ECE 219

Large-Scale Data Mining: Models and Algorithms

WINTER 2020

Project 1

Classification Analysis on Textual Data

Akshay Joshi
Ege Cetintas
Rohan Dutta
Shihan Gao

INTRODUCTION

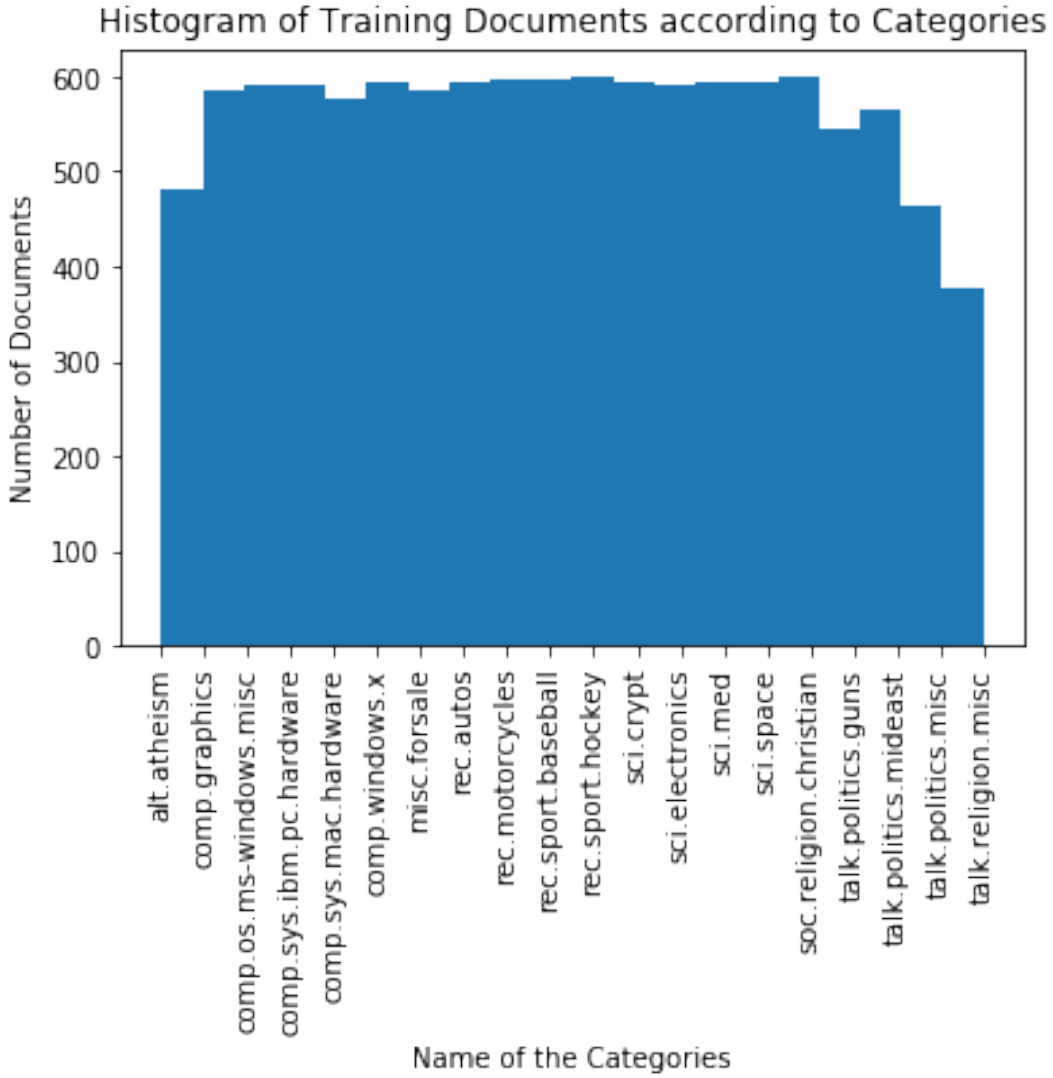
Text classification is a process of categorization of textual data into generic labels. Further, statistical classification refers to the action of predicting the category a textual data, which can be a sentence, a paragraph or even sets of documents, belongs to depending on a predefined training set of documents. In this project we performed classification of textual data on the '20 Newsgroups dataset'. We implemented dimensionality reduction techniques like Non-Negative Matrix Factorization and Latent Semantic Indexing to reduce the dimension of the dataset to improve speed. We used methods like Support Vector Machines, Logistic Regression, and Naïve Bayes Algorithm to train the classifier. Classification performance was evaluated using measures like accuracy, recall, precision, ROC, and confusion matrix for each case.

THE DATASET

The dataset that we used for the execution of this project is the '20 Newsgroup' dataset. It is a collection of approximately 20000 newsgroup documents, partitioned evenly across 20 different newsgroup categories. For this project's purpose, we used documents categorized into 8 categories viz. comp.os.ms-windows.misc, comp.sys.ibm.pc.hardware, comp.sys.mac.hardware, comp.graphics, rec.autos, rec.motorcycles, rec.sport.baseball, rec.sport.hockey. These categories are selected since they are closely related to each other. Following table shows the next division of these categories into two different categories.

Computer Technology	**Recreational Activity**
comp.os.ms-windows.misc	rec.autos
comp.sys.ibm.pc.hardware	rec.motorcycles
comp.sys.mac.hardware	rec.sport.baseball
comp.graphics	rec.sport.hockey

In a classification problem, it is imperative to check the balance of the dataset, which means to check whether the data is evenly spread over the different categories or not. Accordingly, we need to adjust the penalty function, the parameters, or the weights in order to make the model not biased towards a particular set of categories. To check this, we plotted a histogram of number of training documents for each of the 20 categories. The plot is attached below. In our case, the categories that we considered have even distribution. Therefore, we need not adjust the parameters to fit with respect to the number of documents.



FEATURE EXTRACTION

This is an important step in any type of classification. Feature extraction entails extracting important information from the data which proves to be crucial in classification. In text classification particularly, we need to have a representation of documents. We used a 'Bag of Words' model for the purpose. It is a simplifying representation used in natural language processing and information retrieval. In this model, a document is represented by the frequency of occurrence of each word. We used Term Frequency-Inverse Document Frequency commonly known as TF-IDF as a metric to evaluate how important a word is to a document in a collection or corpus. This relates to the importance of a word being directly proportional to the number of times it appears in the document but is offset by the number of times the word appears in the entire corpus. For example, if the corpus is all about soccer, then words like 'shoes', 'football', 'kick', etc. will be present in almost every document and thus these kinds of words would not be distinguishing factors in classification. TF-IDF is composed of two terms. TF and IDF. TF of term 't' is calculated by TF(t) = (Number of times term 't' appears in a document) / (Total number of terms in the document). While computing TF, all terms are considered equally important. However it is known that certain terms, such as 'is', 'of', and 'that', may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the IDF as IDF(t) = log(Total number of documents / Number of documents with term t in it).

Further, we performed lemmatization to obtain root forms of derived words. For example, after lemmatization, 'crying' and 'cries' are mapped to a single root word 'cry'. Thus, lemmatization is useful to reduce the number of unnecessary words which do not provide any more information than already present. We also removed terms that are numbers. Tokens are basically individual units of meaning that we are operating on.

Tokenization is the process of breaking down text document apart into those pieces. In text analytics, tokens are most frequently just words. We implemented preprocessing of data by converting the collection of text documents to a matrix of token counts using sklearn's **CountVectorizer** module. We passed the tokenizer parameter as our lemmatized set of words with numbers and digits removed. We also removed English stop words which prove to be of little importance due to the

high frequency of occurrences in the documents. For example, ‘a’, ‘an’, ‘and’, ‘there’, etc. Also, we set the minimum document frequency as 3, i.e. the words with a minimum frequency of 3 were only considered as features. After preprocessing both the training data and the testing data by passing them to the CountVectorizer module, we obtained the count matrix of both the training dataset and the test dataset. We passed the count matrix into the **TfIdfTransformer** module and performed feature extraction. The module outputted the TF-IDF matrices of the datasets which are as shown in the figure below:

```
[5] print('The shape of the TRAIN TF-IDF matrix is: ', X_train_tfidf.shape)
    print('The shape of the TEST TF-IDF matrix is: ', X_test_tfidf.shape)
```

```
The shape of the TRAIN TF-IDF matrix is:  (4732, 11553)
The shape of the TEST TF-IDF matrix is:  (3150, 11553)
```

DIMENSIONALITY REDUCTION

The dimensions of the output TF-IDF matrices that we got after performing TF-IDF transformation were of a much higher order, thousands to be precise as evident by the above screenshot. It is known that higher number of dimensions, or in other words, more number of features lead to reduction in the model performance. This phenomenon is known as the curse of dimensionality. The more dimensions you add to a data set, the more difficult it becomes to predict certain quantities. Therefore it is necessary to reduce the dimension of the dataset to improve the performance and prediction value. In this project we used two methods to reduce the dimensionality viz. Latent Semantic Indexing (LSI) and Non-negative Matrix Factorization (NMF).

LSI

We computed the Singular Value Decomposition to find the left eigenvectors and the right eigenvectors corresponding to top 50 largest singular values of each of the TF-IDF matrices. The 50 largest singular values correspond to the top 50 distinguishing features that contribute the most to the classification. Therefore, we perform SVD on a matrix X to find its decomposition $\boldsymbol{U\Sigma V^T}$ where $\boldsymbol{U}$ consists of left

eigenvectors, V^T contains right eigenvectors, and Σ consists of singular values placed in a decreasing order on its diagonals. We used the first 50 columns of U and V. Using the TruncatedSVD module, we performed LSI to reduce the dimension of the TF-IDF matrices to map each document (i.e. each row of the matrix) to a 50 dimensional vector. The result obtained was as follows:

```
print('Shape of TRAIN TF-IDF matrix after LSI:',X_svd_train.shape)
print('Shape of TEST TF-IDF matrix after LSI:',X_svd_test.shape)
```

```
Shape of TRAIN TF-IDF matrix after LSI: (4732, 50)
Shape of TEST TF-IDF matrix after LSI: (3150, 50)
```

NMF

Non-Negative Matrix Factorization can also be used to reduce the dimensionality. The matrix X is decomposed to two non-negative matrices W and H such that || X - WH $||^2$F is minimized. Then we used W as the dimension reduced data and in the fit step we calculated both W and H. We performed this factorization to break down X's rows which are the number of documents into a linear combination of r topics. Thus, using the NMF module, we calculated the reduced TF-IDF matrices and the result was as follows:

```
print('Shape of TRAIN TF-IDF matrix after NMF:',W_train.shape)
print('Shape of TEST TF-IDF matrix after NMF:',W_test.shape)
```

```
Shape of TRAIN TF-IDF matrix after NMF: (4732, 50)
Shape of TEST TF-IDF matrix after NMF: (3150, 50)
```

As evident from the above snapshots, we were successfully able to reduce the dimensionality of the dataset to 50 by both methods. Further, we calculated scores for both methods.

```
[10] print('The LSI error is:',LSI_error)
     print('The NMF error is:',NMF_error)

     The LSI error is: 4037.2773053750702
     The NMF error is: 4079.1718544228806
```

Thus, the NMF error is higher than the LSI error. Basically, both the methods are solutions to the optimization problem. The main difference between NMF and LSI is that the NMF method introduces another constraint of non-negativity. Due to this additional constraint, the tolerance for error is increased. Due to this, the error increases for the NMF method.

Classification:

There are various parameters that are used to judge classification result's quality. Confusion matrix, precision, recall, accuracy, F-1 score, etc are some of these parameters. Confusion matrix consists of parameters like True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN). TPs are the correctly predicted positive values which means that the value of actual class is yes and the value of predicted class is also yes. TNs are the correctly predicted negative values which means that the value of actual class is no and value of predicted class is also no. FPs are values When actual class is no and predicted class is yes. FNs are values when actual class is yes but predicted class in no. Accuracy is defined as the ratio of correctly predicted observations to total number of observations. Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. Recall is the ratio of correctly predicted positive observations to all observations in actual class. ROC curve is obtained by plotting the True Positive Rate against the False Positive Rate at various thresholds.

We divided the dataset into two categories viz. Computer Technology and Recreational Activity. We used the SVM classifier first to implement binary classification. We used two values of tradeoff parameter γ to compare the results. We trained one SVM with γ=1000 (hard margin) and another with γ=0.0001 (soft margin).

The SVM with γ=1000 (Hard Margin SVM) performed better than the SVM with γ=0.0001 (Soft Margin SVM)
The Soft Margin SVM is more tolerable to error/misclassification. This leads to reduction in performance.

Experiments with the Gamma Parameter

For Gamma = 1000:

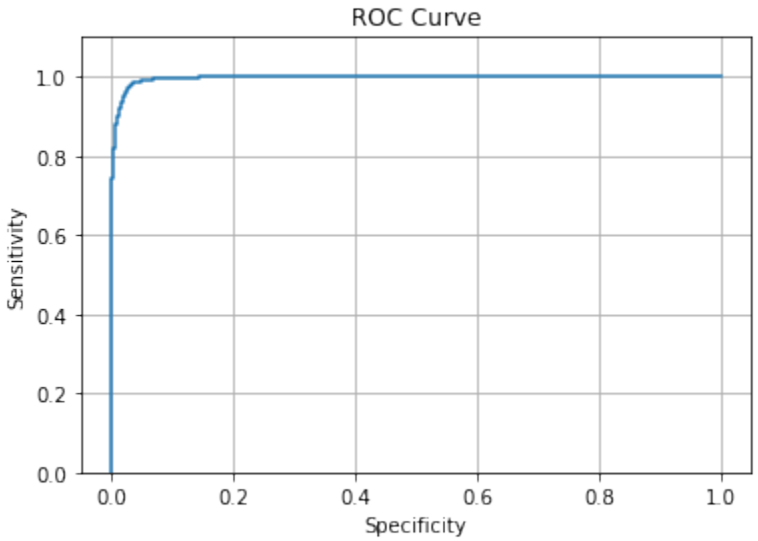
Confusion Matrix:

		Predicted	
		0	1
True	0	1504	56
	1	29	1561

Performance Metrics:

Precision	Recall	F-1 Score	Accuracy
0.9654	0.9818	0.9734	0.9730

ROC Curve:



ROC Curve for the case of Gamma = 1000

stemmingFor Gamma = 0.0001:

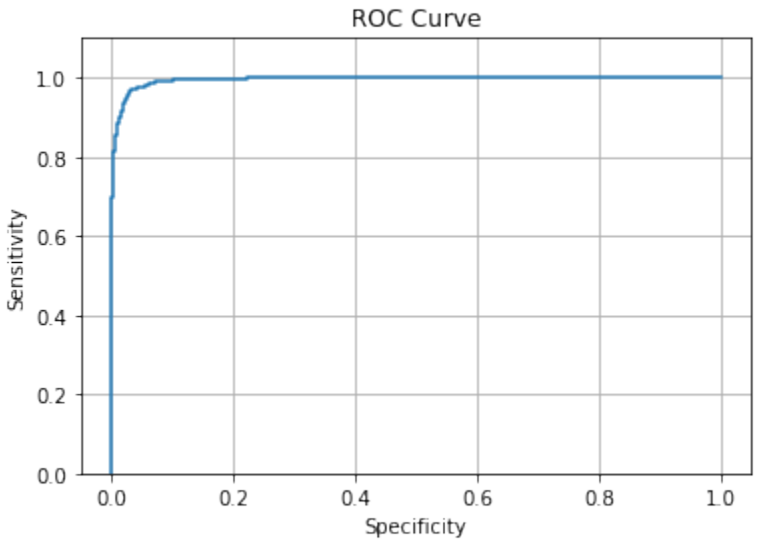
Confusion Matrix:

		Predicted	
		0	1
True	0	597	963
	1	0	1590

Performance Metrics:

Precision	Recall	F-1 Score	Accuracy
0.5048	1.0	0.7676	0.6943

ROC Curve:



ROC Curve for the case of Gamma = 0.0001

The ROC curve of the Soft Margin SVM looked good but it conflicts with other metrics like precision, F1 score and accuracy as is evident from the above table of Performance Metrics.

The best value Gamma = 100:

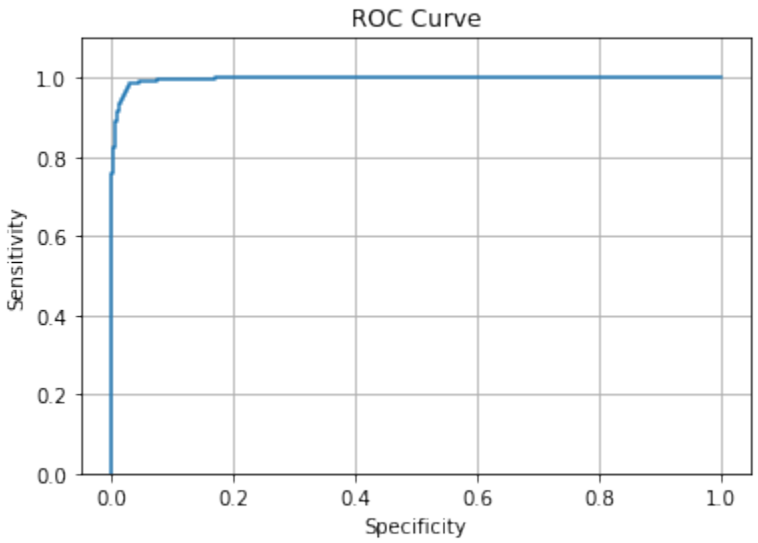
Confusion Matrix:

		Predicted	
		0	1
True	0	1508	52
	1	27	1563

Performance Metrics:

Precision	Recall	F-1 Score	Accuracy
0.9678	0.9830	0.9753	0.9749

ROC Curve:



For comparison purposes, when we look at ROC curves, it can be observed that ROC curves look very similar. The area under the curve and the shape of the curves look very promising however for the soft margin case the other metrics such as *accuracy* conflicts with the ROC curve. This little experiment shows that all of the figure of merits (ROCs, accuracy, precision, recall etc.) need to be considered all together. Otherwise, some metrics could be misleading.

Logistic Regression

1. No regularization

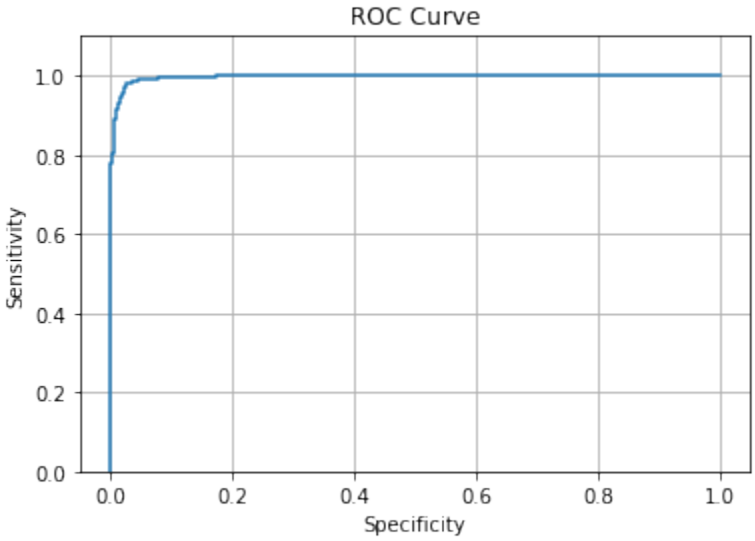
   Logistic Regression Model maps the linear regression results into probabilities ranging from 0 to 1. It can deal with both discrete and continuous variables. It assumes a parametric form for the distribution P(Y|X), where X is features and Y is the classification target; then directly estimates its parameters from the training data. To implement Logistic Regression model, we import class 'LogisticRegression' from package and here we use C=1e13 for no regularization. Since the C value is inversely proportional to the gamma value choosing C value actually means not using regularization at all.

precision	recall	f1	accuracy	roc score
0.9660	0.9824	0.9741	0.9736	0.9966

The confusion matrix:

		Predicted	
		0	1
True	0	1505	55
	1	28	1562

ROC curve



2. L1 and l2 regularization
   For Logistic Regression with regularization, we used both l1 and l2 norm regularization; and in each norm regularization, we changed parameter C in the range of 1e-3 to 1e3 with 5-fold cross-validation on the dimension-reduced-by-svd training data.

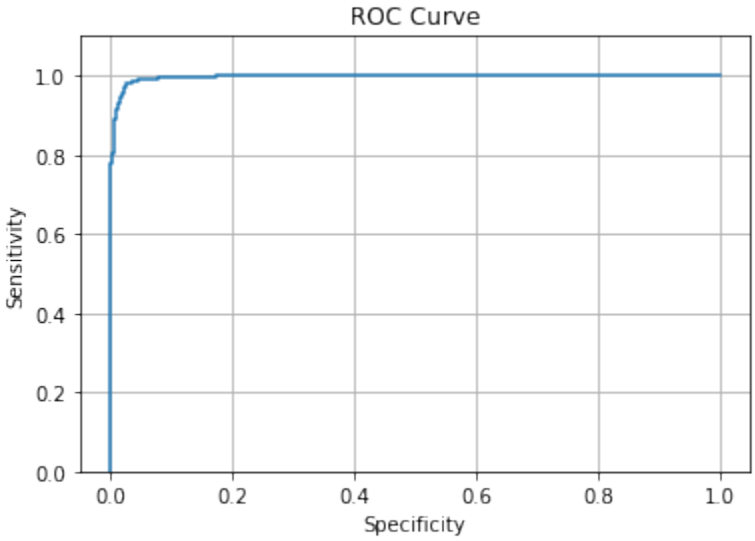
L1 regularization results:

precision	recall	f1	accuracy
0.9678	0.9830	0.9754	0.9749

Confusion matrix:

		Predicted	
		0	1
True	0	1508	52
	1	27	1563

ROC curve:



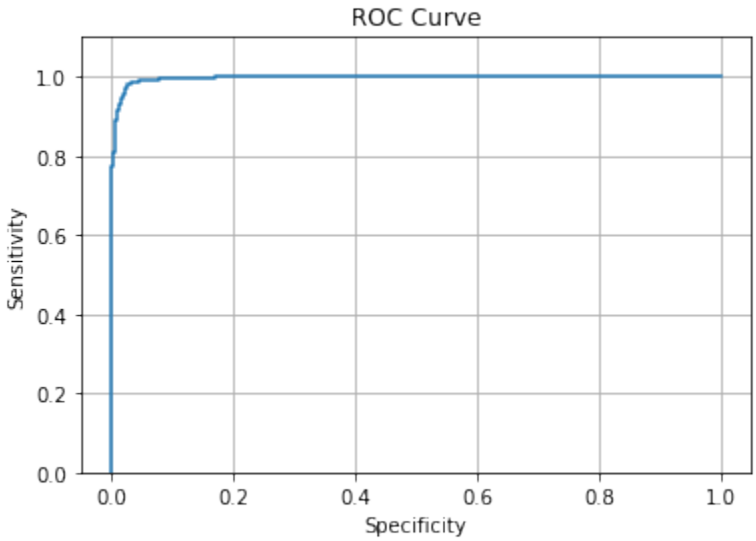
L2 regularization results:

precision	recall	f1	accuracy	roc score
0.9696	0.9836	0.9766	0.9762	0.9966

Confusion matrix:

		Predicted	
		0	1
True	0	1511	49
	1	26	1564

ROC curve:



As we compare the performance (accuracy, precision, recall and F-1 score) of 3 logistic classifiers using test data, l1 norm and l2 norm can get similar best accuracy. We can minimize the loss for the training data but a standard least squares function tends to have some variance. In this case, our model may not generalize well for a data set different than its training data. The aim of regularization is to reduce the variance of the model while without big increase in its bias. Theoretically, regularization is to add a penalty factor to loss function and l1 norm method is to add an item associated with the l1 norm of parameter vectors, so does l2 norm. L1 norm can make weight matrix sparse to realize feature selection, while l2 norm tends to make all the weights smaller to avoid overfitting. Different techniques are useful in different fields due to their differences in shrinking. The best regularization strength for l1 is C=10.0 and C=1000.0 for l2. That is not a good idea not to regularize as the training data could be overfitted and if the training data is noisy, the overfitted data would not perform smoothly on the test cases. L1 and L2 regularizations are used for different purposes. They both have pros and cons. In summary, L1 could be used as a feature selection method which is important for some applications yet it is computationally expensive. On the other hand, L2 regularization can be a solution to the problem of multicollinearity by keeping all

the variables. For the discussion of test accuracy, as the gamma parameter decreases, the test accuracy increases up to a certain point. However, the increase in the test accuracy gradually stops and test accuracy reaches to a maximum value.

Both logistic regression and linear SVM are trying to classify data points using a linear decision boundary. The way linear SVM does is to maximize the margin, i.e. it penalizes mis-classified points linearly in distance to the learned hyperplane. But logistic regression tries to find a hyperplane that minimizes the loss according to the distribution. It penalizes mis-classified points linearly in the distance to the separating hyperplane. Logistic regression might be overfitting, because it is not as accurate on the test data as linear SVM. In comparison, the logistic regression is more sensitive to outliers because logistic regression's loss function grows faster than linear SVM. Both linear SVM and logistic regression perform well on the train data. Linear SVM basically concerned with the support vectors while finding the best hyperplane.

Naïve Bayes

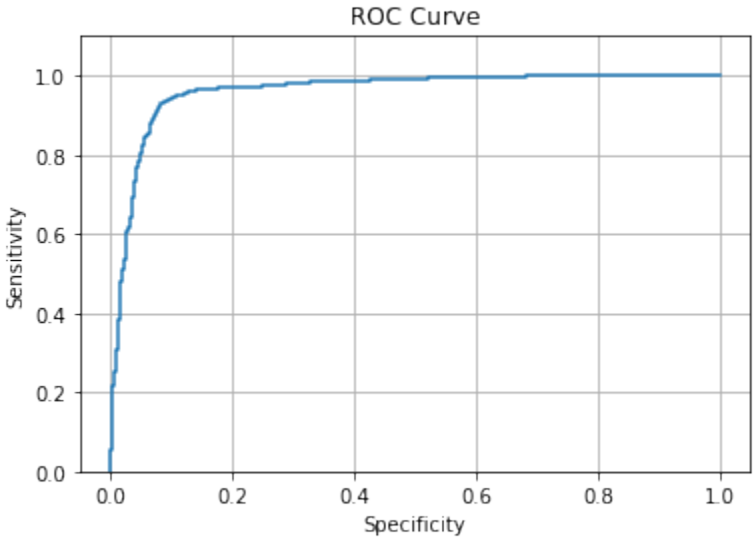
Naïve Bayes classifiers include MultinomialNB, BernoulliNB, and GaussianNB and we import GaussianNB to do the binary classification. Naïve Bayes classifiers assume that features are statistically independent of each other when conditioned by the class the data point belongs to, to simplify the calculation for the Maximum A Posteriori (MAP) estimation of the labels.

precision	recall	f1	accuracy	roc curve
0.9103	0.9384	0.9241	0.9222	0.9605

confusion matrix:

		Predicted	
		0	1
True	0	1413	147
	1	98	1492

ROC curve:



Grid Search for Parameters

We build a composite estimator with a preprocessing pipeline that combine term extraction, dimensionality reduction, and normalization sequentially. The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid and here we use the attribute 'best_params_' to get the best pipeline estimator. We get two tables with different choice of data loading, one with removing "headers" and "footers" and one without.

Procedure	**Options**
Loading Data	remove "headers" and "footers" vs not
Feature Extraction	`min_df = 3` vs 5; use lemmatization vs not
Dimensionality Reduction	LSI vs NMF
Classifier	SVM with the best γ previously found vs Logistic Regression: L1 regularization vs L2 regularization, with the best regularization strength previously found vs `GaussianNB`
Other options	Use default

We built a pipeline using the Pipeline feature of sklearn as follows:

```
pipeline = Pipeline([('count_vectorization',CountVectorizer(min_df=3)),\
                     ('tfidf',TfidfTransformer()),('dim_red',TruncatedSVD()),('classifier',SVC())])
```

The pipeline consisted of the CountVectorizer followed by the TfidfTransformer, TruncatedSVD (Dimensionality Reduction) and finally classifier.

We then used GridSearchCV to find the best parameter using a param grid that spanned the above parameters:

```
pipelinemodel = GridSearchCV(pipeline,cv =5,param_grid = param_grid,scoring='accuracy')
pipelinemodel.fit(train_dataset.data,y_train)
print(pipelinemodel.best_params_)
```

```
{'classifier': LogisticRegression(C=100, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l1',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False), 'classifier__C': 100, 'classifier__penalty': 'l1', 'classifier__solver': 'liblinear', 'count_vectorization': CountVectoriz
                 dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                 lowercase=True, max_df=1.0, max_features=None, min_df=3,
                 ngram_range=(1, 1), preprocessor=None, stop_words=None,
                 strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                 tokenizer=<function no_num_tokenizer_no_lemmatization at 0x7f960b0a39d8>,
                 vocabulary=None), 'count_vectorization__min_df': 3, 'count_vectorization__tokenizer': <function no_num_tokenizer_no_lemmatization at 0x7f960b0a3
               random_state=None, tol=0.0), 'dim_red__n_components': 50}
```

We used this pipeline for two different datasets (1) without headers and footers and (2) with headers and footers and scored the model on test accuracy. Below are the results:

Result table1: without “headers” and “footers”

Feature Extraction	min.df = 5, no lemmatization
Dimensionality Reduction	LSI
Classifier	Logistic regression with l1 regularization

Result table2: with “headers” and “footers”

Feature Extraction	min.df = 3, no lemmatization
Dimensionality Reduction	LSI
Classifier	Logistic regression with l1 regularization

Multiclass Classification

So far, we have been dealing with classification with two target values. We now explore the multiclass classification techniques by different algorithms.

The dataset is taken from the following subclasses: 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'misc.forsale', 'soc.religion.christian'.

For SVM, however, one needs to extend the binary classification techniques when there are multiple classes. There are two ways to extend the binary classifier, one is named “One vs One” and another is “One vs the rest”. “One vs One” means to perform a one versus one classification on all pairs of classes, and given a document class is assigned with the majority, which means training $\frac{n\times(n-1)}{2}$classifiers (n is the number of classes). Then test data with these two classifiers and classify each pair of the test data into the class with the highest vote. While “One VS the Rest” method reduces the number of classifiers by fitting one classifier per class, which reduces the number of classifiers to be learned. In this problem, we use the best C parameter value from the best SVM model.

1. Multiclass classification svm one vs rest
   To realize this method, we set the parameter of SVC ‘ovr’:
   ‘multi_class_svm=SVC(decision_function_shape='ovr',C=10.0)’
   the result of the confusion matrix and the accuracy, recall, precision and F-1 score of our classifier is:

```
Shape of X_train:  (2352, 8704)
Shape of X_test:  (1565, 8704)
NMF error:  1963.6201865812488
Reduced training shape using NMF:  (2352, 50)
Reduced test shape using NMF:  (1565, 50)
LSI error:  1938.02074545448
Reduced training shape using LSI:  (2352, 50)
Reduced test shape using LSI:  (1565, 50)
                         precision    recall  f1-score   support

comp.sys.ibm.pc.hardware       0.81      0.85      0.83       392
   comp.sys.mac.hardware       0.84      0.81      0.82       385
            misc.forsale       0.89      0.88      0.89       390
  soc.religion.christian       0.99      0.98      0.99       398

                accuracy                           0.88      1565
               macro avg       0.88      0.88      0.88      1565
            weighted avg       0.88      0.88      0.88      1565
```

   Multiclass classification svm one vs one

To realize this method, we set the parameter of SVC ‘ovo’:
‘multi_class_svm=SVC(decision_function_shape='ovo',C=10.0)’

	precision	recall	f1-score	support
comp.sys.ibm.pc.hardware	0.81	0.85	0.83	392
comp.sys.mac.hardware	0.84	0.81	0.82	385
misc.forsale	0.89	0.88	0.89	390
soc.religion.christian	0.99	0.98	0.99	398
accuracy			0.88	1565
macro avg	0.88	0.88	0.88	1565
weighted avg	0.88	0.88	0.88	1565

Generally in multiclass classification tasks, SVM (both “One VS One” and “One VS Rest”) perform good with accuracy of 0.88 but “one vs one” trains less no of classifier and hence is faster overall and hence is usually preferred, i.e., single classifier in “one vs one” uses subset of data. But theoretically, for particular classes, “on vs one” is less prone to dataset imbalance.

2. Multiclass classification Gaussian Naïve Bayes
   Regardless of the number of classes, Naïve Bayes algorithm finds the class with maximum likelihood given the data. Therefore, the code is the same as the binary classification part.

	precision	recall	f1-score	support
comp.sys.ibm.pc.hardware	0.61	0.66	0.63	392
comp.sys.mac.hardware	0.69	0.43	0.53	385
misc.forsale	0.63	0.77	0.70	390
soc.religion.christian	0.93	0.99	0.96	398
accuracy			0.72	1565
macro avg	0.72	0.71	0.71	1565
weighted avg	0.72	0.72	0.71	1565

In conclusion, for our dataset in multiclass classification tasks, SVM (both “One VS One” and “One VS Rest”) perform better than Naïve Bayes.