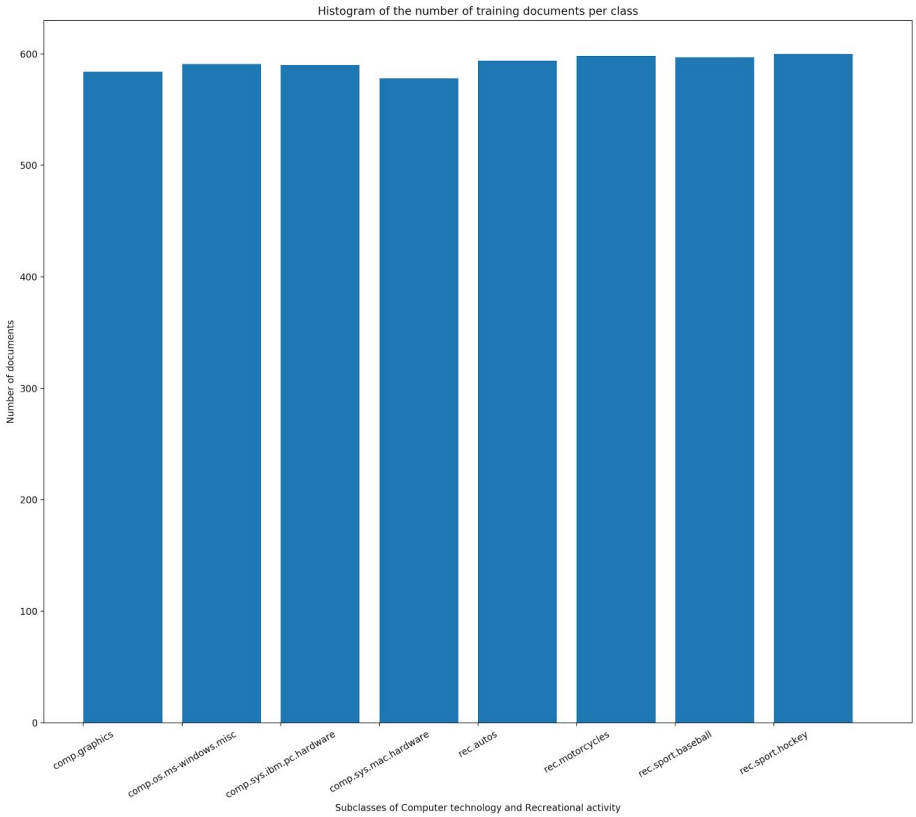
Classification Analysis on Textual Data Report

Abdullah-Al-Zubaer Imran
Curtis Crawford

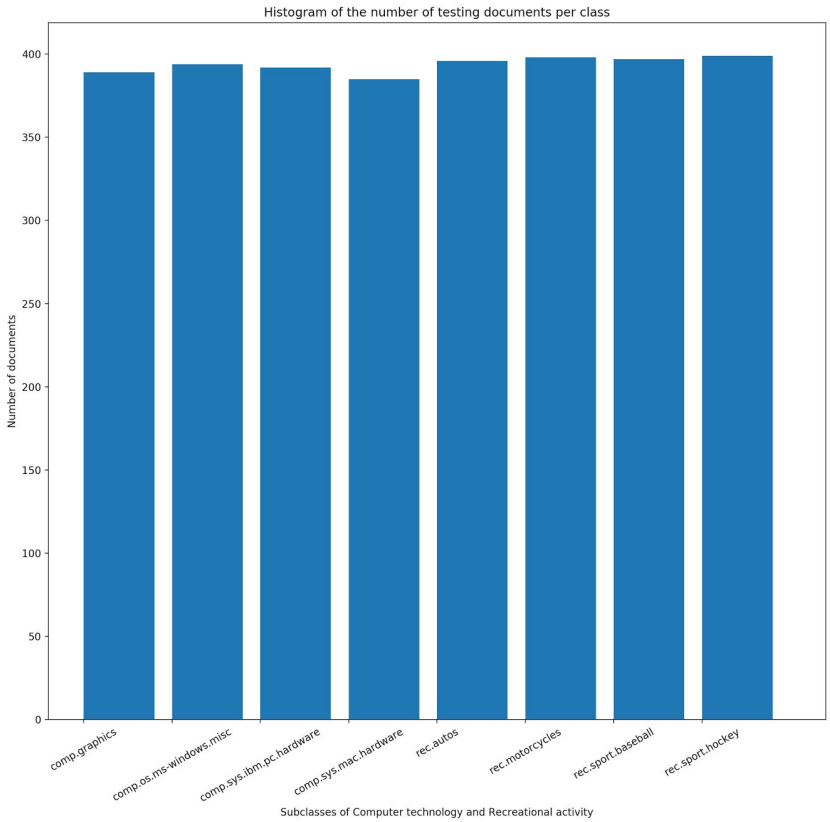
Parts A,B,D

These portions of the project focused on preparing the documents in the 20 Newsgroups dataset for training and testing classifiers. Using the built-in functions of the sklearn python module, these tasks were very easy to implement.

The total number of documents in the each of the 8 sub-classes from the Computer technology and Recreational activity classes contained about 1000 documents in total, split between 600 in a training set and 400 in a testing set. Sklearn provides functions for loading the dataset quickly. Below is a figure showing the even distribution of documents in the training set:



And also the even distribution of documents in the test set.



To vectorize the documents, the standard sklearn CountVectorizer class was extended to also stem words so that words such as “run” and “running” are counted as the same word. The Porter stemming algorithm was used. After vectorization and transformation to a TFxIDF matrix with another sklearn built in function, the dimensions of the feature vector as as follows for different values of the min_df parameter.

min_df	TFxIDF dimensions (documents x terms)
2	4732 x 25335
5	4732 x 10691

Clearly, the higher min_df factor removed about ⅔ of the terms that using a min_df factor used. Using Latent Semantic Indexing (LSI), the TFxIDF can be significantly reduced in dimensionality in order to improve classification tasks. LSI was done using the sklearn provided TruncatedSVD processing function, with a target of 50, such that the dimensions after LSI were 4732 documents x 50 terms.

Part C

To determine the most important terms for the *comp.sys.ibm.pc.hardware, comp.sys.mac.hardware, misc.forsale,* and *soc.religion.christian* sub classes, first all of the documents of a specific class were merged together, so there were 20 documents total where each one contained all the documents from that class. These 20 documents were then vectorized and converted to a TFxICF using the process as creating a TFxIDF. The 10 most important terms for each of these sub-classes is as follows, for a min_df of 2:

comp.sys.ibm.pc.hardware: 0th most common item is: scsi 1th most common item is: drive 2th most common item is: edu 3th most common item is: ide 4th most common item is: use 5th most common item is: line 6th most common item is: com 7th most common item is: subject 8th most common item is: organ 9th most common item is: card	comp.sys.mac.hardware: 0th most common item is: edu 1th most common item is: mac 2th most common item is: line 3th most common item is: subject 4th most common item is: organ 5th most common item is: quadra 6th most common item is: use 7th most common item is: appl 8th most common item is: simm 9th most common item is: scsi
misc.forsale: 0th most common item is: edu 1th most common item is: 00 2th most common item is: line 3th most common item is: sale 4th most common item is: subject 5th most common item is: organ 6th most common item is: post 7th most common item is: new 8th most common item is: com 9th most common item is: univers	soc.religion.christian: 0th most common item is: god 1th most common item is: christian 2th most common item is: edu 3th most common item is: jesu 4th most common item is: church 5th most common item is: subject 6th most common item is: peopl 7th most common item is: line 8th most common item is: say 9th most common item is: christ

Notably the stemmer made some made some interesting choices, such as shortening “jesus” to “jesu.” However, this likely would not affect classification as any document run through the classifier would also be stemmed.

Parts E, F, G, H, I

First, the documents were re-classified as either a Computer or a Recreational document, based on the high-level class of the sub-class they belonged to. This was done as these sections work with binary classifiers.

For parts E and F, the sklearn SVC classifier class was used to train the classifier and make predictions. Using a linear kernel and various values of γ to investigate its impact. The value of min_df between 2 and 5 had little impact on classifier performance

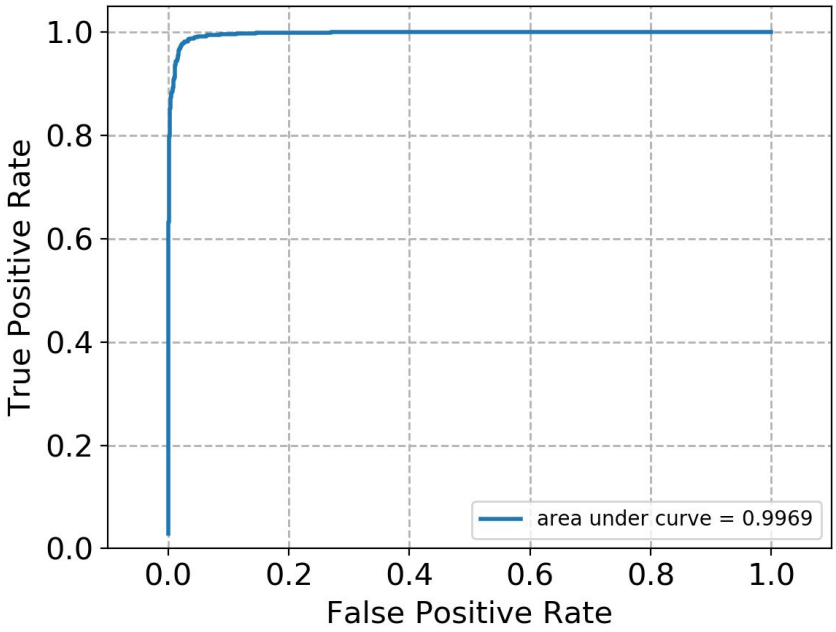
Hard-Margin vs Soft-Margin SVC

While for both min_df=2 and min_df=5 Hard-margin SVC with a linear kernel lead to approximately 97% accuracy, soft-margin only achieved a 50% accuracy score. Thus, with γ=0.001, the SVC was no better than a flip of a coin at determining if an article was computer focused or recreation focused. Below are the ROC curves and a table relating the accuracy, precision, recall, and confusion matrix for these classifiers.

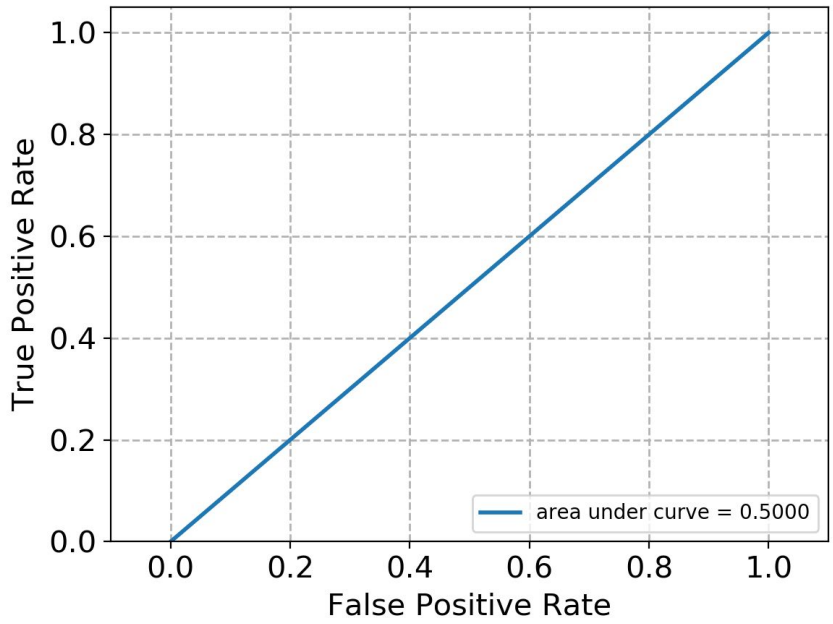
Value of γ	min_df	Accuracy	Precision	Recall	Confusion Matrix
1000	2	0.975	0.968	0.982	1508 | 52 28 | 1562
1000	5	0.973	0.970	0.977	1512 | 48 37 | 1553
0.001	2	0.505	0.505	1.0	0 | 1560 0 | 1590
0.001	5	0.505	0.505	1.0	0 | 1560 0 | 1590

No practical difference in ROC curves was noticed so only the curves for min_df=2 are included below:

Heavy Margin:



Soft Margin:



SVC Cross Validation

Through 5-fold Cross validation for different values of γ, we attempted to determine a value that was suitable for classification but not overly aggressive. We determined that γ=0.1 is the minimum value to use with this data set and the LSI dimension reduction that we performed. Below is a table detailing the accuracy, recall, precision, and confusion matrix of predicted labels after 5-fold cross validation. Cross validation was done using the sklearn provided function that performs cross validation and predicts labels for the dataset, cross_val_predict().

Value of k	Value of γ	Accuracy	Precision	Recall	Confusion Matrix
-3	0.001	0.505	.0505	1.0	0 2343 0 2386
-2	0.01	0.505	.505	1.0	1 2342 0 2389
-1	0.1	0.969	.0964	.0974	2255 69 63 2326
0	1	0.972	0.971	0.964	2274 56 63 2326
1	10	0.974	0.976	0.973	2287 48 63 2326

Value of k	Value of γ	Accuracy	Precision	Recall	Confusion Matrix
2	100	0.977	0.976	0.977	2286 57 54 2335
3	1000	0.976	0.977	0.975	2289 54 60 2329

Naive Bayes Classifier

Using the Naive Bayes Classifier required that instead of performing dimension reduction on the TFxIDF matrix using LSI we use Non-Negative Matrix Factorization (NMF). This is because LSI can result in negative terms in the reduced matrix, which are not valid inputs for the Naive Bayes algorithm. Sklearn provides an implementation of NMF so we were able to slightly modify our pipeline to perform this classification, as well as a Multinomial Naive Bayes Classifier implementaiton.

Logistic Regression

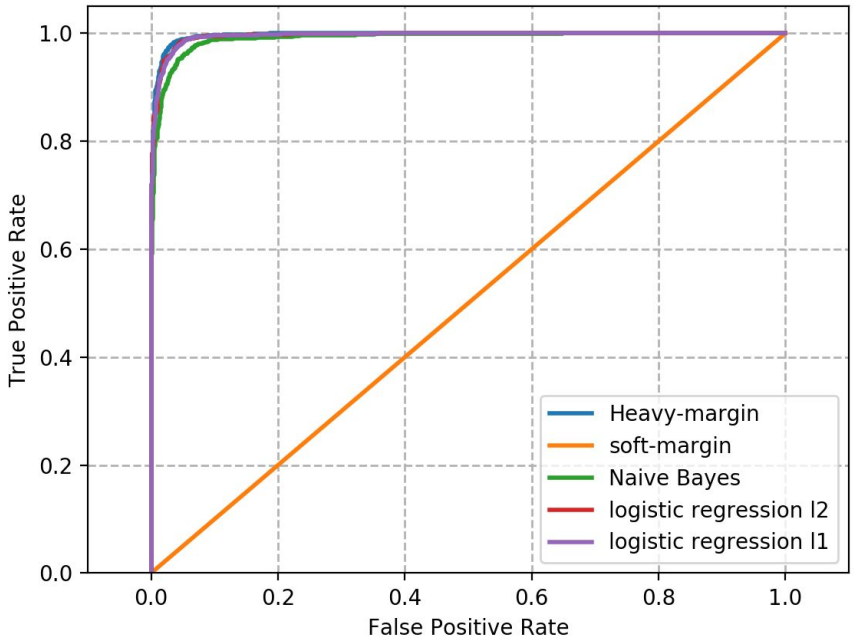
As logistic regression was performed on the TFxIDF matrix reduced using LSI, and was also done using sklearn implementations of the logistic regression algorithms.

Comparison of Logistic Regression, Naive Bayes, and SVC

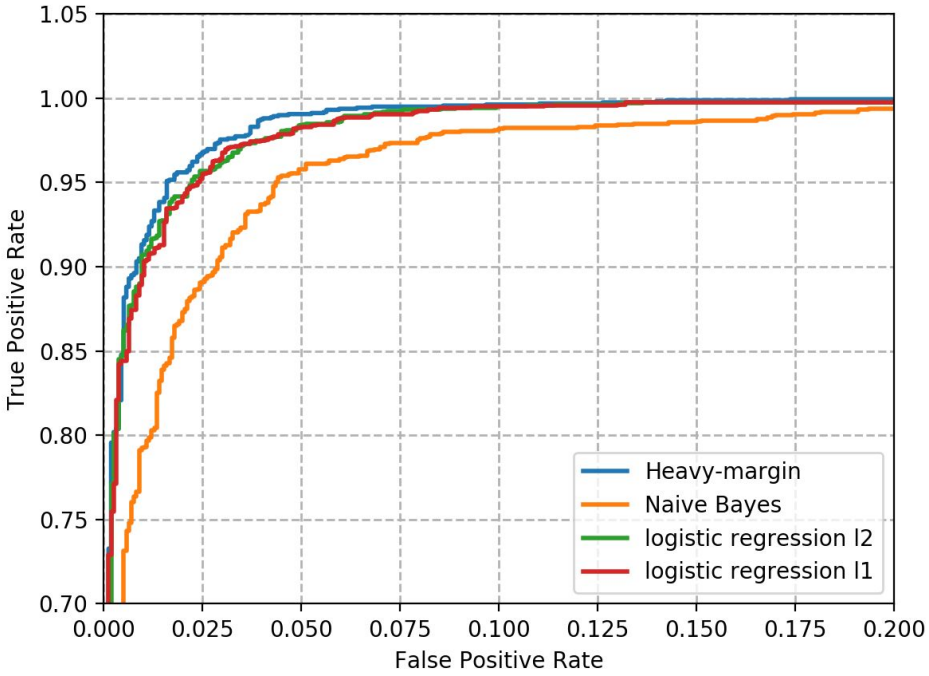
Below is a table comparing the results of using various Logistic Regression classifiers, the Naive Bayes classifier, and a Heavy-margin (γ=1000) SVC classifier. Following this is a plot showing the ROC curves of these methods, as well as a detailed view of the minute differences in their behavior.

Classifier	Accuracy	Precision	Recall	Confusion Matrix
Naive Bayes	0.930	0.891	0.983	1367 | 192 27 | 1563
Logistic Regression “l2”	0.967	0.958	0.977	1492 | 68 37 | 1553
Logistic Regression “l1”	0.966	0.956	0.977	1489 | 71 37 | 1553
Linear SVC Heavy-margin	0.975	0.968	0.982	1508 | 52 28 | 1562

ROC Curves including the Soft-margin results. All are nearly as good, with Naive Bayes slightly under performing the others.



Close-up of the curve to show their slightly different plots:



Multi-class classification

We performed multi-class classification on a dataset containing just documents from the four classes called out in the problem statement. Thus, the number of targets in our dataset was 4. Sklearn implements two different types of multiclass SVM, one that uses one-versus-one classification and one that uses one-versus-rest classification. Both can be done using the linear kernel. This means that all our classification tasks for these three different approaches, with 11 total classifiers if done by hand, can be implemented in only three sklearn module functions. The results are below, with the precision and recall given for each class individually

Naive Bayes

Accuracy: 0.803
Precision & Recall:

Class	Recall	Precision
comp.sys.ibm.pc.hardware	0.793	0.663
comp.sys.mac.hardware	0.566	0.845
misc.forsale	0.854	0.789
soc.religion.christian	0.990	0.947

Confusion Matrix:

```
[[311  35  39   7]
 [112 218  46   9]
 [ 46   5 333   6]
 [  0   0   4 394]]
```

One-vs-One Linear SVM

Accuracy: 0.875
Precision & Recall:

Class	Recall	Precision
comp.sys.ibm.pc.hardware	0.850	0.782
comp.sys.mac.hardware	0.795	0.845
misc.forsale	0.895	0.886
soc.religion.christian	0.957	0.995

Confusion Matrix:

```
[[333  41  18   0]
 [ 55 306  23   1]
 [ 26  14 349   1]
 [ 12   1   4 381]]
```

One-vs-Rest Linear SVM

Accuracy: 0.888

Precision & Recall:

Class	Recall	Precision
comp.sys.ibm.pc.hardware	0.821	0.850
comp.sys.mac.hardware	0.847	0.842
misc.forsale	0.903	0.867
soc.religion.christian	0.978	0.992

Confusion Matrix:

```
[[322  45  25   0]
 [ 32 326  26   1]
 [ 21  15 352   2]
 [  4   1   3 390]]
```

Multi-class conclusion

Clearly, all the classifiers were able to classify documents from the soc.religion.christian sub-class very well. However, the recall and precision of the two computer classes were lower, as these documents are likely more similar to each other than they are different from the other two classes. Overall, both the one-vs-one and one-vs-rest SVM approaches out performed using the Naive Bayes classifier, but these both required more computational work to complete and were about equivalent in performance compared to one another.