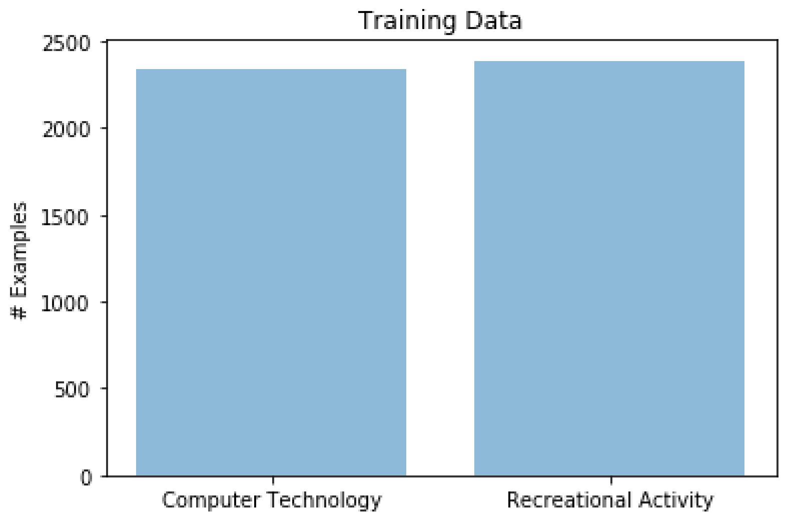
Peter Kim (204271299)
Jonathan Hurwitz (804258351)

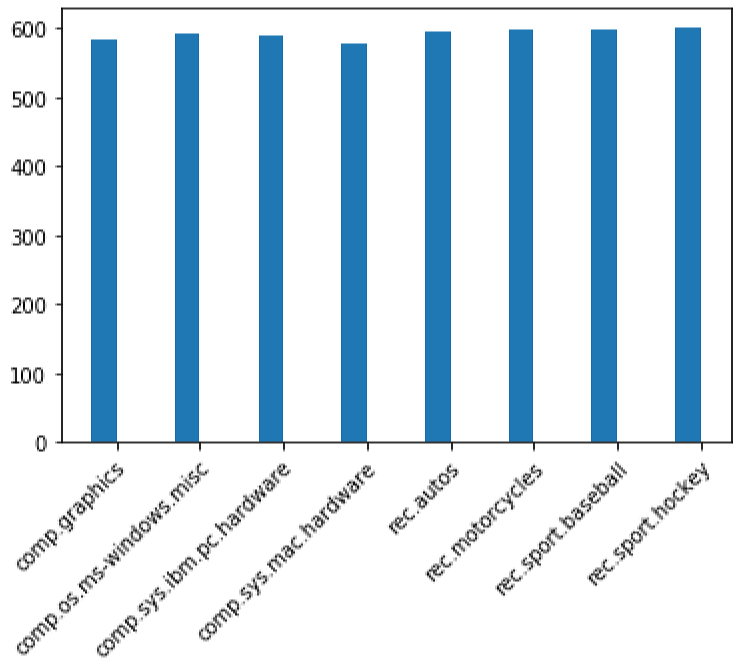
Project 1 Report

All of the code and corresponding figures were generated in a jupyter notebook titled "text_classifier.ipynb". Please refer to this notebook for the code.

a) It's extremely important to visualize and understand data before performing analysis and applying learning algorithms. The following histograms show the distributions of number of examples per two major classes, and per 8 subclasses, respectively. The number of training documents per class are evenly distributed.



Imbalanced data in a classification task is called the "class imbalance problem." One method of dealing with this is to change the cost function used. Another option is to use sampling approaches. Oversampling is when we add more to the minority class. Undersampling removes some of the minority class so it has less of an effect. Hybrid approaches mix this.



b)
In order to vectorize the data, we needed to tokenize each document into words. We wrote our own tokenizer class that uses a Porter stemmer[1] to perform stemming. We excluded stop words from "text.ENGLISH_STOP_WORDS". We also removed punctuation. Finally, we created TFxIDF vector representations by using the CountVectorizer. We fit our training data by calling "fit_transform" and fit our test data by just calling "transform" after the vectorizer had been fit.

The following table summarizes the number of features/terms for the vectorized versions of the data, corresponding to min_df=2 and min_df=5 respectively. Since min_df=5 imposes a stricter filtering criteria, it's expected that the number of terms will be fewer than the results of min_df=2.

Min_df = 2	**Min_df = 5**
25305 terms	10691 terms

c) From the spec, TFxICF is defined as the product of term frequency and inverse class frequency:

$$TFxICF = tf(t,c) * icf(t)$$

[1]http://snowball.tartarus.org/algorithms/porter/stemmer.html

tf(t,c): term frequency of term t in class c
cf(t): class frequency (number of classes that contain t)

icf(t) = log(n_classes/cf(t)) + 1

Class frequency refers to the number of classes in which there is at least a document with term.

The first step was to go through each category, and for each document within it combine into an aggregated “category” doc. Then, the CountVectorizer was used in order to create a vectorized form of the aggregated docs. Max term frequency per class, denoted as max_tf_per_class, and class count per term, denoted as class_count_per_term were calculated for each (class, feature) pair. Then, the tf_icf matrix was populated by calculating the tf_icf for each (term, class) pair, using the equation discussed above.

The ten most significant terms for each category, in the order of 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'misc.forsale', and 'soc.religion.christian' were:

['scsi', 'drive', 'edu', 'thi', 'use', 'line', 'com', 'subject', 'organ', 'ide']
['edu', 'thi', 'line', 'mac', 'subject', 'organ', 'use', 'appl', 'post', 'problem']
['edu', '00', 'line', 'subject', 'sale', 'organ', 'post', 'new', 'thi', 'com']
['thi', 'god', 'wa', 'christian', 'edu', 'hi', 'jesu', 'subject', 'peopl', 'line']

At a glance, these confirm our intuitions. The stemmer removed some suffixes from these words. For example, “apple” became “appl”, and “jesus” became “jesu”. The first category has to do with IBM PC hardware, and SCSI refers to “small computer system interface”. This was a standard for physically connecting computers and transferring data. The next category has the words “mac” and “apple” in them. The religion category has the words “god”, “christian”, and “jesus”.

d)

Before proceeding, we needed to perform feature selection through dimensionality reduction. By default, the TFxIDF vectors are extremely high dimensional and have lots of sparsity. This makes it difficult to do learning, and also makes large-scale problems computationally intractable. Some degree of dimensionality reduction needs to be done on data like this. LSI, a dimensionality reduction method, minimizes mean squared residual between original data and reconstruction from the low dimensional approximation.

We tested classifiers with LSI and NMF separately. In addition, we varied min_df between 2 and 5 per classifier. The feature space was reduced to 50 dimensions, as requested in the specifications.

e)

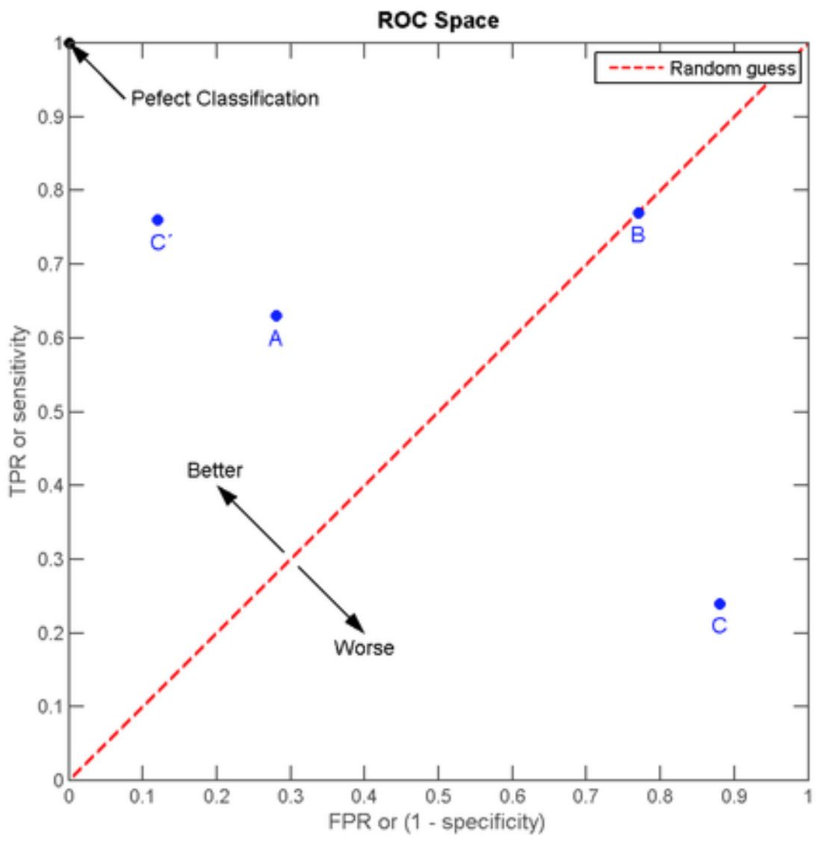
The following table summarizes the F1-score, accuracy, precision, recall, and confusion matrix. F1-score the harmonic mean of precision and recall, and is represented by the following equation:

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\mathrm{recall}} + \frac{1}{\mathrm{precision}}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

F1-score is a reasonable metric to choose for “at-a-glance” evaluation of classifiers. The confusion matrix shows the distribution of true positives, false positives, true negatives, and false negatives. Ideally, we’d like all the classifications to be true positives or true negatives.

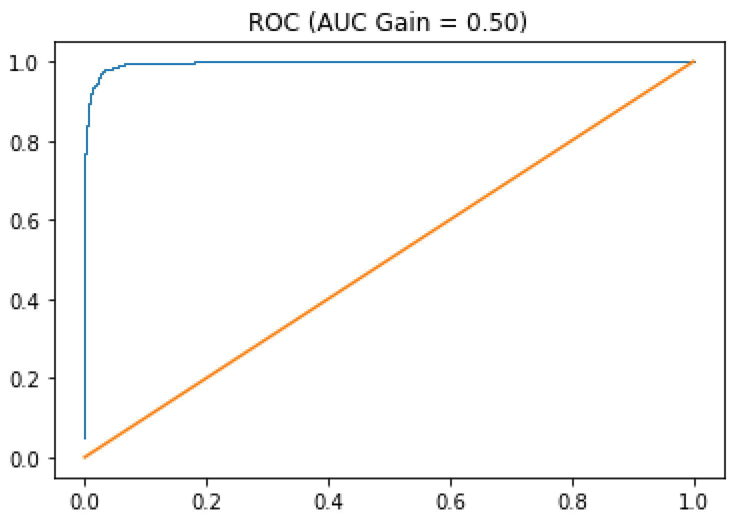
	F1	**Accuracy**	**Precision**	**Recall**	**Confusion Matrix**
Hard SVM (min_df=2, LSI)	0.9717	0.9717	0.9719	0.9716	[[1501 59] [30 1560]]
Hard SVM (min_df=5, LSI)	0.9746	0.9746	0.9747	0.9745	[[1510 50] [30 1560]]
Hard SVM (min_df=2, NMF)	0.9638	0.9638	0.9639	0.9637	[[1491 69] [45 1545]]
Soft SVM (min_df=2, LSI)	0.3354	0.5048	0.2524	0.5	[[0 1560] [0 1590]]
Soft SVM (min_df=5, LSI)	0.3354	0.5048	0.2524	0.5	[[0 1560] [0 1590]]
Soft SVM (min_df=2, NMF)	0.3354	0.5048	0.2524	0.5	[[0 1560] [0 1590]]

A receiver operator characteristic (ROC) curve plots the true positive rate (TPR) against the false positive rate (FPR). ROC is used to help select optimal models and discard suboptimal models. The line of slope 1 with y-intercept of 0 represents the ROC curve of the random classifier. A perfect classification exits in the upper left corner. Better classifications lie above the line and to the left, whereas worse classifications lie below the line, as shown in the figure below:

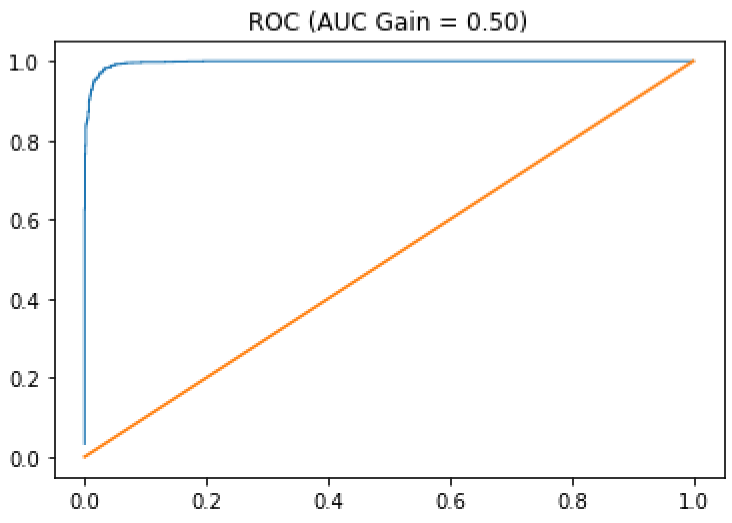


The following ROC curves are associated the six classifiers in the order that they appear in the table above. The hard margin with min_df=2 performed well. We can see this because the classifier's ROC curve is in the upper left corner.

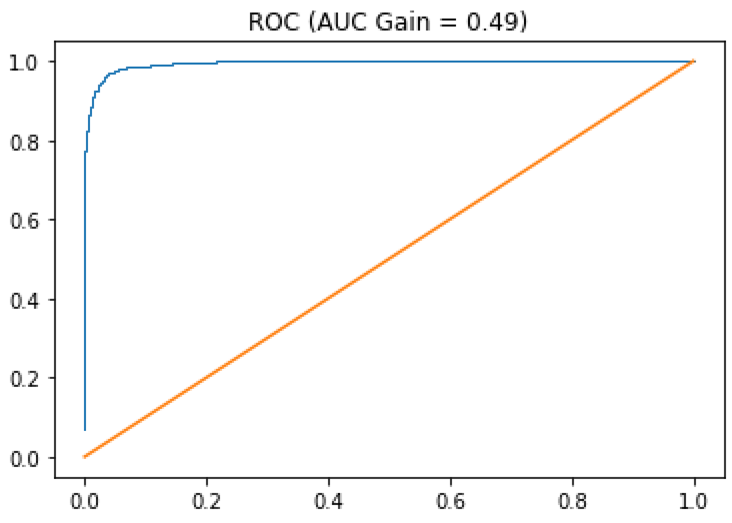
Peter Kim (204271299)
Jonathan Hurwitz (804258351)



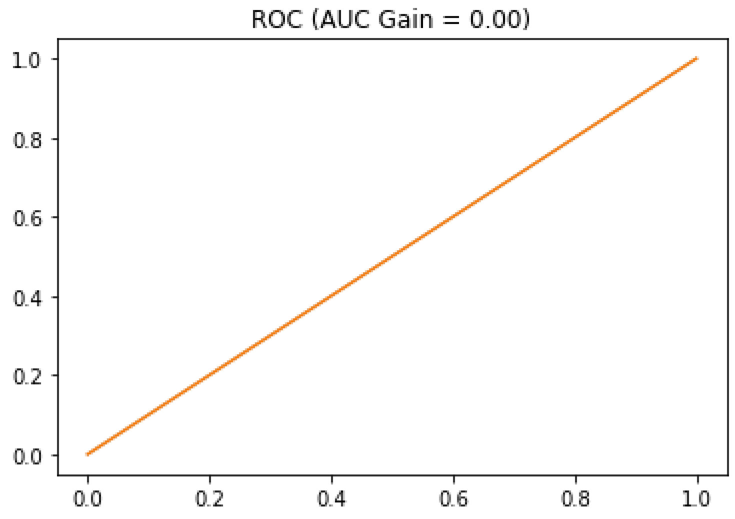
Hard SVM (min_df=2, LSI)



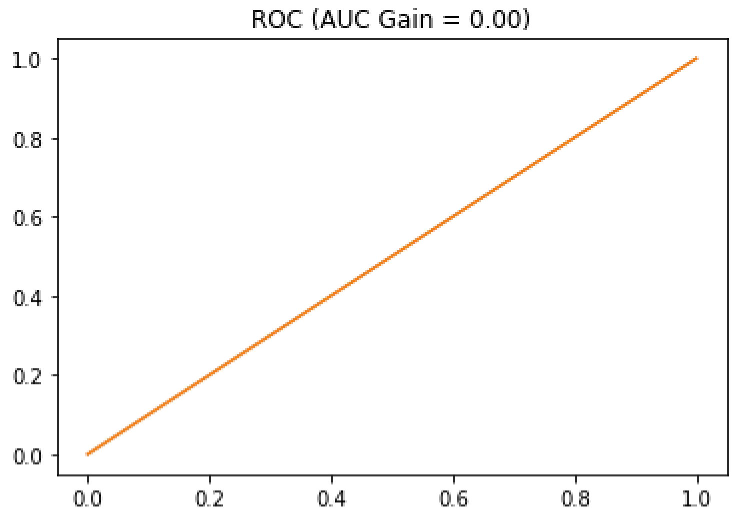
Hard SVM (min_df=5, LSI)



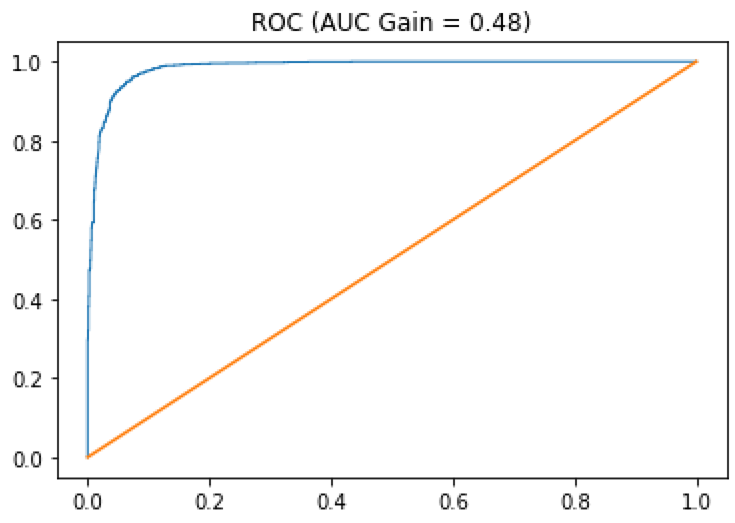
Hard SVM (min_df=2, NMF)



Soft SVM (min_df=2, LSI)



Soft SVM (min_df=5, LSI)



Soft SVM (min_df=2, NMF)

f)

During cross-validation, classifier performance was judged based on accuracy. Other options would have been to select based on F1, recall, or precision. It's also possible to choose based on the mean of all metrics, which would give a balanced classifier. We would have liked to have compared selection methods given more time. Our hypothesis is that the mean of all metrics would give a well-balanced classifier that generalizes well.

Cross-validation results are shown in tabular form below. We included F1-score, accuracy, precision, recall, and the confusion matrix. The best performance was LSI & min_df=2, across all metrics.

Cross-Validation Results for LSI & min_df=2

The best classifier using LSI & min_df=2 was the one with gamma=100.

	F1	**Accuracy**	**Precision**	**Recall**	**Confusion Matrix**
gamma=100	0.973	0.973	0.9732	0.9729	[[1504 56] [29 1561]]

Cross-Validation Results for LSI & min_df=5
The best classifier using LSI & min_df=5 was the one with gamma=10.

	F1	**Accuracy**	**Precision**	**Recall**	**Confusion Matrix**
gamma =10	0.9692	0.9692	0.9697	0.9691	[[1489 71] [26 1564]]

Cross-Validation Results for NMF & min_df=2
The best classifier using NMF & min_df=2 was the one with gamma=1000.

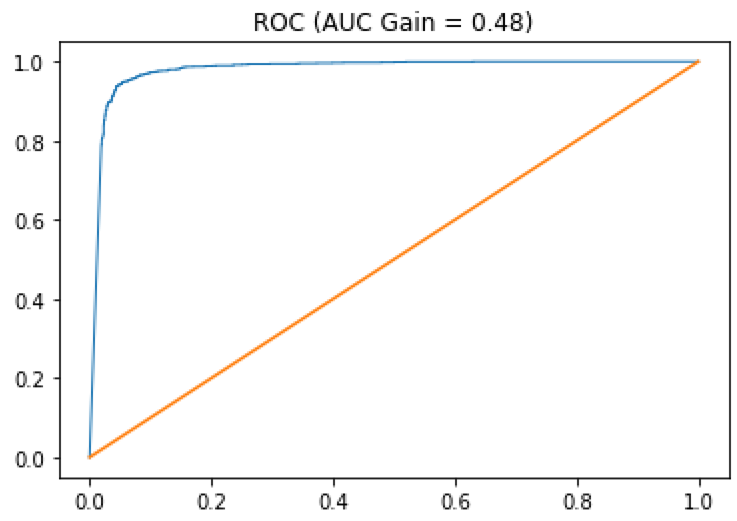
	F1	**Accuracy**	**Precision**	**Recall**	**Confusion Matrix**
gamma= 1000	0.9578	0.9578	0.9582	0.9577	[[1474 86] [47 1543]]

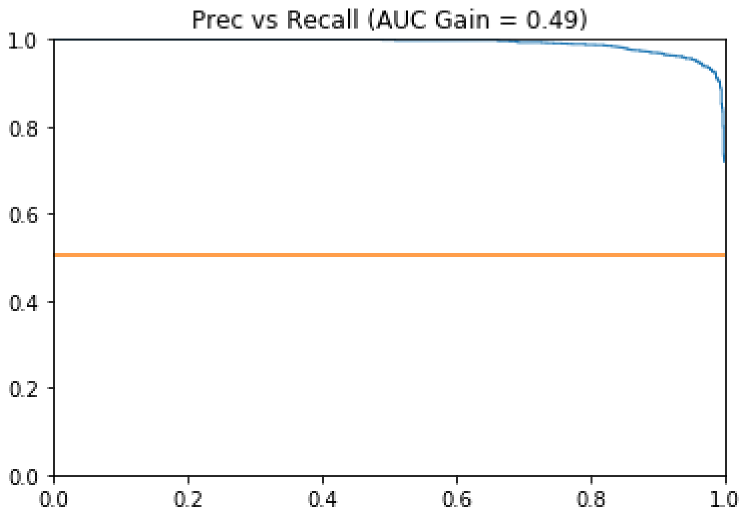
g)

We trained a multinomial naive bayes classifier as the specifications requested. The naive Bayes classifier estimates maximum likelihood (MLE) probability of a class given the document feature set. We assume statistical independence amongst the features.

Accuracy	**Precision**	**Recall**	**Confusion Matrix**
0.9412	0.9422	0.9414	[[1392 168] [12 1578]]

The following ROC curve and Precision Vs. Recall curve is for the naive bayes classifier.



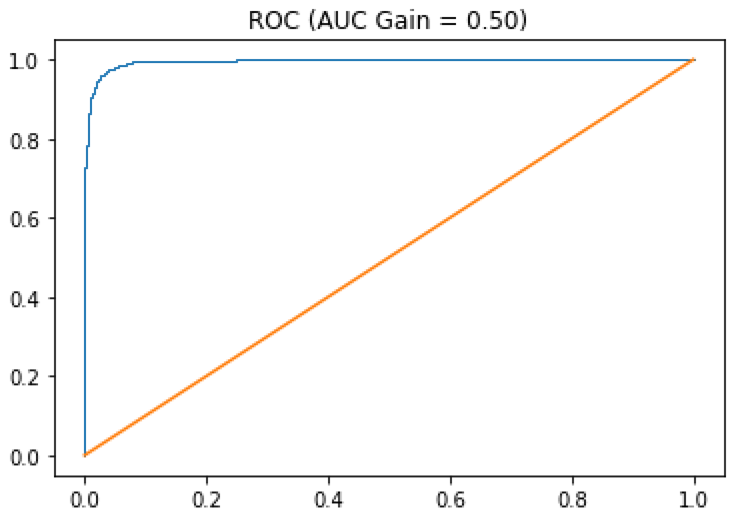


h)

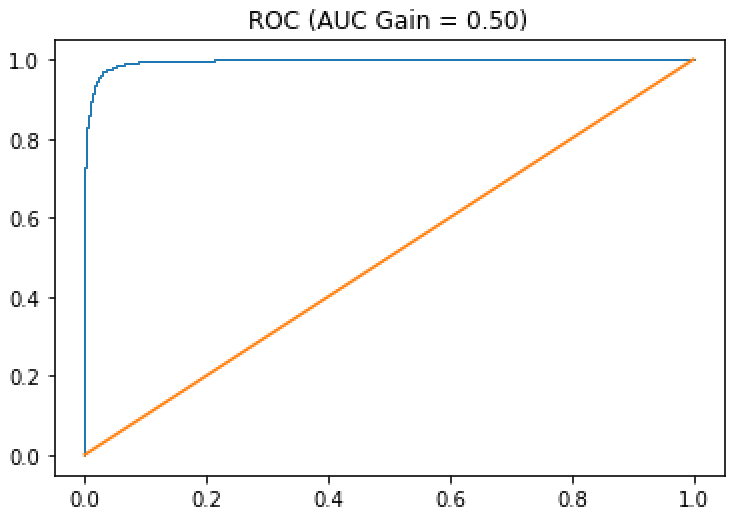
We repeated the same task with the logistic regression classifier. This time, we reported accuracy, precision, recall, and the confusion matrix. Also, we used L2 regularization.

	Accuracy	**Precision**	**Recall**	**Confusion Matrix**
Log Reg L2 (min_df=2, LSI)	0.9653	0.9657	0.9652	[[1486 74] [35 1555]]
Log Reg L2 (min_df=5, LSI)	0.9666	0.9670	0.9665	[[1487 73] [32 1558]]

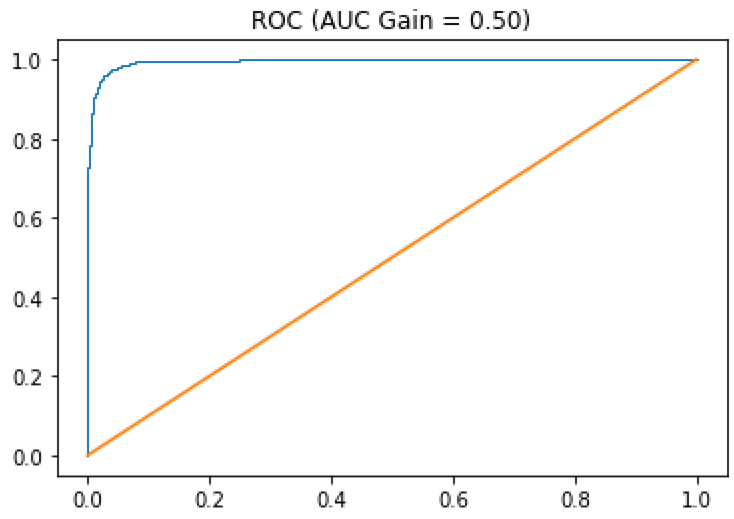
Log Reg L2 (min_df=2, NMF)	0.9653	0.9657	0.9652	[[1486 74] [35 1555]]



Log Reg L2 (min_df=2, LSI)



Log Reg L2 (min_df=5, LSI)



Log Reg L2 (min_df=2, NMF)

i1)

In this section, we added regularization terms to logistic regression. We used L1 regularization rather than L2. Both regularization methods attempt to accomplish the same goal: balance growing weights with classifier cost function. L2 is the sum of square weights and L1 is just the sum of the weights. In general, L1 regularizations lead to more sparse matrices vs. L2.

The table below shows results for logistic regression with L1 regularization. We reported accuracy, precision, recall, and the confusion matrix. Both NMF and LSI were attempted.

	Accuracy	**Precision**	**Recall**	**Confusion Matrix**
Log Reg L1 (min_df=2, LSI)	0.9692	0.9693	0.9691	[[1486 74] [35 1555]]
Log Reg L1 (min_df=5, LSI)	0.9692	0.9693	0.9691	[[1487 73] [32 1558]]
Log Reg L1 (min_df=2, NMF)	0.9619	0.9623	0.9617	[[1486 74] [35 1555]]

Comparing the table in h) and the one above, one can clearly claim that using an L1 penalty gives marginally better performance in all our evaluation metrics. Thus, we will proceed with using L1 as our regularizer, and perform a K-Fold cross validation search on the hyperparameter C (inverse of regularization strength) from a range [1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3]. The best performing classifier was when C = 1e1 with the following performance:

F1	Accuracy	Precision	Recall	Confusion Matrix
0.9406	0.9406	0.9407	0.9407	[[1482 78] [109 1481]]

Smaller values of gamma result in worse test errors empirically. We stored the averaged accuracy for each of the parameters during the sweep. The errors were as follows: [0.49513942783066822, 0.49513942783066822, 0.5959518746267598, 0.95668192872534608, 0.96935910870342723, 0.96745680631390696, 0.96851455992980018].

Clearly, as we increase the regularization parameter, the accuracy improves. The logistic regression optimization problem trades off between the slack variables and the weights. Our inductive bias based on Occam's razor is such that minimum number of parameters and smaller parameter values are better due to simplicity. Regularization penalizes large parameter values in an effort to prevent overfitting. Thus, larger regularization strength will result in smaller hyperplane coefficients. In addition, imposing an L1 regularization over L2, tends weights to zero, thereby performing feature selection in sparse feature spaces.
i2) It can be observed that the generally more complex SVM classifiers perform better than the Naive Bayes classification method.

	F1	Accuracy	Precision	Recall	Confusion Matrix
Naive Bayes (min_df=2, NMF)	0.8064	0.8095	0.8081	0.8119	[[302 89 44 2] [42 246 10 1] [41 42 327 3] [7 8 9 392]]
SVM (One vs. One, min_df=2, LSI)	0.8799	0.8798	0.8832	0.8791	[[342 33 17 0] [62 302 21 0] [24 13 352 1] [12 3 2 381]]
SVM (One vs. One, min_df=2, NMF)	0.7745	0.7801	0.7897	0.7793	[[213 99 78 2] [37 271 74 3] [23 7 359 1] [0 1 19 378]]
SVM (One	0.8854	0.8862	0.8854	0.8856	[[325 41 24 2]

vs. All, min_df=2, LSI)					[39 317 27 2] [18 16 354 2] [4 1 2 391]]
SVM (One vs. All, min_df=2, NMF)	0.8253	0.8281	0.8252	0.8270	[[282 69 36 5] [69 273 39 4] [26 11 350 3] [1 2 4 391]]