Large-Scale Data Mining: Models and Algorithms **ECE 219, Winter 2020**
Prof. Vwani Roychowdhury
UCLA, Department of Electrical and Computer Engineering

Project 1: Classification Analysis on Textual Data

Michael H. Kim (005351728) and Drew A. Mendinueto (705031837)

Introduction

In this project, we will examine and discuss various techniques of data mining and classification for the case of textual data. Classification is the problem of assigning a category or label to a data point, given a training dataset with known categorical assignments. Here, we examine the “20 Newsgroups” dataset, which contains roughly 20,000 documents partitioned into 20 different newsgroup categories. We then perform feature extraction using the “Bag of Words” representation, and “Term Frequency - Inverse Document Frequency” (tf-idf) metric. In these steps, we also perform lemmatization and stemming to remove excess words that are too frequent or too rare across all documents to be useful for classification.

Since the dimensionality of the feature space extracted from our dataset can be very large, we perform dimensionality reduction. Here we compare two different methods of dimensionality reduction: Latent Semantic Indexing (LSI), which is based on a truncated Singular Value Decomposition (SVD), and Non-negative Matrix Factorization (NMF), which approximates our data matrix as a product of two lower dimensional non-negative matrices.

After feature extraction and dimensionality reduction has been performed, we then train and compare different classifiers for our dataset. The first classifier we examine is the Support Vector Machine (SVM), which creates a decision boundary by maximizing the margin between two different classes of data points. The next classifier we demonstrate is the Logistic Regression classifier, which uses a logistic function to determine the probability of a data point belonging to a certain class. The last classifier we train is a Naive Bayes (NB) classifier, which simplifies the Maximum-a-posteriori (MAP) estimation problem by assuming the features are independent when conditioned by the class the data point belongs to, and allows one to estimate the probability of a data point belonging in a class given its features.

All of these classifiers are trained, and their hyperparameters are tuned using cross-validation. However, in order to determine the best classifier for our data, we must perform an exhaustive grid search to isolate the best parameters and classifiers to use. We construct a pipeline that performs the feature extraction, dimensionality reduction, and classification, and performed a grid search with 5-fold cross validation to compare and tune various settings and hyperparameters in our models.

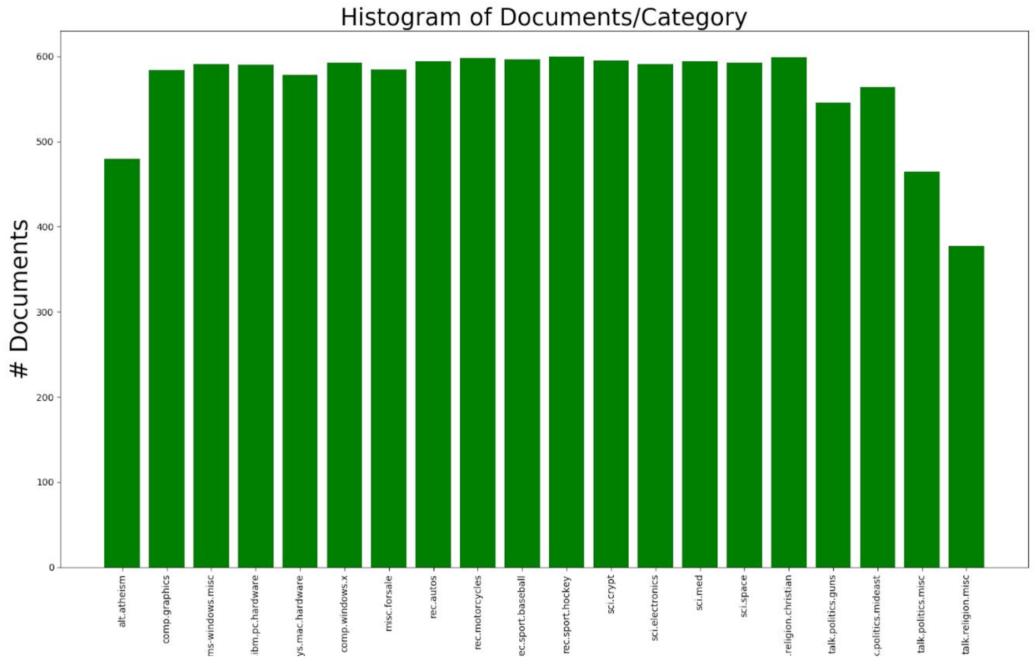
To further generalize our work on binary classification, we also examine multiclass classification using SVM and NB classifiers. NB classifiers are already robust to multiclass classification, since there is no inherent structure in the NB formulation that limits the classifier to only binary classification. However, since SVM determines a decision boundary between two classes, SVM must be further extended to accommodate multiple classes. We examine multiclass SVM classification through both the one vs. one and one vs. rest formulation, and compare the results of these SVM classifiers as well as that of the NB multiclass classifier.

Dataset

The dataset analyzed in this project is the "20 Newsgroups" dataset, which is provided by scikit-learn. This dataset contains roughly 20,000 documents partitioned into 20 different newsgroup categories. It is important to ensure the dataset we use to train our classifiers in the subsequent steps of this project is balanced, so that no one class can skew the metrics we use to evaluate our classifier such that our classifier will favor to classify data points as that class. For example, if we are performing binary classification between two classes, A and B, and 90% of our data belongs to class A, then the classifier can achieve 90% accuracy simply by classifying all the data points as class A. This behavior when training our classifier on imbalanced datasets causes are classifiers to perform poorly on more general datasets. There are several techniques which can combat the negative effects of imbalanced datasets, such as resampling techniques or synthetic sample generation. In our case, the categories of documents we analyze are evenly distributed as examined in the next section, so these techniques are not necessary for our dataset.

Question 1: Distribution of Examples for Training Data

To ensure that the number of documents within each category are evenly distributed, a histogram was created to visually represent this information. As can be seen across the 20 different categories, the number of documents across the 20 categories varied slightly with only 4 categories demonstrating a significant difference in document count. These categories were: "alt.atheism", "talk.politics.misc", "talk.religion.misc", and "talk.politics.guns". Since these skewed categories are not to be included within our dataset, we can safely conclude that our model will not face imbalance challenges posed by our training sets.



Feature Extraction

The first step in classifying textual data is to extract features in our data with enough information to perform classification, while avoiding computational intractability. Since our dataset initially comes as a collection of documents, we need some numerical metric to assess these documents in order to sort them into their respective classes. We use the "Term Frequency-Inverse Document Frequency (TF-IDF)" metric, which produces a normalized count of the vocabulary words in a document corresponding to the importance of that word to that document. This allows us to isolate some discriminating words on which a classifier can more easily sort the documents into their correct class. The TF-IDF score is defined to be:

$$tf\text{–}idf(d,\ t)\ =\ tf(t,\ d)\ \times\ idf(t)$$

where term frequency, tf(d, t), represents the frequency of term t in document d, and inverse document frequency, idf(t), is defined as:

$$idf(t)\ =\ log(\frac{n}{df(t)})\ +\ 1$$

with n being the total number of documents, and the document frequency, df(t), being the number of documents that contain term t.

Before TF-IDF is performed, however, we must clean our dataset for the text to be more tractable for processing. To do this, we also remove specific stopwords, which are words that appear too frequently in all documents to be useful for classification. Furthermore, we also exclude terms that our numbers, and reduce words further based on linguistic rules via stemming and lemmatization. We also set a minimum threshold for which we consider a document frequency significant. The results of our data cleaning and feature extraction will be detailed in the following section.

Question 2: Extract Features from Textual Data

In order to extract features from our textual data, a "Term Frequency-Inverse Document Frequency (TF-IDF)" metric was utilized to demonstrate the significance of a word in regards to each document. Prior to determining the TF-IDF metric for the 20newsgroup dataset, four specifications were met to create a stronger representation of the bag-of-words under analysis.

Common words or phrases can skew the weights of the model and lead to unreliable predictions on future input. To reduce the number of insignificant words within our dataset, the documents were pre-processed using four specifications: 'english' stopwords, exclusion of numbers, lemmatization, and a minimum document frequency (min_df). The 'english' stopwords consisted of common words such as "I" and "he" that could take away from the classifier's ability to create an appropriate representation of each classification. For the purposes of this project, we created a single list of words containing these stopwords and punctuation that would be omitted from the document. Additionally, within the same text analyzer function, we removed any terms that were numbers.

The lemmatization of each document was also performed within the text analyzer function. The lemmatizer would essentially reduce wording by simplifying terms to their respective root meanings. For example, "is" and "are" perform the same function and both their meaning comes from the root word "to be". Therefore, the lemmatizer would process both terms and output "be", reducing the number of different words being processed within the training set. Finally, the minimum document frequency was set to 3, meaning that a term would only be considered significant if it was present within at least 3 different documents.

Specifications:

1) Use the “english” stopwords of the CountVectorizer

In order to achieve this specification, the ‘english’ stopwords set was imported from nltk. This word-set would then be used as the parameter for ‘stopwords’ in CountVectorizer. The importance of stopwords is to ignore frequent words that might take away from the TF-IDF analysis of a corpus, thereby, allowing for distinguishable wording to take on more significance within this metric.

2) Exclude terms that are numbers (e.g. “123”, “-45”, “6.7” etc.)

This specification performs the same function as the usage of stopwords. To achieve this specification, a new function was defined (no_number_preprocessor) to remove any numeric character from each body of text. This function would then be used as the parameter for ‘processor’ in CountVectorizer.

3) Perform lemmatization with nltk.wordnet.WordNetLemmatizer and pos tag

The importance of lemmatization is to further reduce the repetition of words that essentially perform the same function. For example, ‘walks’ would be considered the same as ‘walk’. While stemming finds similarities through deeper roots of each word (ignoring tense), lemmatization looks to maintain differentiation between these usages of words as well, upholding linguistic rules. In order to perform lemmatization within each document, the corpus string was first converted into a list of words. Secondly, each word in question was lower-case alphabetized and tagged with its respective part of speech (noun, verb, adverb, adjective), then processed through the WordNetLemmatizer().

4) Use min df=3.

This sets the minimum document frequency to 3, so that only words that appear in at least 3 documents will be a relevant feature for our classification.

These specifications would be placed as parameters within the CountVectorizer() function and an array of term frequencies would be output. The resulting array would then be placed through a TfidfTransformer() to assign each term with a weight based upon its frequency and the number of documents that contain the word.

TFidfTransformer() training dataset shape:

(4732, 13620)

TFidfTransformer() testing dataset shape:

(3150, 13620)

The same TfidfTransformer() object and CountVectorizer() object were used to process both the training and testing datasets. This would maintain uniformity throughout the processes of the data pipeline.

After the transformation, we obtain a TF-IDF matrix that contains the tf-idf value for each element (i, j) of some document-term pair, (d_i, t_j). The shape of the TF-IDF matrix tells us the number of documents scanned and the number of unique words found across these documents. Additionally, we can observe that the training dataset has approximately 50% more documents than the testing dataset which is accountable for the longer length of the training dataset.

Dimensionality Reduction

In order to make analysis of our data more tractable, we can reduce the dimensionality of the feature vectors that we extract. After we performed feature extraction, our feature dimensions are on the order of 10^4. To train and evaluate classifiers on a feature set this large would take large amounts of computation, and the classifier may perform poorly on such high dimensional data. We can take advantage of the sparsity of our TF-IDF matrix, and isolate a subset of the features that carry enough information to accurately classify our data. To determine this lower dimensional set of features, we can employ two different dimensionality reduction methods: Latent Semantic Indexing (LSI) and Non-negative Matrix Factorization (NMF).

Latent Semantic Indexing (LSI)

Latent Semantic Indexing is based on the SVD of the TF-IDF matrix, **X**. After performing the SVD of the matrix **X**, we obtain $X = U\Sigma V^T$**,** with U and V orthogonal, and the singular values in Σ are sorted in descending order. We can pick the *k* largest singular values of **X,** and isolate the corresponding *k* singular vectors of **X** as $\mathbf{V_k}$. Here, we can use $\mathbf{XV_k}$ as the dimensionally reduced feature set, and having learnt $\mathbf{V_k}$, we can also project the test TF-IDF matrix by right multiplying by $\mathbf{V_k}$.

Non-negative Matrix Factorization (NMF)

Non-negative Matrix Factorization approximates our original matrix **X** as the product of two lower-dimensional non-negative matrices **WH**. In the context of TF-IDF, when we perform this type of dimensionality reduction, we are essentially trying to describe the documents as a non-negative linear combination of topics. The number of topics we attempt to represent the documents as corresponds to a reduced dimensionality compared to the original feature space. We can find **WH** by minimizing the mean squared residual between **X**, and we use **W** as the reduced feature set. We can use **H** from the fit step to calculate via minimization of mean squared residual the testing $\mathbf{W_t}$ used to represent the test data in the lower dimensional feature space.

Question 3: Dimensionality Reduction on Textual Dataset

We performed both LSI and NMF on our dataset to reduce the dimensionality of our feature space. We apply LSI to the TF-IDF matrix corresponding to the 8 chosen categories with k = 50, so that each document is mapped to a 50-dimensional vector. The result matrix shape is then:

LSI Dimensionality Reduction, Reduced TF-IDF matrix shape (training set):
    (4732, 50)
LSI Dimensionality Reduction, Reduced TF-IDF matrix shape (testing set):
    (3150, 50)

We also reduce dimensionality through NMF, with k = 50, which yields the same shapes:

NMF Dimensionality Reduction, Reduced TF-IDF matrix shape (training set):
    (4732, 50)
NMF Dimensionality Reduction, Reduced TF-IDF matrix shape (testing set):
    (3150, 50)

We can compare the residual from LSI vs. NMF by computing:

LSI: $\| X - U_K \Sigma_k V_k^{\,T} \|_F^{\,2}$

NMF: $\| X - WH \|_F^{\,2}$

We obtain the residuals for the respective dimensionally reductions as:

LSI: 63.7908

NMF: 64.1142

We observe that both residuals are fairly close in value, but the residual calculated for the LSI is slightly smaller. This makes sense because LSI uses the SVD to reduce the feature dimensionality of the dataset. Since the SVD aims to decompose a matrix into its singular values and its corresponding singular vectors, if we keep the subset corresponding to the largest singular values of that matrix in our dimensionality reduction, we can retain a large portion of the information contained in the original matrix. The residuals are fairly large for both because we are projecting the matrices into a much smaller feature subspace, and most of the information cannot be reconstructed from the smaller feature subspace to the original feature space. More specifically, the zeroes are reinterpreted by the NMF and LSI matrices as extremely small non-zero, positive numbers.

Classification Algorithms and Metrics

In the following sections, we will train various classifiers on the dimension-reduced training data from LSI, and evaluate the trained classifiers with test data. Specifically, we will examine three classifiers: 1. Support Vector Machines (SVM), 2. Logistic Regression, and 3. Naïve Bayes Classifier (NB). The SVM creates a decision boundary by maximizing the margin between two different classes of data points. The Logistic Regression classifier uses a logistic function to determine the probability of a data point belonging to a certain class. The Naive Bayes (NB) classifier simplifies the Maximum-a-posteriori (MAP) estimation problem by assuming the features are independent when conditioned by the class the data point belongs to, and allows one to estimate the probability of a data point belonging in a class given its features.

We assess these classifiers on an assortment of metrics: Receiver Operating Characteristic Curve (ROC Curve), Confusion Matrix, Accuracy, Recall, Precision, and F-1 Score. The ROC curve is a plot of the True Positive Rate against the False Positive Rate of a classifier at certain thresholds, and its area under the curve (AUC) is a measure of the classifier's separability. When the AUC is close to 1, then the classifier performs very well, and when it is close to 0, the classifier performs poorly. The Confusion Matrix is a matrix whose rows contain the number of instances in a predict classes and whose columns contain the number of instances in the actual classes. The Accuracy is the proportion of correctly classified data points to the total number of points. The Recall is the ratio of true positives to true positives + false negatives, or in other words the ratio of selected relevant data points to total relevant data points. The Precision is the ratio of true positives to true positives + false positives, or in other words the ratio of selected relevant data points to the total selected data points. The precision describes how relevant the results are whereas the recall describes how completely the relevant results were classified. Since there is typically a trade-off between these two metrics, we use the F-1 Score, which is the harmonic mean of Precision and Recall, to better evaluate the performance of classifiers.

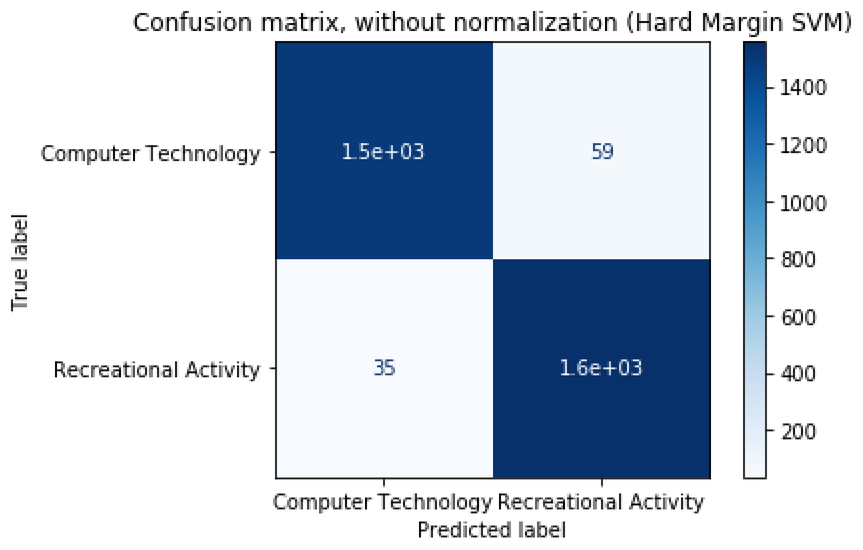
Question 4: Support Vector Machines

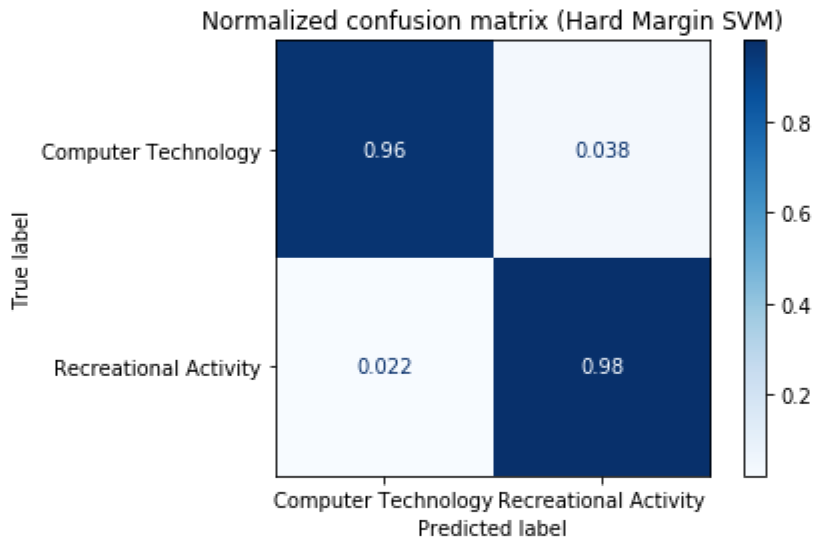
Hard Margin SVM

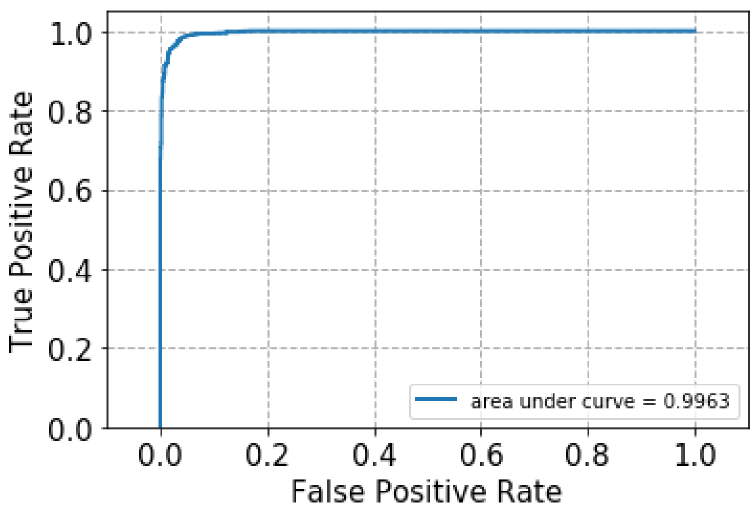
We trained a linear SVM with $\gamma = 1000$. This γ parameter determines the strength of the margin of the SVM. In this case, the SVM that we are training is a "Hard Margin" SVM, in which misclassified points are heavily penalized in the training phase. A Hard Margin SVM seeks a decision boundary in which a minimum number of points are misclassified in the training set. In terms of generalizability, this could mean that the SVM will not separate the two classes very well, which could lead to a larger error when evaluated on a test dataset.

We report the performance metrics of our Hard Margin SVM evaluated on our testing dataset below:

Accuracy:	0.9701587301587301
Precision:	0.9634448574969021
Recall:	0.9779874213836478
F-1 score:	0.9706616729088638


Confusion matrix, without normalization (Hard Margin SVM)
True label
Computer Technology
Recreational Activity
1.5e+03
59
35
1.6e+03
Predicted label
1400
1200
1000
800
600
400
200


Normalized confusion matrix (Hard Margin SVM)
True label
Computer Technology
Recreational Activity
0.96
0.038
0.022
0.98
Predicted label
0.8
0.6
0.4
0.2

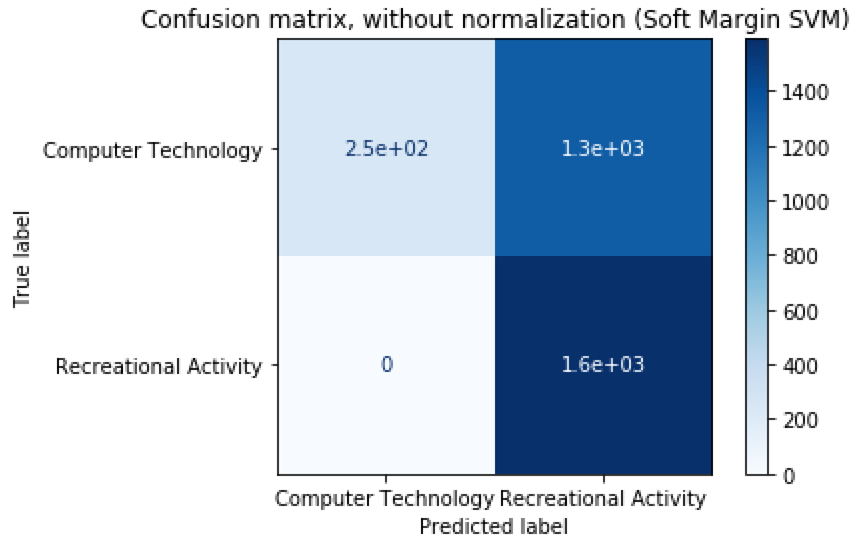

True Positive Rate
False Positive Rate
area under curve = 0.9963

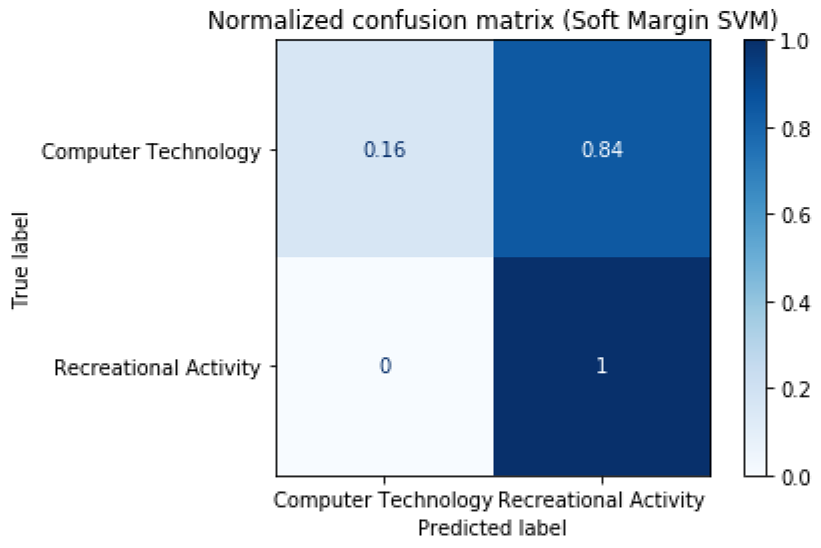
Soft Margin SVM

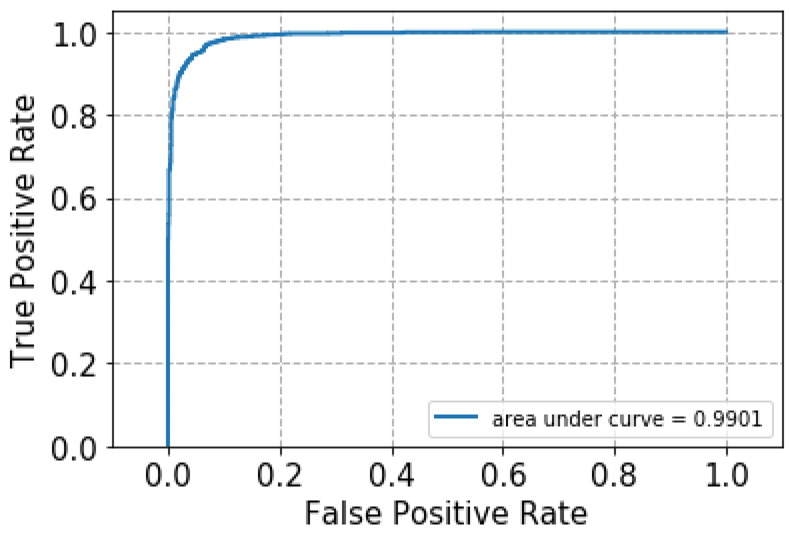
We trained a linear SVM with $\gamma = 0.0001$. This γ parameter determines the strength of the margin of the SVM. In this case, the SVM that we are training is a “Soft Margin” SVM, in which misclassified points are not penalized that much in the training phase, so long as the points in the training set are well separated. A Soft Margin SVM seeks a decision boundary in which some points can be misclassified in the training set, but the decision boundary maximizes the distance between the closest data point of both classes to the boundary. In terms of generalizability, this means that the SVM will separate the two classes very well but some points can be misclassified, which could lead to a larger error when evaluated on a test dataset.

We report the performance metrics of our Soft Margin SVM evaluated on our testing dataset below:

Accuracy:	0.5831746031746031
Precision:	0.5477092662762659
Recall:	1.0
F-1 score:	0.7077676385488537







Hard Margin vs. Soft Margin SVM

If we compare the performance of both the Hard and Soft Margin SVMs, we see that the Hard Margin SVM overall performs much better. The Hard Margin SVM has consistently high metrics of accuracy, precision, recall, F-1, and AUC in the high 90% range. The Soft Margin SVM, however, has a perfect recall score, and even though its accuracy, precision and F-1 scores are relative low and are vastly outclassed by the Hard Margin SVM, its ROC curve looks very good with an AUC of 0.9901. This good performance based on the ROC curve conflicts with the precision, accuracy, and F-1 metrics, but it makes sense considering the perfect recall score. This is likely due to the Soft Margin maximizing the distance between the two classes, such that when it's evaluated on the test set, even though not all the points are correctly classified, all the points of one particular class are classified correctly regardless of the other misclassifications. Overall, however, the Hard Margin SVM performs better, especially when assessing the performance based on the F-1 score. This is because although Soft Margin SVM has a perfect recall, it has extremely low precision, which is also an important metric to consider when evaluating the performance of a classifier.

Tuned Hyperparameter SVM

In this next section, we use cross-validation to tune the hyperparameter γ, which corresponds to the strength of the margin. We use 5-fold cross-validation, in which we partition the training dataset into 5 folds, and train the classifier on 4 of those folds and evaluate the performance on the remaining fold. We repeat this process until the classifier has been assessed on all of the folds. From this, we can tune the hyperparameter γ and pick the best value that resulted in the highest average validation accuracy. We tune the hyperparameter using a separate validation set from the testing set so as not to overfit the testing set itself, and preserve the generalizability of the classifier. If we were to tune the hyperparameter on the testing dataset, then we would likely overfit the classifier on the testing set and reduce the generalizability of the classifier. Here, we find the best value of the parameter γ in the range 0.001 to 1000. We first construct a tuning space that ranges from 0.001 to 1000 with 50 points that are spaced in logarithmic increments, i.e. logspace(0.001, 1000, 50). This ensures that each order of magnitude in the tuning space is well represented, and we can sweep the hyperparameter over the entire range with a large number of sample points to precisely determine an optimal hyperparameter.

After cross-validation, we determine an optimal γ value to be:

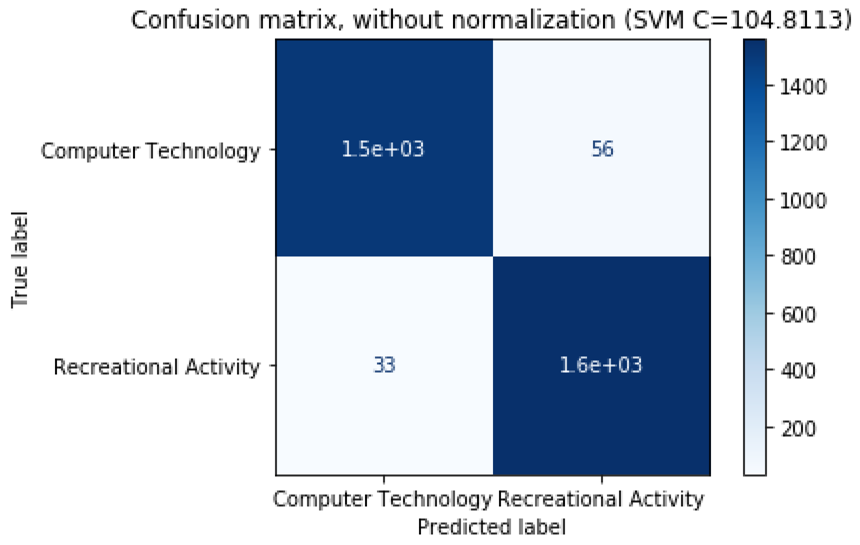
$\gamma = 104.81131341546852$

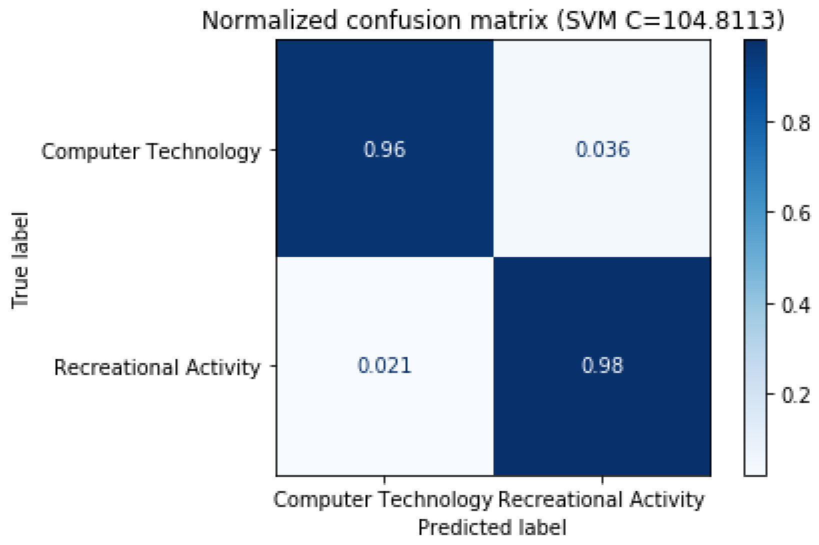
A summary of the cross-validation is shown below, with mean accuracy evaluated on the validation set:

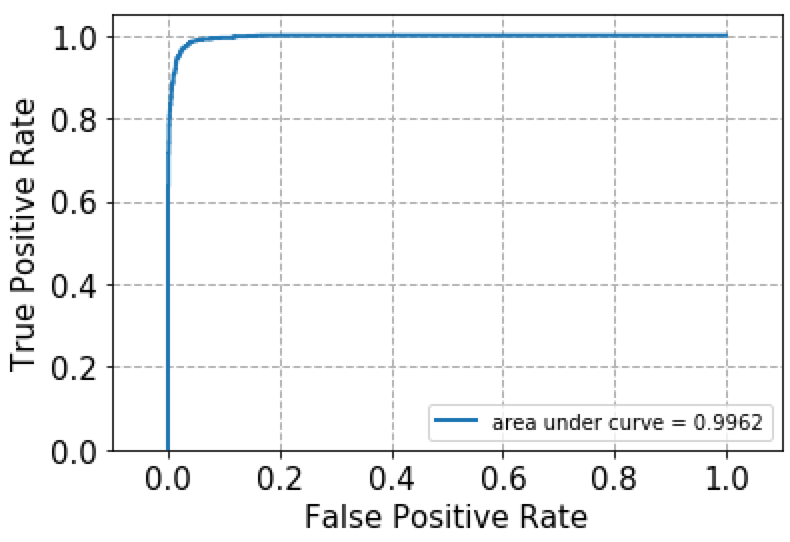
γ	0.001	0.012649	0.12068	1.1514	10.9854	**104.811**	1000
Accuracy	0.92183	0.948424	0.959267	0.965649	0.967392	**0.969943**	0.963145

We evaluated the classifier with the tuned hyperparameter on the testing dataset, and found the metrics to be:

Accuracy:	0.9717460317460317
Precision:	0.9652820830750155
Recall:	0.9792452830188679
F-1 score:	0.9722135497970652







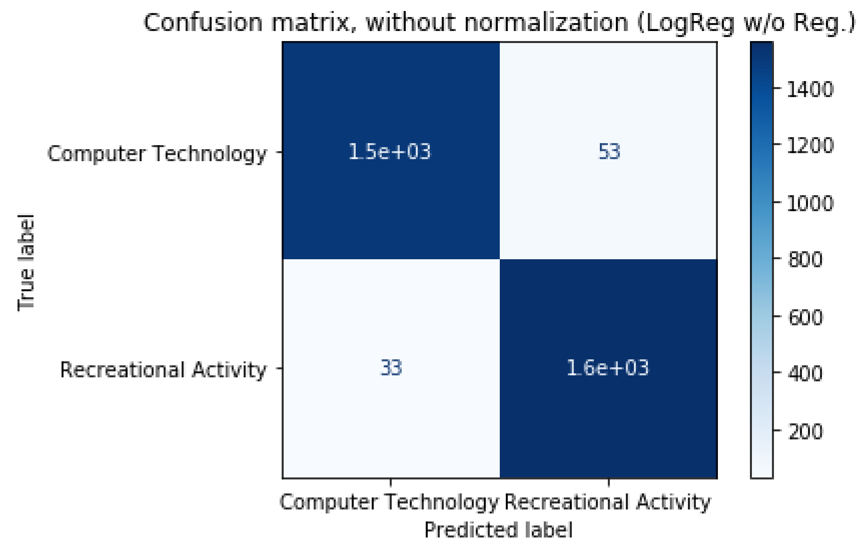
We see that our tuned SVM performs well, similar to the performance of the Hard Margin SVM.

Question 5: Logistic Classifier

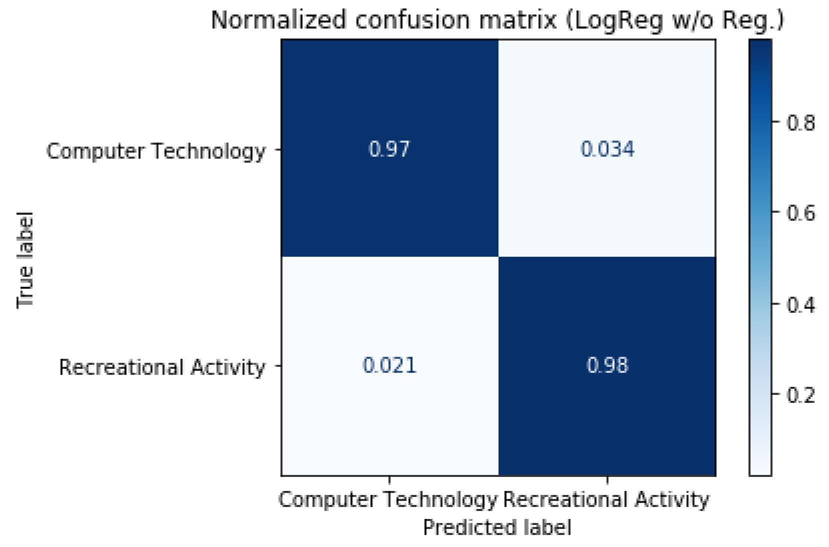
Without Regularization

We trained a Logistic Classifier without regularization on our training dataset. To approximate the training phase without regularization, we set the parameter, C, in the Logistic Classifier constructor to be a very large number, since this C corresponds to the inverse of the regularization strength. After training, we evaluated our classifier on the testing dataset and obtain the following metrics:

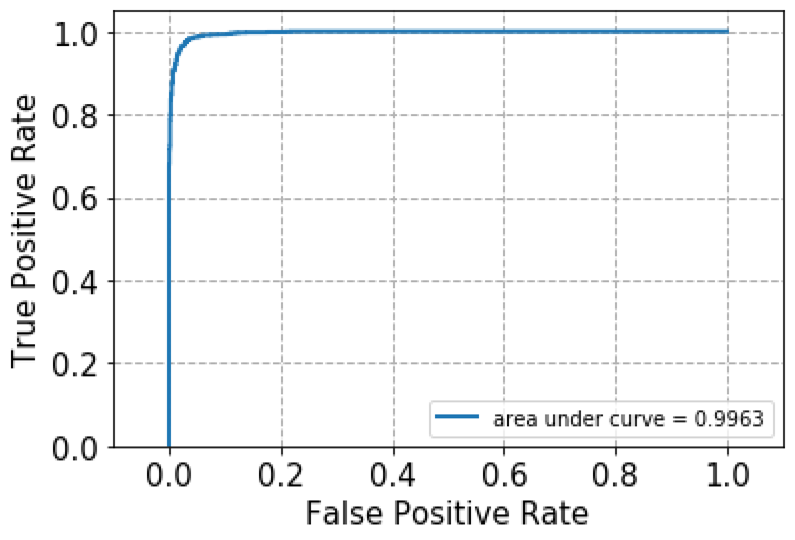
Accuracy:	0.9726984126984127
Precision:	0.9670807453416149
Recall:	0.9792452830188679
F-1 score:	0.973125



Confusion matrix, without normalization (LogReg w/o Reg.)
Computer Technology
Recreational Activity
True label
1.5e+03
53
33
1.6e+03
Computer Technology
Recreational Activity
Predicted label
1400
1200
1000
800
600
400
200



Normalized confusion matrix (LogReg w/o Reg.)
Computer Technology
Recreational Activity
True label
0.97
0.034
0.021
0.98
Computer Technology
Recreational Activity
Predicted label
0.8
0.6
0.4
0.2



True Positive Rate
False Positive Rate
area under curve = 0.9963
0.0
0.2
0.4
0.6
0.8
1.0

L2 Regularization

Using 5-fold cross-validation, tuned the regularization strength of our logistic classifier on the training dataset through evaluation on the validation dataset. In this case, we want to compare different types of regularization of our Logistic Classifier, so we start with L2 Regularization. We swept the regularization strength, C, in 50 logarithmic increments from 0.001 to 1000. After cross-validation, we tuned the regularization strength, C, for L2 Regularization to be:

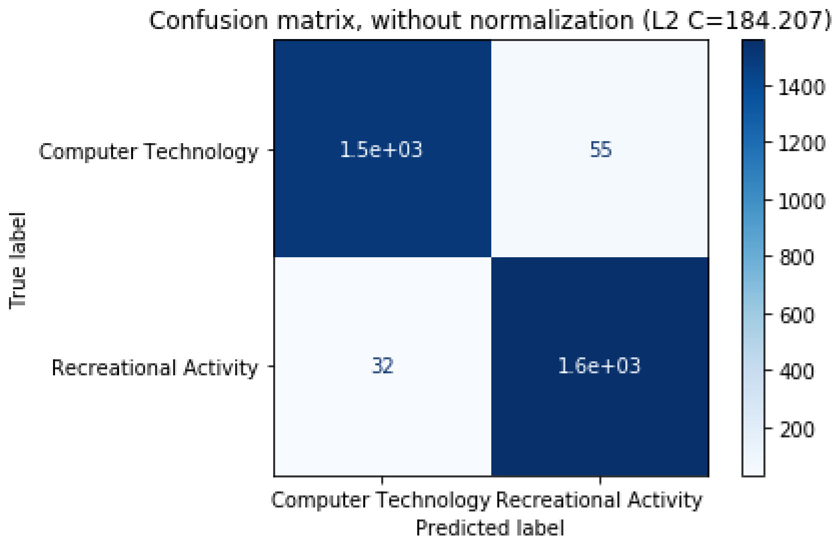
C = 184.207

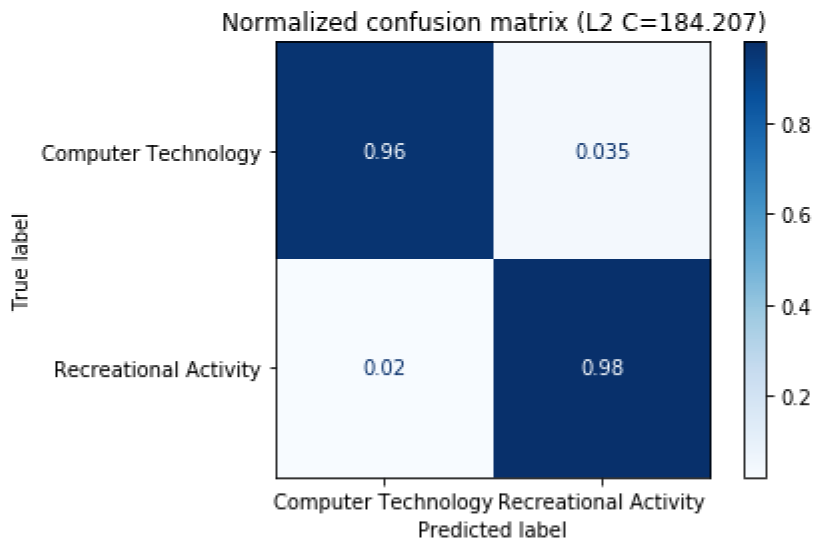
A summary of the cross-validation is shown below, with mean accuracy evaluated on the validation set:

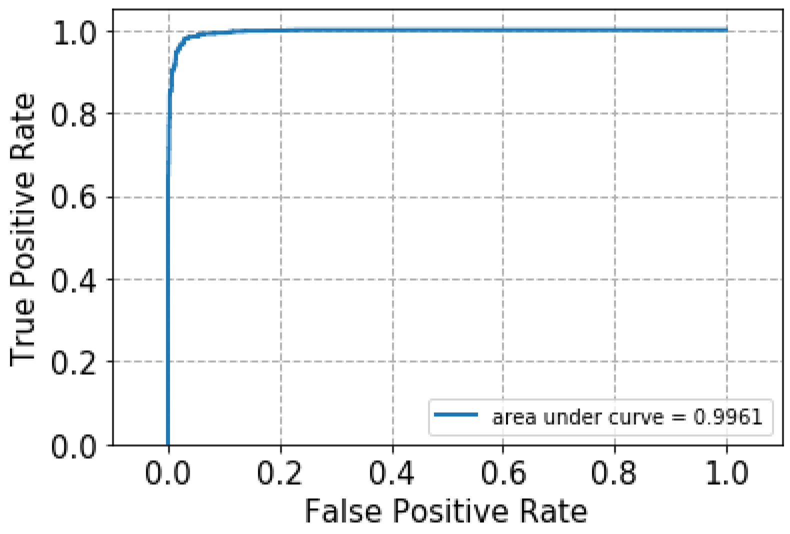
C	0.001	0.01265	0.12068	1.1514	10.9854	104.811	**184.207**	1000
Accuracy	0.76954	0.93213	0.94970	0.95949	0.96757	0.97081	**0.97102**	0.97018

We evaluated this classifier on the test dataset, and obtain the following metrics:

Accuracy:	0.9723809523809523
Precision:	0.9659020458772474
Recall:	0.979874213836478
F-1 score:	0.9728379644083671







L1 Regularization

Likewise, for L1 Regularization, we used 5-fold cross-validation to tun the regularization strength, C, of our logistic classifier on the training dataset through evaluation on the validation dataset. We swept the regularization strength, C, in 50 logarithmic increments from 0.001 to 1000. After cross-validation, we tuned the regularization strength, C, for L1 Regularization to be:

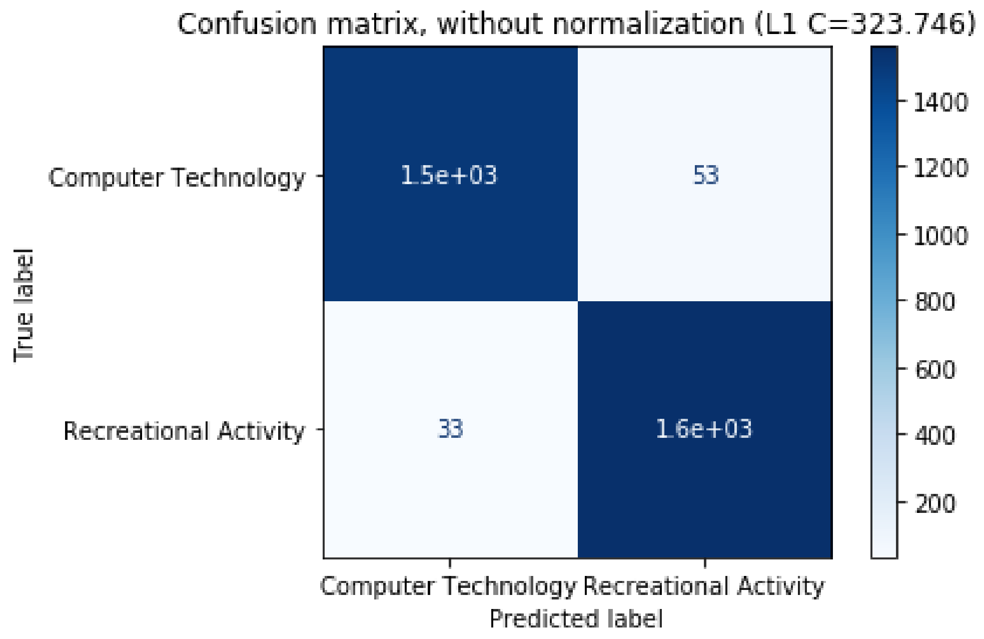
C = 323.746

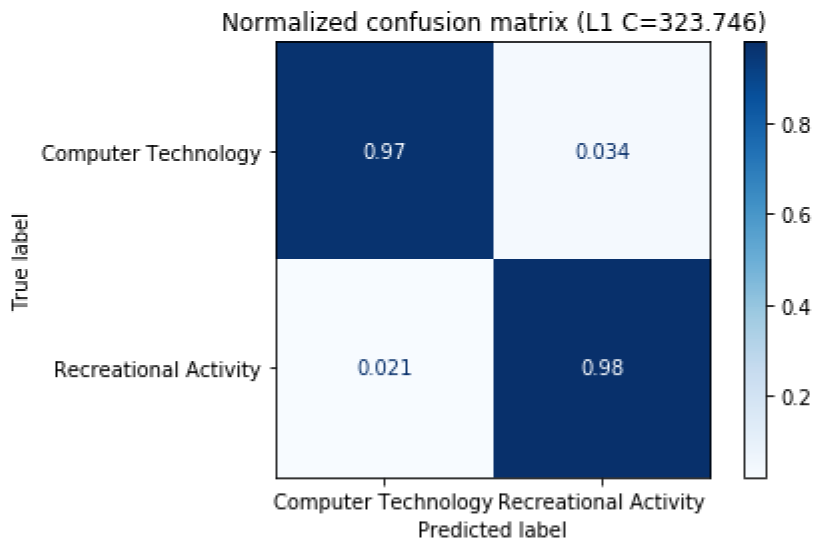
A summary of the cross-validation is shown below, with mean accuracy evaluated on the validation set:

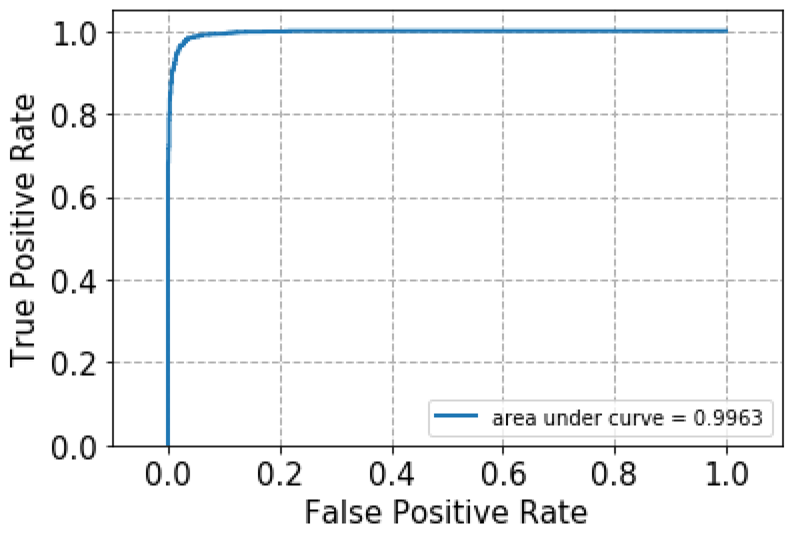
C	0.001	0.01265	0.12068	1.1514	10.9854	104.811	**323.746**	1000
Accuracy	0.50000	0.92077	0.93548	0.96493	0.96892	0.96955	**0.97019**	0.96998

We evaluated this classifier on the test dataset, and obtain the following metrics:

Accuracy:	0.9726984126984127
Precision:	0.9670807453416149
Recall:	0.9792452830188679
F-1 score:	0.973125







Comparison of Results for Logistic Classifier

If we compare the performance of the three Logistic Classifiers when evaluated on the test data, we see that the three classifiers perform very similar. The accuracy of each classifier is approximately 0.97, and the F-1 scores of each classifier are all about 0.97 as well, with L2 Regularization having a slightly lower F-1 score on the order of 0.001 compared to the other two. Furthermore, the ROC curve and AUC across all three classifiers are nearly identical.

As the regularization strength increases (as C decreases, since C is inverse regularization strength), the testing accuracy tends to decrease. The learnt coefficients of decision boundary (i.e. a hyperplane for logistic regression) should swing toward 0 as the regularization strength increases. This is because increasing the regularization strength would increase the weights of these coefficients in the loss function of the Logistic Regression, which increases the incentive to minimize these coefficients.

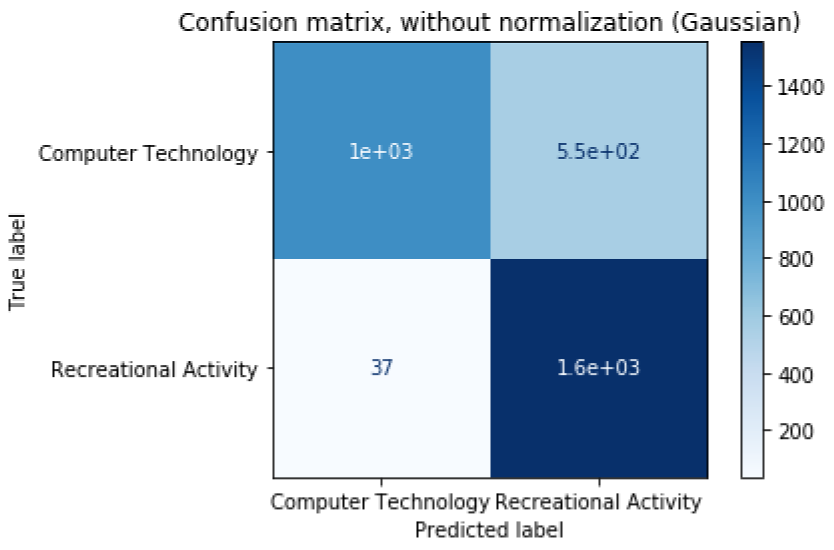
One might be interested in each type of regularization for the different attributes each type of regularization has. One might be interested in L1 regularization because it tends to create sparse outputs. In a sense, this adds the benefit of additional feature selection in the algorithm. Conversely, L2 regularization does not have this robust feature selection built-in, but it does tend to produce more overall accurate classifiers.

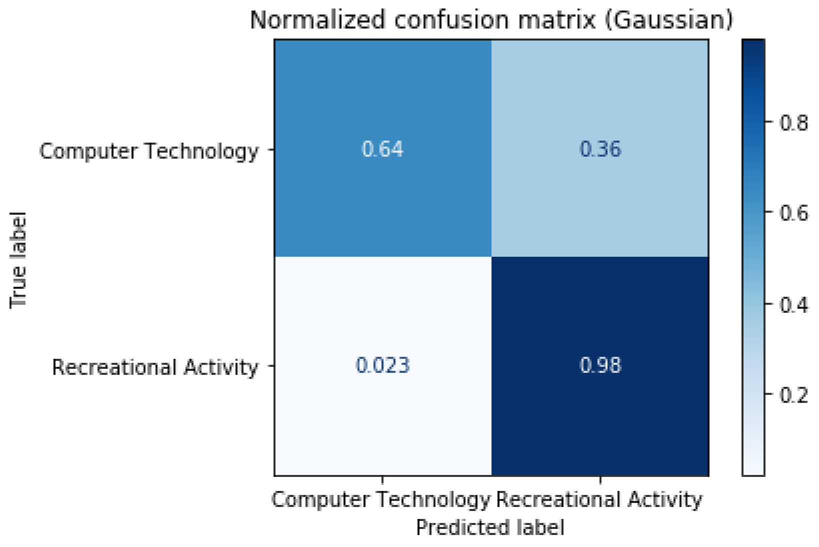
Both Logistic Regression and Linear SVM classify data points using a linear decision boundary. The primary difference is the way these two classifiers determine this boundary. In Linear SVM, the algorithm seeks to maximize the margin, or distance, between two different classes. This creates a linear decision boundary that separates the classes with as large a margin as possible. In Logistic Regression, a logistic function is used to determine whether a data point belongs to one class or the other. This logistic function has a smooth fall-off typically, and the decision boundary is then based on how large or small the output of this logistic function is when passed the data point as an input. This logistic function can also be interpreted as a function that transforms the data point into some probability of being in a certain class, and the classifier's decision boundary is based on which class yields the highest probability. The primary difference between two classifiers can be summarized by the type of loss function that is minimized. In SVM, it is typically hinge loss which is minimized, whereas in Logistic Regression it is log loss. The performance of SVM vs Logistic Regression differs because of this loss function difference; this is because logistic functions are more sensitive to outliers than the hinge loss, due to the faster divergence of logistic loss. Logistic loss functions also do not go to 0 even when a point is correctly classified, which could negatively affect the accuracy of the classifier. Overall, we should expect a well trained and tuned SVM to dominate in terms of performance.

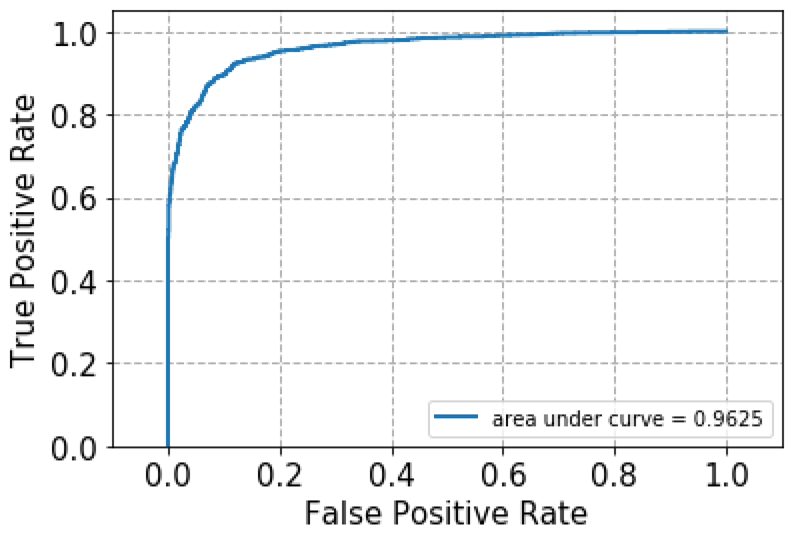
Question 6: Naïve Bayes Classifier

We trained a Gaussian NB classifier on our training dataset, and we evaluated the classifier on the testing dataset. We found the metrics to be:

Accuracy:	**0.8123809523809524**
Precision:	**0.7370669197911723**
Recall:	**0.9767295597484277**
F-1 score:	**0.8401406545847986**


Confusion matrix, without normalization (Gaussian)
True label
Predicted label
Computer Technology
Recreational Activity
Computer Technology
Recreational Activity
1e+03
5.5e+02
37
1.6e+03
1400
1200
1000
800
600
400
200


Normalized confusion matrix (Gaussian)
True label
Predicted label
Computer Technology
Recreational Activity
Computer Technology
Recreational Activity
0.64
0.36
0.023
0.98
0.8
0.6
0.4
0.2


True Positive Rate
False Positive Rate
area under curve = 0.9625
0.0
0.2
0.4
0.6
0.8
1.0

We see from these results that the Gaussian NB classifier does not perform as well as the SVM or Logistic Classifiers. This could likely be due to the NB model strongly relying on a particular probability distribution, since the NB classifier is a probabilistic generative model. The NB classifier estimates the probabilities of a data point belonging to each class and chooses the class corresponding to the highest probability as its classification. If we assume the underlying distribution is Gaussian when perhaps the data is better modeled by some other distribution, such as Multinomial, this could lead to significant degradations in the model accuracy.

Question 7: Grid Search using Pipeline

We created a pipeline that performs feature extraction, dimensionality reduction, and classification and conducted a grid search using 5-fold cross-validation. We compared the following parameters in terms of their mean accuracy score:

Procedure	**Options**
Loading Data	Remove “headers” and “footers” vs. not
Feature Extraction	*Minimum Document Frequency:* min_df = 3 vs. 5 Use lemmatization vs. not
Dimensionality Reduction	LSI vs. NMF
Classifier	SVM with best γ previously found vs. Logistic Regression: L1 vs. L2 Regularization with the best regularization strength previously found Gaussian Naive Bayes

Due to the size of the table and the difficulty to present all the information, we decided to discuss the impact of each parameter on the different classifiers.

Loading Data:
When comparing the accuracies of the different classifiers, the impact of headers/footers vs no headers/footers was slightly seen as favoring the removal of headers/footers. Most classifiers without the header/footer information performed a few percentage points higher than its respective header/footer contained counterpart, with only 2 examples out of the norm. Therefore, it can be safely concluded that for the best results, a training dataset without headers/footers will lead to a stronger classification model.

Lemmatization:
One of the big contributors to creating a reliable classifier was the implementation of lemmatization to the documents. Without lemmatization, many of the classifiers failed to converge and the ones that did failed to classify the documents at a reliable rate of success. The highest rate of success for classifiers without lemmatization was 0.408571 by the L2 regularized logistic regression classifier with an NMF transformed dataset. Therefore, the impact of lemmatization is vital for textual analysis via machine learning models.

Minimum Document Frequency:
Overall, the change of 3 to 5 for the minimum document frequency parameter had little to no impact on the strength of the classifier. For the most part, when the parameter was set to 5, stronger models were seen, but again the impact of this characteristic was very little.

Dimensionality Reduction:
This parameter had a more unique impact on the models than the other variables within the pipeline. The NMF models overall performed worse in comparison to LSI/LSA when given a stronger training dataset (min_df=5, no headers/footers, lemmatization), but performed better when given a weaker training dataset (no lemmatization, headers/footers). At times the LSI/LSA model would fail to converge to the high variability within the training set or fail to create a reliable model based upon the input. On the other hand, the NMF models would at least be able to converge and create a classification model (although very inaccurate) for the high variance training dataset. Therefore, given the circumstances of a rough training dataset, NMF would be the dimensionality reduction technique to go, but for all other situations (to create a reliable model), LSI/LSA would perform the best.

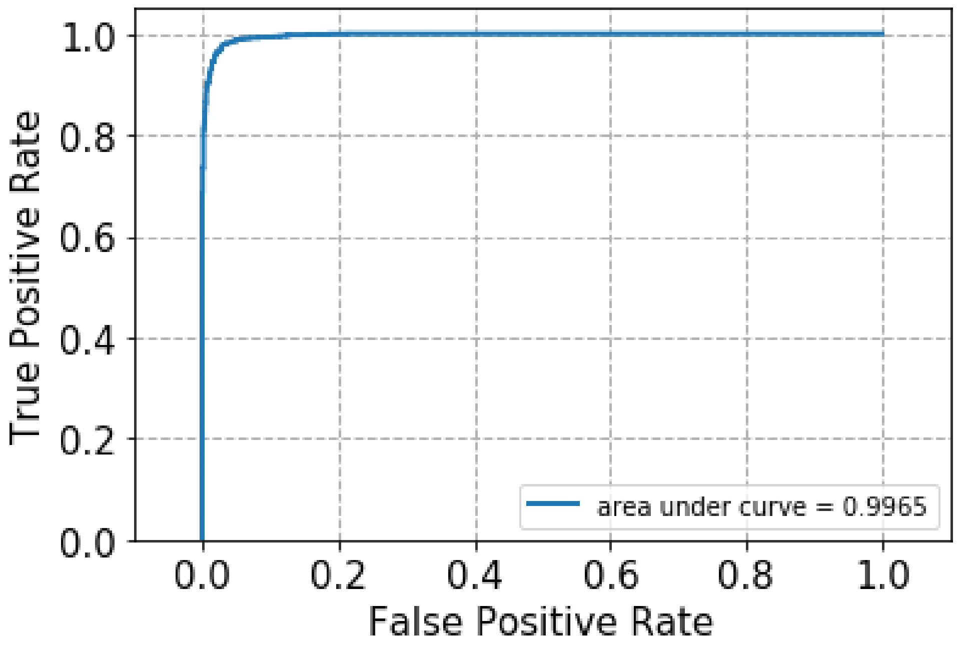
From the comparison of the results from the pipeline, we determined the best combination to be:

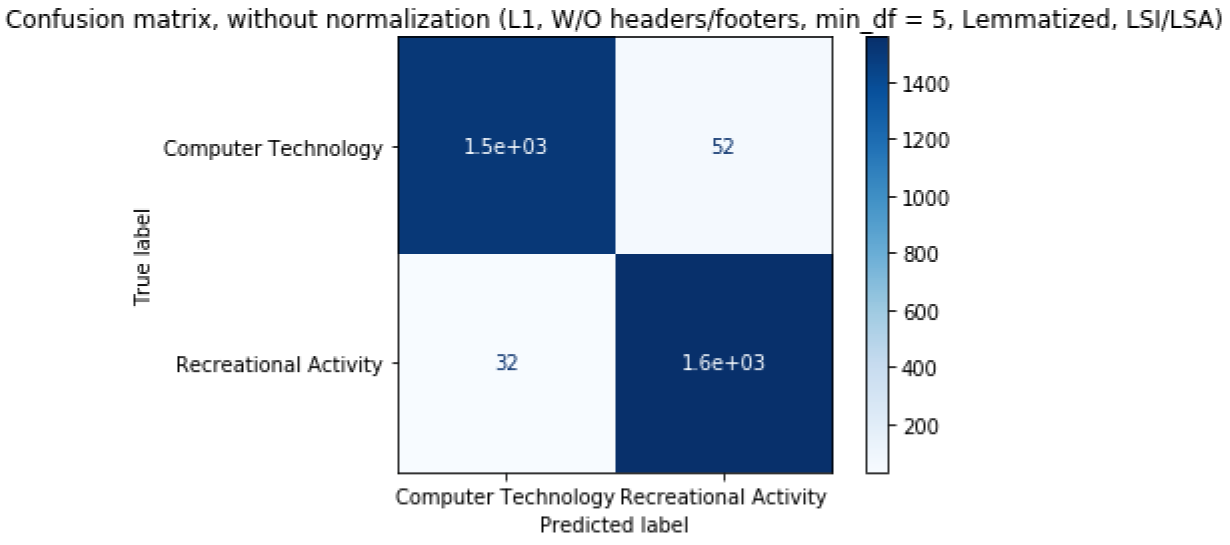
Best performer:

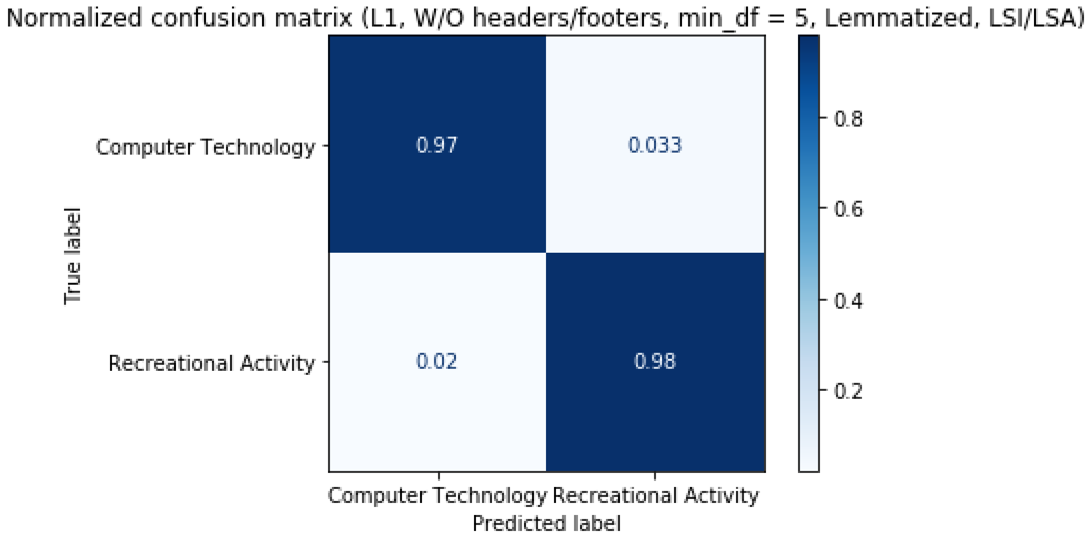
Logistic Regression
L1 Regularization with C = 323.746
Removed headers/footers
Lemmatized
min_df = 5
TruncatedSVD (LSI/LSA)

After evaluation on the test dataset, the performance metrics for the L1 Regularization Classifier are shown below:

Accuracy:	0.9733333333333334
Precision:	0.9677018633540373
Recall:	0.979874213836478
f1 score:	0.97375







We see that the performance metrics of this grid searched classifier does not perform as well as some of our previous Logistic or SVM classifiers. This could be due to overfitting of the training set, such that when we evaluate our classifiers on the testing set our classifiers lack the sufficient generality to classify the testing data set well. Also, it could be that our grid search somehow found a local minimum of the classification error, and as a result made our final classifier suboptimal compared to a global minimum in classification error over the validation set. However, overall our classifier does perform well, despite the lower performance compared to the previously examined classifiers.

A full comparison table of the pipeline is presented in the corresponding Jupyter Notebook.

Question 8: Multiclass Classification

In this section, we explore multiclass classification techniques through both the SVM and NB classifiers. NB classification algorithm finds the maximum likelihood class given the data, regardless of the number of classes on which we are classifying. Therefore, the NB classifier model is already robust to multiclass classification. For SVM, however, we must extend the binary classification that a linear SVM performs to the case of multiple classes. We do this either by creating a “one vs. one” classifier, in which an SVM is trained to classify between all pairs of classes, and classify the data point as the majority vote of these classifiers. The other method for extending SVM to multiclass classification is to train the SVM for each class against the rest of the classes, in a “one vs. rest” scheme. In comparison of these two schemes in the cases where we have C classes, the “one vs. one” classifier would need $\binom{C}{2}$ classifiers to be trained, whereas the “one vs. rest” scheme would only require C classifiers trained.

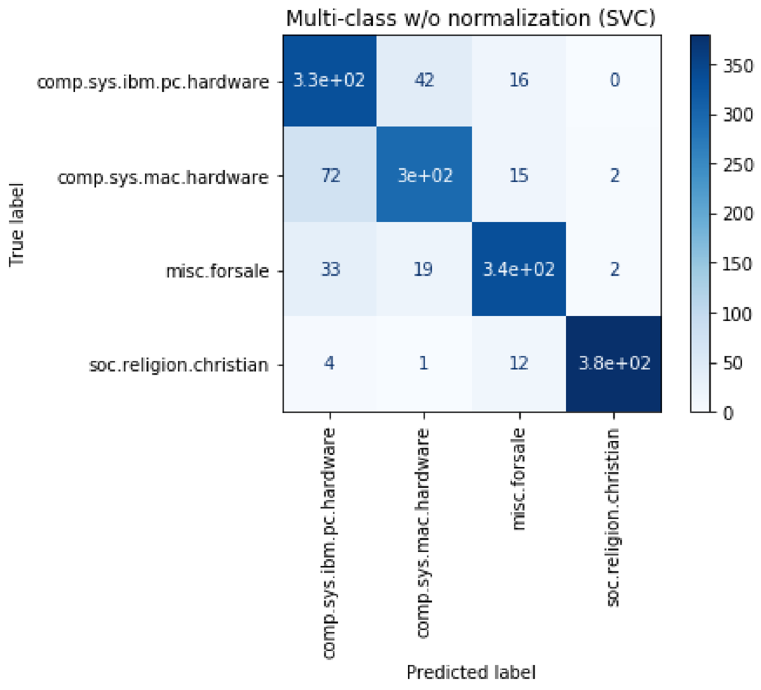
We trained a multiclass NB classifier, a “one vs. one” SVM classifier, and a “one vs. rest” SVM classifier on the training data, and evaluated the classifiers on the test dataset. Using a Pipeline and GridSearch, we checked each classifier’s accuracy against one another with a 5-fold cross-validation. Below are the results of our comparison:

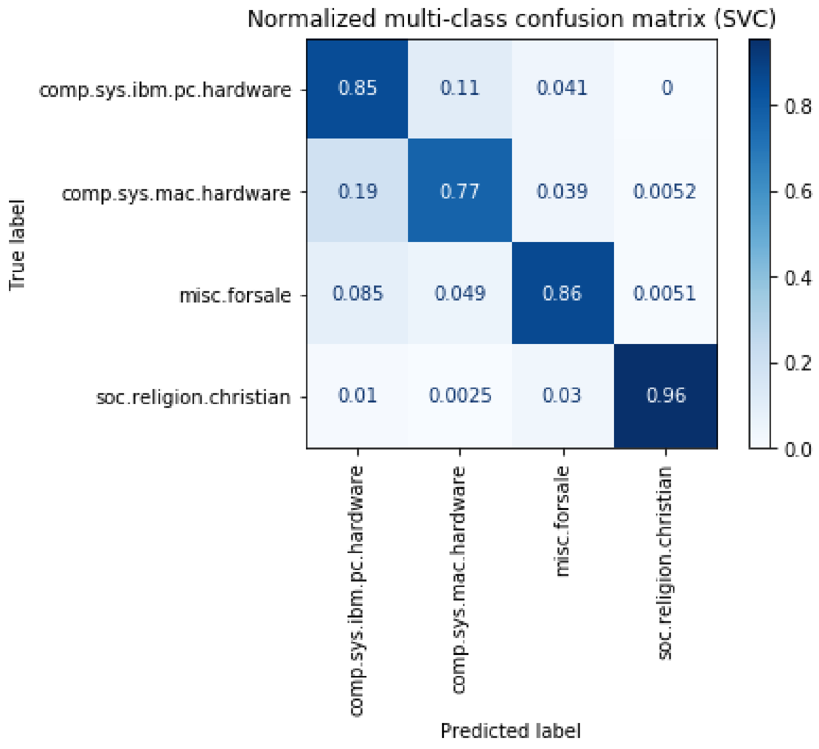
Classifier	Type	Training Accuracy
LinearSVC(loss='hinge',C=104.81)	One vs Rest	0.871565
SVC()	One vs One	0.882428
GaussianNB()	Naive Bayes	0.729073

As can be seen by our results, the SVC (one against one) classifier performed the best of the 3 classifiers for multi-class classification purposes. A summary of the performance metrics for these classifiers is given below:

SVC() Classifier

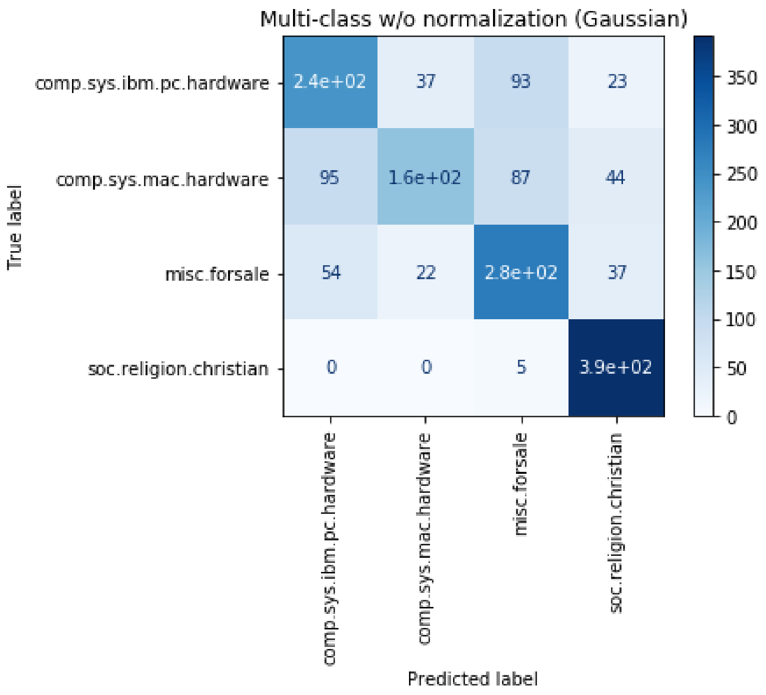
Accuracy:	0.860702875399361
Precision:	0.860702875399361
Recall:	0.860702875399361
f1 score:	0.860702875399361

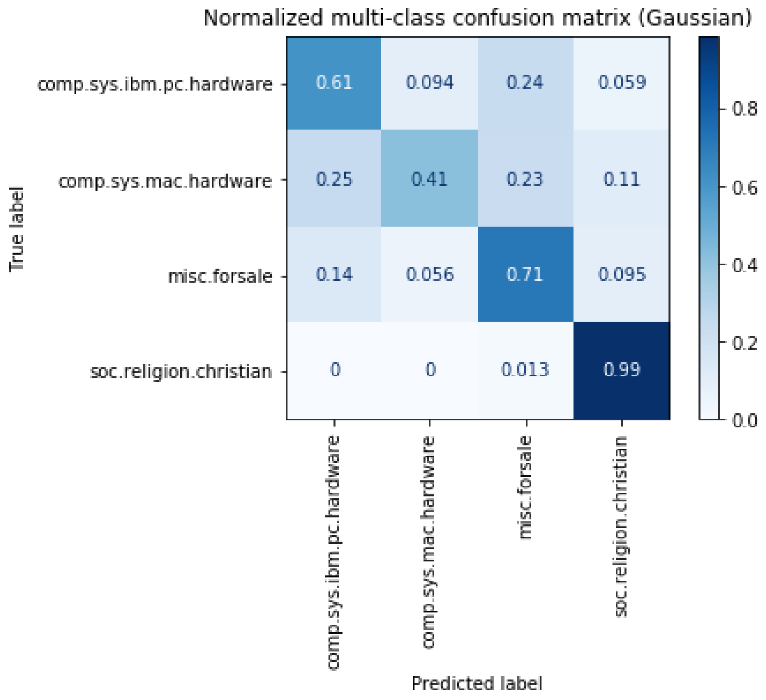




GaussianNB() Classifier

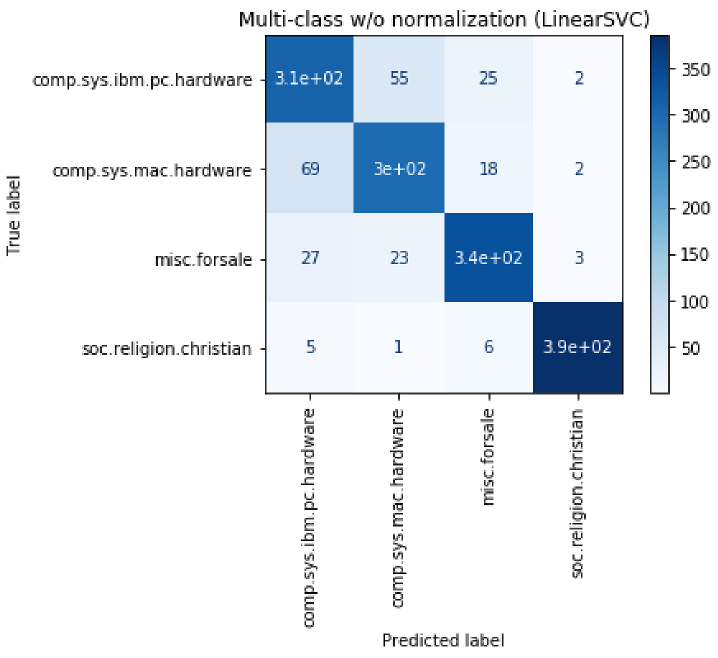
Accuracy:	0.6824281150159744
Precision:	0.6824281150159744
Recall:	0.6824281150159744
f1 score:	0.6824281150159744

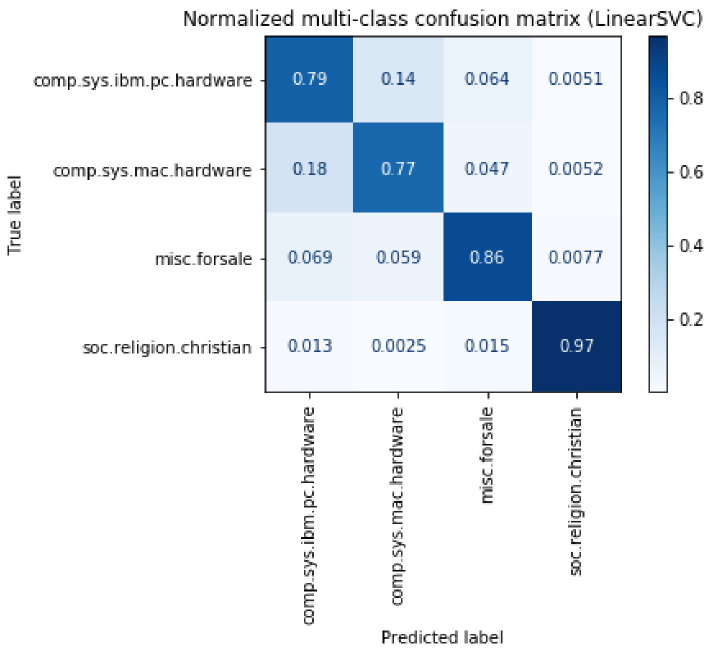




LinearSVC(loss='hinge',C=104.81131341546852) Classifier

Accuracy:	0.8492012779552716
Precision:	0.8492012779552716
Recall:	0.8492012779552716
f1 score:	0.8492012779552716





We see from the results that although the classifiers do not perform as well as their binary classification counterparts, they still perform reasonably well. From the results, we determine that the SVMs perform better than the NB for multiclass classification. Furthermore, we see that the one vs. one SVM classifier performs better than the one vs. rest SVM classifier. Overall, this demonstrates the robustness of the SVM and NB techniques for multiclass classification.

Conclusion

In short, we have demonstrated the training and evaluation of various classifiers on training and testing data sets with features extracted from a large collection of text documents. We assessed and compared many different classifiers and data mining techniques for processing data in a pipeline. For various hyperparameters and parameters in our models, we performed an exhaustive grid search to determine the best combination of parameters methods to achieve the best performance in classification of textual data. Overall, the feature extraction, dimensionality reduction, and classification techniques discussed in the project are robust and can be used for a variety of data.