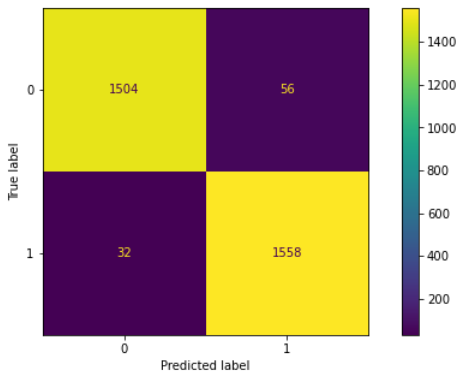

Figure 5: Confusion Matrix for Hard Margin SVM

From these statistics, especially the vastly improved accuracy of .97 over .67, we conclude that the hard margin SVM performs much better.

Looking more closely at the soft margin results, we see that the ROC curve is not very different than the hard margin ROC, despite the very different accuracy and other metrics. This indicates that the model has chosen a poor threshold for separating the data.

Next, we want to find the best possible SVM. To do this, we use cross-validation in testing to find the optimal value for γ. In particular, we use 5-fold cross-validation and look for a $\gamma \in \{10^k | -3 \leq k \leq 3, k \in \mathbb{Z}\}$.

Testing each parameter, we find that the best value is $\gamma = 1$, which gave an average accuracy of 0.9727 during our cross-validation trial. We then use this value to train a new SVM with the full training dataset, which gives the scores in the below table and in Figures 6 and 7.

SVM	Accuracy	Recall	Precision	F-1 Score
$\gamma = 1$	0.9727	0.9726	0.9730	0.9727

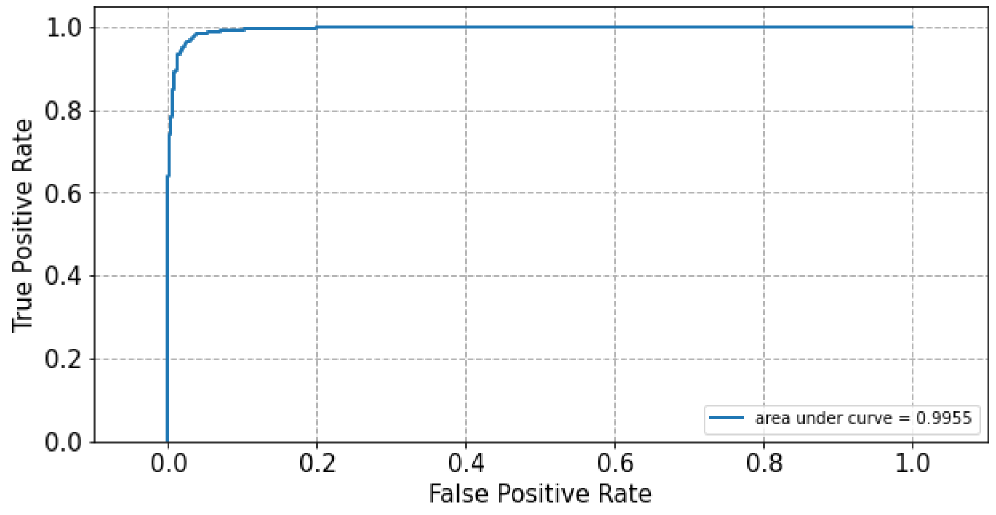

Figure 6: ROC for Optimal SVM

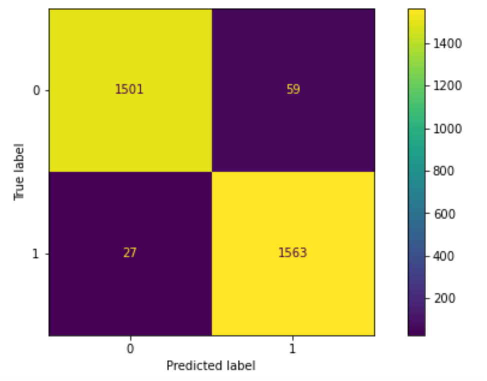

Figure 7: Confusion Matrix for Optimal SVM

5 Question 5

Next, we analyze logistic classifiers for this task using the same methods as in section 4. We begin with a non-regularized classifier. Since we are using sklearn's implementation, we set C to a very large number to approximate an unregularized classifier. The statistics for this classifier are found in the table below, and the ROC and confusion matrix are shown in Figures 8 and 9.

Log. Classifier	Accuracy	Recall	Precision	F-1 Score
Unregularized	0.9717	0.9635	0.9647	0.9717

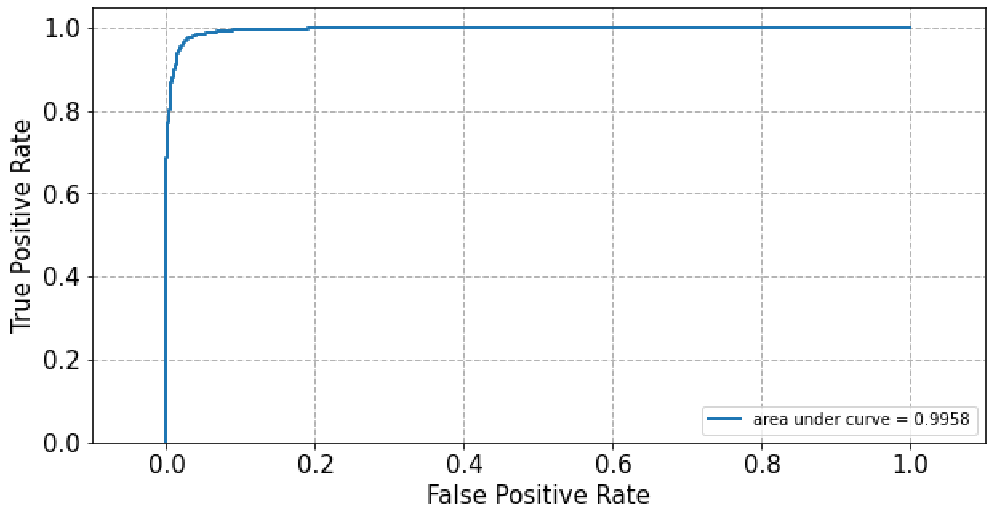

Figure 8: ROC for Unregularized Logistic Regression Classifier

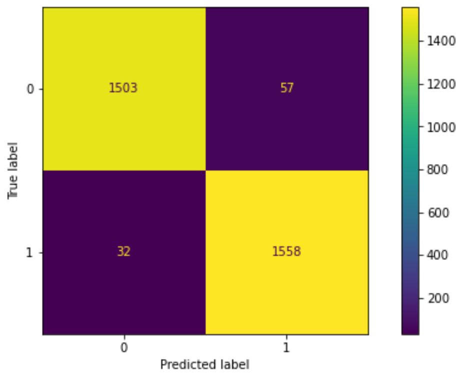

Figure 9: Confusion Matrix for Unregularized Logistic Regression Classifier

As before, we use five-fold cross-validation to find the optimal regularization parameter C. However, we now have two options for regularization type: L1 and L2. Thus, we must run the cross-validation twice, once for each type.

We find that for both regularization types, the optimal value of C is 10. The performance metrics for these optimal classifiers are given in the tabe below, and the ROC and confusion matrix for each are shown in Figures 10 - 13.

Log. Classifier	Accuracy	Recall	Precision	F-1 Score
L1, $C = 10$	0.9717	0.9717	0.9721	0.9717
L2, $C = 10$	0.9717	0.9716	0.9720	0.9717

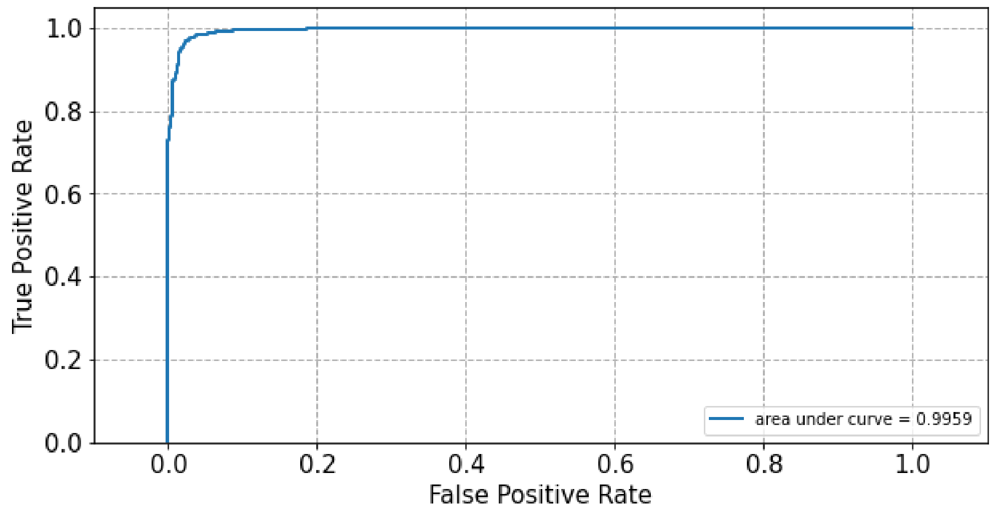

Figure 10: ROC for L1 Logistic Regression Classifier, $C = 10$

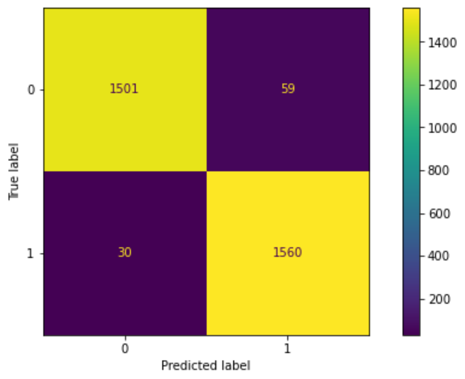

Figure 11: Confusion Matrix for L1 Logistic Regression Classifier, $C = 10$

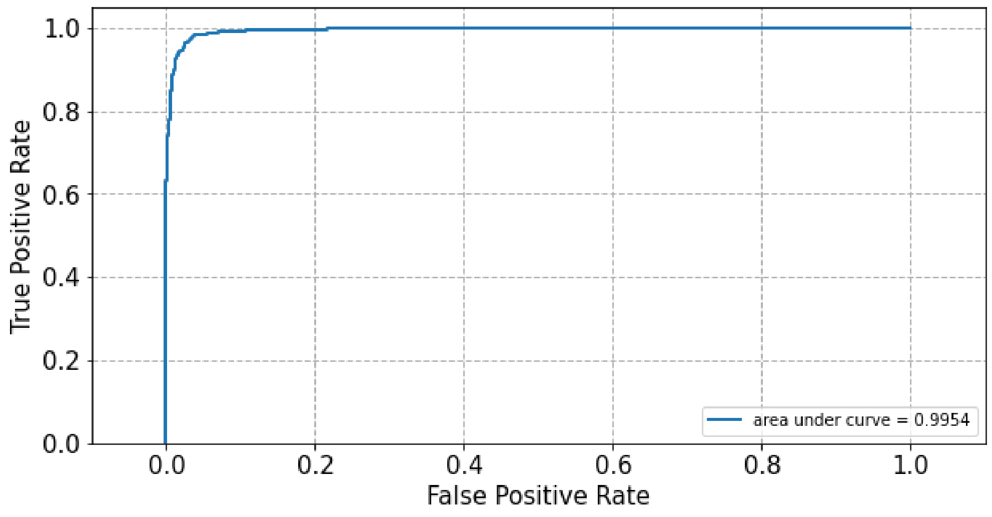

Figure 12: ROC for L2 Logistic Regression Classifier, $C = 100$

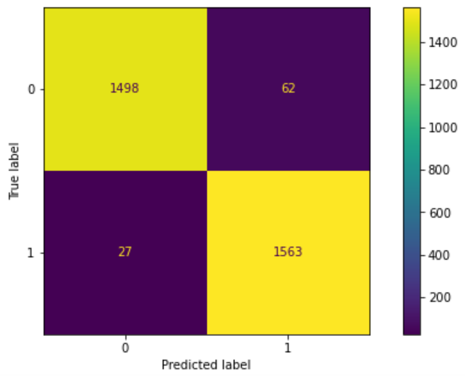

Figure 13: Confusion Matrix for L2 Logistic Regression Classifier, $C = 100$

As we can see, the logistic regression classifier with L1 regularization was slightly better than the others when comparing recall and precision across the models. The unregularized classifier had the worst performance.

Looking specifically at the learned coefficients, we see that they are fairly large for the unregularized classifier, with an average absolute value of around 13. This may suggest that the model is overfitted to the training data.

While L2 regularization is computationally easier and produces good results often, L1 regularization is useful when the feature space is not too large.

In comparison to the previous section, we see that the logistic regression classifiers do not perform as well as the optimal SVM classifier. This is likely due to the ways that the two models attempt to create a decision boundary. SVMs learn a feature vector by looking at all of the training data points and minimizing a loss function with regularization. In contrast, logistic regression takes a probabilistic approach by iteratively attempting to find the optimal decision boundary. For structured data like our text documents in this problem, SVMs have better performance.

6 Question 6

Finally, we train a Naïve Bayes classifier on our dataset and find the same metrics, demonstrating worse performance than either of the two previous models. The below table summarizes the information, and the ROC curve and confusion matrix are shown in Figures 14 and 15.

Classifier	Accuracy	Recall	Precision	F-1 Score
Naïve Bayes	0.89	0.89	0.90	0.89

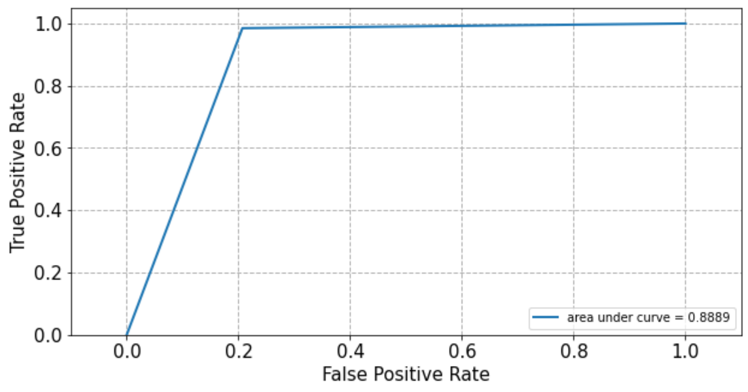

Figure 14: ROC for Naïve Bayes Classifier

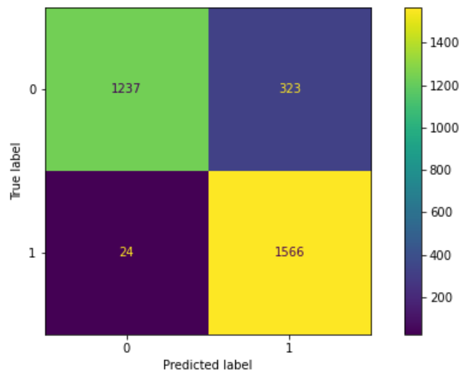

Figure 15: Confusion Matrix for Naïve Bayes Classifier

7 Question 7

Now that we have encountered each of these classifiers individually, we perform a grid search to find the optimal combination of parameters. We compare differences in data loading (removing or leaving "headers" and "footers"), feature extraction (`min_df` = 3 or 5, using lemmatization or not), dimensionality reduction (NMF or LSI), and classifier type (SVM, Logistic Regression, or Naïve Bayes with previously found optimal parameters). Two sets of training/test data were used: one with headers/footers and another without. `GridSearchCV` was run twice to tune the (hyper)parameters for each set of data, and the grid search that had the best performance on the test set was selected. The results are shown below:

Dataset Type	Lemmatization	min_df Value	LSI/NMF	Classifier	Test Accuracy
With Headers/Footers	Yes	3	LSI	Logistic Regression w/ L1 reg and C=10	0.9721
Without Headers/Footers	Yes	3	LSI	Logistic Regression w/ L1 reg and C=10	0.9670

As can be seen in the table, the best model parameters were actually the same for both data types. However, because of its better performance on the test set, the best combination includes data with headers and footers. This is why it's highlighted in the table.

8 Question 8

Next, we consider GLoVE embeddings of text data. GLoVe embeddings are trained on the ratio of co-occurrence probabilities rather than the probabilities themselves because the ratio of co-occurrence probabilities produces more meaning. The probabilities themselves can be high or low depending on the contextual relationship of the words in question, but the ratios produce numbers that cancel out much of the "noise" from probe words that either relate to both words of interest or neither. Using the example from the paper, the co-occurrence probabilities for the words of interest i = ice and j = steam differ based on the probe word k used. In the case of a word like "solid" that correlates strongly with ice but poorly with steam, the ratio provides a number much greater than one that gives meaning to the relationship between the two probabilities. In contrast, for a probe word like "water," the probabilities themselves are high for both correlations water—ice and water—steam, but the ratio is close to 1, which provides more contextual meaning about the relative relationship of the probe word water to both ice and steam. Similarly, a probe word like "fashion" which has low probabilities for both correlations fashion—ice and fashion—steam will also have a ratio close to 1. The probabilities are different but the ratios add more meaning regarding the relevance or irrelevance of the probe words.

For the sentences "James is **running** in the park" and "James is **running** for the presidency", GLoVE will **not** return the same vector for the word "running". This is because GloVe incorporates global statistics (word co-occurrence) to obtain word vectors rather than just relying on local statistics (local context information of words).

For all of the expressions

$$||\texttt{GLoVE["queen"] - GLoVE["king"] - GLoVE["wife"] - GLoVE["husband"]}||_2,$$

$$||\texttt{GLoVE["queen"] - GLoVE["king"]}||_2,$$

$$||\texttt{GLoVE["wife"] - GLoVE["husband"]}||_2,$$

we expect a value of zero.

Given a word, lemmatization is preferable to stemming before mapping to its GLoVE embedding. Stemming operates on a single word without knowledge of the context, and therefore cannot discriminate between words which have different meanings depending on part of speech. Lemmatization considers the roles of the words in the sentence.

9 Question 9

Our feature engineering process that leverages GLoVE embedding is as follows. First, for each document, we grab all characters that follow the strings: "Subject:" and "Keywords:" up until the next newline character and store them in a new string. We also remove the strings "Re:" and "Fwd:" if they can be found in this string, since they are common terms (found in the subject line) that don't give us any information about the document's class. We then tokenize the string and lemmatize each of the words. For each of the words, if an embedding exists in the GLoVE embeddings dictionary, we grab the corresponding vector and normalize it. We then average all of these vectors to form a single vector that represents the entire document. For the documents where none of the lemmatized words have corresponding embeddings, we decided to not include them in the dataset to be fed into the classifier. There were about 50 cases of this per dataset, which is still small compared to the number of total documents in each dataset.

For our classifier, we picked the same best parameters from Q7 (Linear Regression with L1 Regularization, C=10) and got a test accuracy of 0.9160, which is a good result. This isn't as good as our previous pipeline, but with more hyperparameter tuning we can expect some better results.

10 Question 10

Next, we plot the relationship between the dimension of GLoVE embeddings and the accuracy of our model. The results are shown in Figure 16.

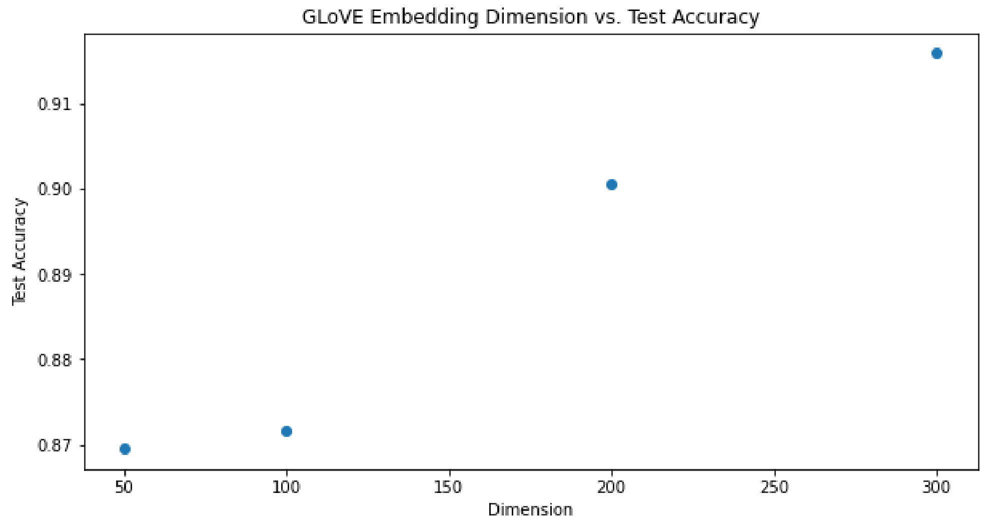

Figure 16: Dimensionality of GLoVE embeddings vs. accuracy of classifiers

Overall, the higher dimensional embeddings gave better results. This is as expected, as it is in line with the results in the original paper. The original authors reported diminishing returns after the dimension exceeded 200, which can be seen in our plot. By having a higher dimensional embedding, we are able to capture more defined literal/contextual information about a word, which allows us to better differentiate between text data belonging to one of two classes. However, if the dimension is too large, then we're "stretching" a word too thin; each feature in the vector becomes less representative of the word, making it more difficult to train (and easier to overfit) and resulting in a lower performance overall. In our case, we got better results with 300 dimensions than with 200 dimensions, but if we keep increasing this number than we would expect our test accuracy to decrease.

11 Question 11

Using UMAP, we project our GLoVE embeddings for the test dataset into the 2D plane, as shown in Figure 17. For comparison, we also generated a random set of normalized 300-dimensional vectors and projected them using the same UMAP methodology (Figure 18).

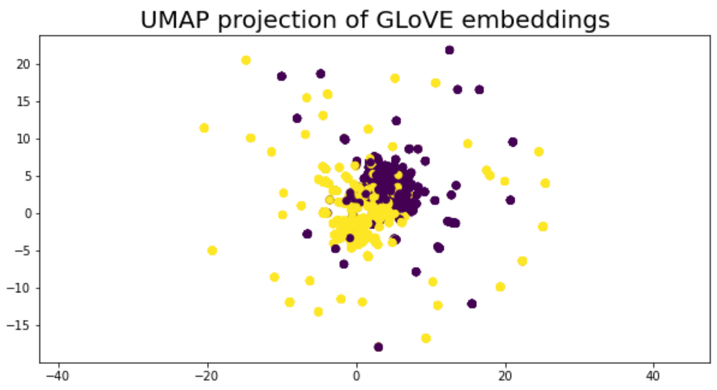

Figure 17: GLoVE Embeddings: Purple is “Computer Technology”, Yellow is “Recreational Activity”

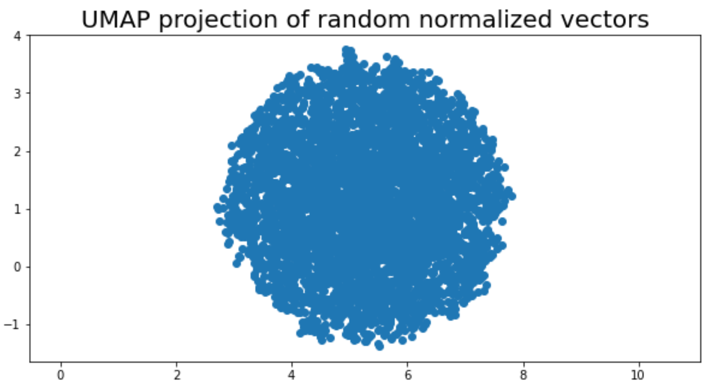

Figure 18: Random projected vectors

In both plots, we see that the points are largely clustered around the origin. However, the clustering on the random vectors is much more uniform, and there are many more “outliers” (points further from the origin - note the different axis scales between the two figures) in the GLoVE plot. In addition, we can see that the data is roughly partitioned, with the purple "Computer Technology" points falling roughly to the upper-right of the center, and the yellow "Recreational Activity" points to the lower-left. This indicates that meaningful information is contained in the GLoVE embeddings, while the random vectors contain no information.

12 Question 12

Finally, we perform multiclass classification on our data. We test Naïve Bayes, One vs. One SVM, and One vs. Rest SVM classifiers to classify data with labels `comp.sys.ibm.pc.hardware`,`comp.sys.mac.hardware`,`misc.forsale`, and `soc.religion.christian`. The below table summarizes the performance of the classifiers, and Figure 19 shows the confusion matrix for each classifier. Comparing accuracy scores for each method, we see that the One vs. Rest SVM has the best performance, and the Naïve Bayes classifier has the worst.

Classifier	Accuracy	Recall	Precision	F-1 Score
Naïve Bayes	0.69	0.69	0.70	0.68
One vs. One SVM	0.8652	0.8652	0.8663	0.8656
One vs. Rest SVM	0.8773	0.8773	0.8775	0.8773

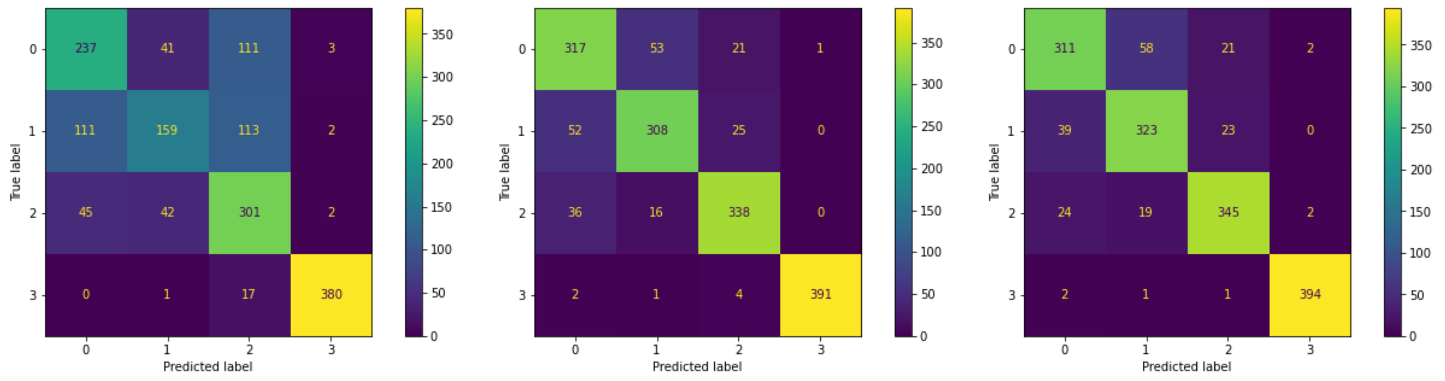

Figure 19: Confusion matrices for multiclass classifiers: Naïve Bayes (Left), One Vs. One SVM (Center), and One Vs. Rest SVM (Right)

As with our binary-classed data, the Naïve Bayes classifier was the least accurate for multiclass data. However, we found all of our classifiers to be less accurate overall when applied to multiclass data. This is likely because our classifiers are now over-fitting the training data. When we divide our dataset to more classes, the sample size for each class is reduced, leading the classifier to do a poorer job of generalizing so that it can correctly classify novel data.