EE 219: Large-Scale Data Mining

Project 1

Chinni Sarat Bhargava 805219546
Eden Haney 005221845
Osama Hassan 705221917
Shannon Sabino 905227598

January 24, 2019

Contents

Introduction

The goal of this project is to apply and compare different classification techniques.

Classification is categorized as a type of supervised machine learning. [1] In a broad sense, classification is the branch of machine learning that aims to classify data correctly, given no prior knowledge of a datum's actual classification. In a general classification problem, we begin with a dataset x_{given}, and its corresponding classification y_{given}. We refer to y_{given} as the"target". This makes sense when we view the problem as a mapping from x-space to y-space. The goal of classification then is to find the function f that maps $f : x$. Consider a small example.

Let's call our $x_{given} = [-1, 3, 2, -2]$ and our $y_{given} = [1, 0, 0, 1]$. We can somewhat trivially find a function mapping between these two data sets, such as:

$$f = \begin{cases} 1, & x < 0 \\ 0, & x >= 0. \end{cases} \tag{1}$$

However, this isn't the only mapping that fits this data. We could just as easily have called our function:

$$f = \begin{cases} 1, & x >= 2 \\ 0, & x < 2. \end{cases} \tag{2}$$

So how do we find the best mapping? To answer this, we first must define what makes a mapping "good".

Metrics

Common metrics used in binary classification are accuracy, precision, recall and and F-1, as well as the confusion matrix. These four metric each rely on the true (or correct) positive and negative (TP and TN) classifications, as well as the reported positive and negative (P and N) classifications. In binary classification, TP is just the number of accurate classifications of the "positive" class, which is often numerically labeled as class "1". In other words the number of times that data with a classification of "1" was predicted as "1". In contrast to this, FP is the number of times a datum from the "negative" class, class "0" or "-1", is predicted as "1". TN and FN are the negative class equivalents of TP and FP.

Accuracy

Accuracy is a measurement of how high the TP and TN are compared to the P and N,

$$ACC = \frac{TP + TN}{P + N}. \tag{3}$$

This means that for TP=P (perfect positive classification) and TN=N (perfect negative classification), the accuracy will be 1. To see the drawbacks of accuracy, first consider a perfectly balanced dataset, with P=N. If our model only returns positive classifications and never classifies anything as negative, our accuracy will be 50%. Now consider that the dataset is actually very skewed towards positive, for example 90% positive classifications. A model that only predicts positive will still have a 90% accuracy.

Confusion matrix

Partly because of this, accuracy is generally not the preferred performance measure for classifiers, especially when you are dealing with skewed datasets. A better alternative to evaluate the performance of a classifier is to look at the confusion matrix. Each row in a confusion matrix represents an actual class, while each column represents a predicted class.

Predicted→ Actual↓	Positive	Negative
Positive	TP	FP
Negative	FN	TN

A perfect classifier would have only true positives and true negatives, so its confusion matrix would have nonzero values only on its main diagonal.

Precision

The confusion matrix gives you a lot of information, but sometimes you may prefer a more concise metric. An interesting one to look at is the precision of the classifier:

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

The precision is the ratio of correct positive predictions to all positive predictions, and is therefore sometimes called the positive predictive value (PPV). We can see that precision might still be an issue for skewed datasets and a biased model. We can alternatively take a look at a similar metric, the negative predictive value (NPV), so that we can get a sense of how specifically the negative data is being classified.

Recall

A trivial way to have perfect precision is to make one single positive prediction and ensure it is correct - this results in a 100% precision. This would not be very useful, since the classifier would ignore all but one positive instance. To balance precision's shortcomings, it is typically used along with another metric named recall, which is also called sensitivity or true positive rate (TPR). This is the ratio of positive instances that are correctly detected by the classifier:

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{P} \tag{5}$$

F1

It is often convenient to combine precision and recall into a single metric called the F1 score. It is particularly helpful if you need a simple way to compare two classifiers. The F1 score is the harmonic mean of precision and recall. Whereas the regular mean treats all values equally, the harmonic mean gives much more weight to low values. As a result, the classifier will only get a high F1 score if both recall and precision are high.

$$F1 = 2\frac{(PPV)(TPR)}{PPV + TPR} = 2\frac{(Precision)(Recall)}{Precision + Recall} \tag{6}$$

It is worth noting that increasing precision reduces recall, and vice versa. This is called the precision/recall tradeoff. [1].

ROC

The receiver operating characteristic (ROC) curve is another common tool used with binary classifiers. It is very similar to the precision/recall curve, but instead of plotting precision against recall, the ROC curve plots the true positive rate (recall) against the false positive rate (FPR) for different thresholds [1]. The FPR is the ratio of negative instances that are incorrectly classified as positive, and is also equal to one minus the true negative rate. The true negative rate (TNR) is the ratio of negative instances that are correctly classified as negative. The TNR is also called specificity. Hence, we can summarize the ROC curve as a plot of recall versus (1 – specificity).

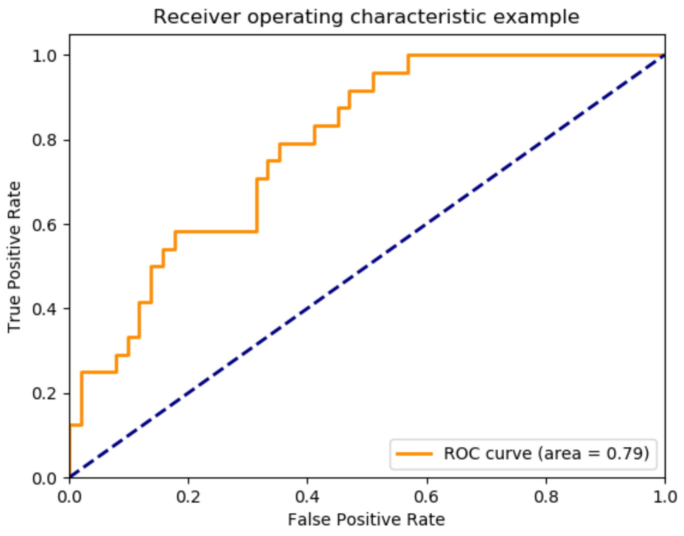


Figure 1: ROC Example from [2].

Here again there is a trade-off: the higher the recall (TPR), the more false positives (FPR) the classifier produces. The dotted line represents the ROC curve of a purely random classifier; a good classifier stays as far away from that line as possible (toward the top-left corner). One way to compare classifiers is to measure the area under the curve (AUC). A perfect classifier will have a ROC AUC equal to 1, whereas a purely random classifier will have a ROC AUC equal to 0.5.

General Procedure

Now that we have given ourselves a framework for evaluating the different classification models that we will create, we can begin creating them! Before we dive in, let's first discuss the general process that is followed for creating a classifier.

Creating a classifier begins with collecting the raw data. (Question 1). Because we are working with text documents, we call this collection our corpus. This data must then be processed in some way to filter out the pieces we aren't interested in categorizing. In a numeric data set, for example, this may consist of removing all instances of “NaN” and “inf”.

Now we aim to make our data more “computable”. The text in our corpus has no meaning to our computer, so we aim to transform it into a more manageable matrix of numbers. (Question 2). Next, the resultant matrix is likely going to have a very high dimensionality, especially in image classification problems. Therefore, we further attempt to reduce redundancy and reduce our data set, while still retaining the important information. (Question 3,4).

With the dataset reduced to a computationally manageable size, we choose a classification algorithm and start training it on the training dataset. Finally, once our models have been trained, we use our test dataset to judge their performances. Most algorithms are going to have parameters and hyperparameters that need tuning over several iterations, so this process will rely heavily on the previously established metrics. (Questions 5,6,7,8).

Method

This project is done entirely in Python and relies upon the Scikit-Learn library, which contains tools for common machine learning tasks. [2]

Question 1

We begin this problem by importing the data set that we will be working with, the 20 newsgroups text dataset, from the SciKit-Learn library. This corpus consists of around 18000 newsgroup posts on 20 different topics, and splits its data into a “test” and ‘train” set by a posts date. [2]

It is important in classification to work with “balanced data”, or data with equal representation for each target, in our training set. Before we talk about why, let's take a look at the distribution of targets in our dataset.

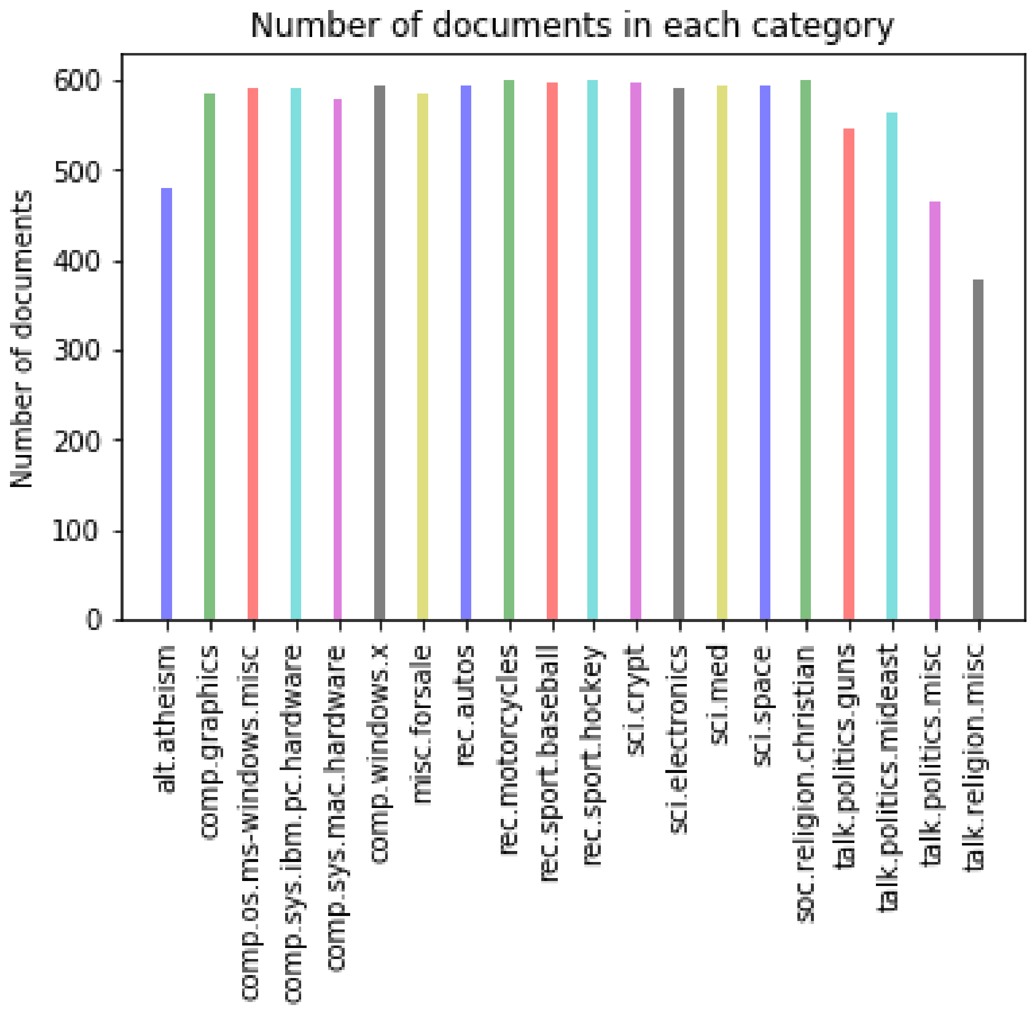


Figure 2: Frequency distribution of the classes in the 20 newsgroups dataset.

None of these targets is especially unbalanced. For our particular situations, we mainly work with targets sets that are extremely well balanced (all of the “comp”, “misc”, “rec”, and “sci” labels). This helps to ensure that our classification algorithm does not learn to favor a target with more representation. Our test data is equally likely to be of any category, and we do not want to bias the prediction toward a target with a larger representation in the training set.

There are situations where this would not be the case. For example, let’s say we want to create a classifier for the color of a car seen in an image, and we want to implement it on a street light. This means that we will be performing our prediction on real-world cars that are driving by that street light, and real-life cars do not present an equal representation of colors. So what happens if we train our classifier using an equal amount of yellow car images and black car images? If our model generalizes well, then we still have great performance on what we predict from the streetlight camera! However, if our model is overfitting to the training data and has built its model around probability distributions of the classes, it will perform worse. [3] This means that classifying talk.religion.misc (the lowest category) against soc.religion.christian (the highest category) may or may not lead to problems, depending on the application and the model.

Question 2

Feature extraction is a key step in any classic machine learning task. In our project we need to obtain a feature vector representation of the document in a corpus. One simple approach is to use bag of words. In this method, we represent each document as count vector of the number of times each word occurred in the document. Clearly we are losing some information in this representation such as sequence information in our case, which can be solved by using bigram or N-gram models instead of using 1-gram model.

To reduce redundant information from words originating from the same root, lemmatization was implemented using "WordNet". For example, words such as "goes", "going", "went", and "gone" are all counted only once as "go". Similar transformations are done to all parts of speech (e.g. nouns, verbs, adjectives, adverbs).

Similarly, words that rarely occur do not add value in the classification task and are removed. In this project we removed words that occur less than 3 times (i.e. $min\_df = 3$)

There are a few words like "this, the, he" etc. which occur very frequently in each document, but are of no value for the classification task. Therefore, we remove them. These words are called stopwords. We used only the default "english" stopwords provided by CountVectorizer function as specified in the question 2 for the entire project.

Normalization step is done to the count of vocubalry words in each document. The purpose of this nomralization step is to sepcify the importance of a word to document in the corpus. Term Frequency-Inverse Document Frequency (TF-IDF)” metric was utilized to evaluate this importance

$tf(t, d)$ is the count a term in document d. $Idf(t)$ is inverse document frequency of a term. So the metric is penalizing frequently occurring terms and encouraging the less frequent terms.

$$tf - idf(d, t) = tf(t, d) X idf(t) \tag{7}$$

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

$$idf(t) = log \frac{n}{df(t)} + 1 \tag{8}$$

where n is the total number of documents,and df(t) is the document frequency.

The resulting shape of the TF-IDF matrices for the training and test datasets are reported in Table 1.

Dataset	TF-IDF Matrix Shape
Training Dataset	(4732, 16600)
Test Dataset	(3150, 16600)

Table 1: The shape of the TF-IDF matrices of the train and test subsets.

Question 3

After performing feature extraction the size of the term-document count vectors (TF-IDF vectors) are still too large for learning algorithms, which perform better on low-dimensional data. Note that the TF_IDF matrix is rather sparse as most documents only use a small subset of the total set of terms. We can utilize this sparseness to select a subset of more relevant terms and transform the features into a lower dimensional space.

In this project, we used two dimensionality reduction methods: Latent Semantic Indexing (LSI), and Non-negative Matrix Factorization (NMF), both of which minimize the error derived from the Frobenius norm of the difference between the low-dimensional approximation of the terms in each document and the original data. [4]

Latent Semantic Indexing (LSI)

LSI applies Singular Value Decomposition (SVD) to a document-term matrix, "creating a low-dimensional embedding of the data that is designed to capture semantic similarity of words." [7] To paraphrase, it uses the SVD to group words by their meanings and assumes that words with similar meanings will occur in similar documents.

LSI was designed to overcome a fundamental problem of dimensionality reduction techniques that attempt to match words of queries with words of documents.[5] The problem is that categories should be organized by conceptual content yet there may be many terms used to express such a concept. In addition, most words have multiple meanings, so some terms will literally match terms in documents that are not related.

To perform LSI a matrix containing word counts per document (columns represent unique terms and rows represent each document) is constructed from a large piece of text and singular value decomposition (SVD) is used to reduce the number of columns while preserving the similarity structure among rows. [6] The SVD allows the arrangement of the space to reflect major associative patterns in the data and ignore the smaller, less important influences. It is then possible to find the "distance" between each document vector and term vector.

If the original term-document matrix is represented by an SVD as below:

$$A = U\Sigma V^T \tag{9}$$

then Latent Semantic Indexing (LSI) can be represented as a reduced version:

$$A_k = U_k \Sigma_k V_k^T \tag{10}$$

LSI then minimizes:

$$\|X - U_K \Sigma_K V_k^T\|_F^2 \tag{11}$$

Non-Negative Matrix Factorization (NMF)

One concern with the LSI model is that the orthogonal U and V vectors may be negative while the data being represented is non-negative. One can argue that the learned dictionary will be interpreted easier if it is a sum of positive parts. Non-Negative Matrix Factorization (NMF) accounts for this by restricting the matrices to non-negative values. [8]

In addition, unlike using the SVD in LSI, the matrix factorization done is:

$$A_k = W_k H_k, \quad W_k, H_k \geq 0. \tag{12}$$

And the equation we are trying to minimize is:

$$\min_{W \in \mathbb{C}, Z \in \mathbb{R}^{L \times N}} \frac{1}{2} \sum_{i=1}^{N} \|x_i - W z_i\|_2^2 \quad \text{s.t. } W \geq 0, z_i \geq 0 \tag{13}$$

Because W is nonnegative, each column of W can be interpreted as a bag of words while the elements of H can be considered weights. We can interpret the basis elements as topics or set of words found simultaneously in different documents, while the weights in the linear combinations (the matrix H) assign the documents to the different classes.

Results

Let's first compare the sizes of our reduced matrices.

Matrix	Dimensions
Shape of train before	(4732, 16600)
Shape of test before	(3150, 16600)
Shape of train TF-IDF matrix	(4732, 16600)
Shape of test TF-IDF matrix	(3150, 16600)
SVD reduced training set shape	(4732, 50)
SVD reduced testing set shape	(3150, 50)

We truncated both the LSI and NMF representations of the original data to a linear combination of 50 base vectors. As you can see, LSI performed better here. One potential cause of this that NMF is a strictly non-negative linear combination so this may have resulted in more error on our dataset than LSI.

	LSI	NMF
Error from training set	3895.4186796933272	3940.342513932875
Error from testing set	2676.4912327221027	2689.0690267386613

Question 4: Support Vector Machine

A Support Vector Machine (SVM) is a very powerful learning model, which provides models to both linear and nonlinear classification. In addition, it provides solutions to regression and outlier detection [1].

In classification problems, SVM algorithms try to maximize the margin between classes. In order to achieve this, SVM can be used in either in hard-margin or soft-margin modes. In the hard-margin regime, the algorithm does not allow any of the training data to exist inside the margin. This makes this method very sensitive to outliers and might results in too narrow margin (see Figure 3 (right)) or makes the separation impossible (see Figure 3 (left)). Moreover, the hard-margin classification only works with linearly separable data.

The soft margin regime tries to tackle these issues by allowing some limited margin violations to keep enough margin. Based on the problem and the nature of the data, an optimum value can be achieved to solve this trade-off.

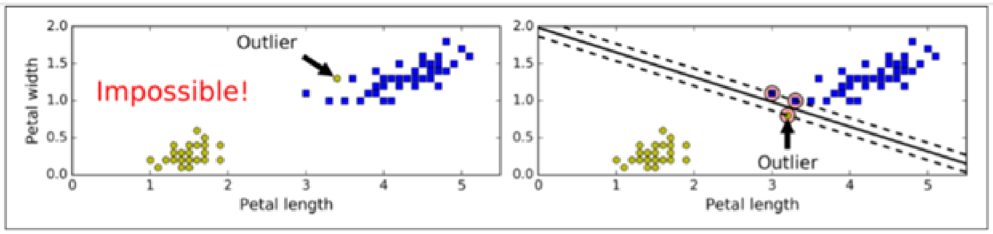


Figure 3: The strict requirement of excluding any outliers from the margin in the hard margin regime may result in too narrow margin (right) or makes the separation impossible (left). Reproduced form [1]

We used SVM as the first algorithm to build a binary classifier. The documents were categorized to be either “Computer Technology”, labeled as “1”, or “Recreational Activity”, labeled as “0”. The type of documents belonging to each category are summarized in Table 2.

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

Table 2: The main two categories and the documents' subcategories in each.

The tf-idf matrix was fed to a hard-margin SVM classifier with $\gamma = 1000$ and a soft-margin classifier with $\gamma = 0.0001$. The performance of both classifiers were compared in terms different metrics in Table 3.

Algorithm$\rightarrow$ Metric$\downarrow$	Hard-Margin	Soft-Margin
Confusion Matrix	$\begin{array}{\|cc\|} 1564 & 26 \\ 77 & 1483 \end{array}$	$\begin{array}{\|cc\|} 1590 & 0 \\ 1220 & 340 \end{array}$
Training accuracy	0.9797125950972105	0.6189771766694844
Test accuracy	0.9673015873015873	0.6126984126984127
Recall	0.9506410256410256	0.21794871794871795
Precision	0.9827700463883366	1.0
F-1 score	0.9664385793418051	0.3578947368421052

Table 3: Evaluating the performance metrics of hard-margin and soft-margin classifiers.

The accuracy of the soft-margin classifier is consistently less than the hard-margin classifier in both the training and test data. A more careful diagnosis to the soft-margin regime shows that it did not result in any false positives, which is evident in the unity precision and the zero false positives in the confusion matrix. However, this high precision was achieved at the cost of very low recall (~ 0.218). This means the soft-margin classifier tends to classify documents as negative, which is reflected in the very high number of false negatives (1220 times) while it is too conservative in classifying a document as positive (i.e. computer graphics). This resulted in a deterioration in the overall performance of the positive category classification which is quantified by the F-1 score (~ 0.35789). The reason that the soft margin classifier performed poorly is that it made too much margin violations because of is is very high γ value for the sake of maximizing the boundary margin. On the other hand, the hard-margin classifier resulted in some false positive (26 times). This means the classifier classified a “recreation activities” document as a “computer graphics” document 26 times as summarized in its confusion matrix. However, the classifier performed pretty well in detecting most of the “computer graphics” documents which is reflected in its high recall value ($\sim$ 0.95) and the lower number of false negatives compared with the soft-margin case. The overall performance of the positive category classification, which is quantified by the F-1 score ($\sim$ 0.967), is much better than the case of the soft-margin classifier.

In short, When we decrease the gamma parameter we allow classification errors to take place while decreasing the objective function that is norm of weight vector (try to increase the margin). When we kept the gamma to be very low, the classifier made a lot of errors (gamma very low so no penality) while decreasing the objective function that is norm of weight vector (tryto increase the margin). So at gamma at 0.001 we got very poor accuracy. So gamma has to be tuned shouldn’t be tooless or high.

To optimize the recall-precision trade-off, the classifier was trained with different γ values and the optimum value was $\gamma = 10$. The performance metrics of this optimized SVM classifier are shown in Table 4. The confusion matrix shows a better balance between the false positives and false negatives compared with the two earlier classifiers. This better balance is evident in the high values of both recall and precision which means that the classifier is able to detect most “computer graphics” documents and it rarely incorrectly classify “recreational activity” documents as positives. The overall performance that takes into account both the precision and recall of the classifier is quantified by the F-1 score, which is $\sim$ 0.970 which is slightly higher than the hard margin classifier’s F-1 score ($\sim$ 0.966).The test accuracy increased from 96.7% to 97.07% by using Optimal Threshold (C is 10) SVM instead of hard-margin SVM.

A comparison between the performance of the optimized classifier, the soft-margin classifier, and hard-margin classifier in terms of their ROC curves is shown in Figure 4. It is evident that that the optimum-margin and the hard-margin have much better performance relative to the soft-margin SVM as they have larger area under the curve (AUC) which is obvious in the figure. To show that the optimum-margin has better performance compared with the hard-margin, the AUC values for all algorithms are calculated (see Table 5). The optimum-margin has the largest AUC value which demonstrates its optimal performance.

Algorithm→ Metric↓	Hard-Margin
Confusion Matrix	$\begin{array}{cc} 1559 & 31 \\ 61 & 1499 \end{array}$
Test accuracy	0.9707936507936508
Recall	0.9608974358974359
Precision	0.9797385620915032
F-1 score	0.9702265372168285

Table 4: Evaluating the performance metrics of the optimized SVM classifier.

Algorithm	AUC
Soft-margin	0.968452668924367
Hard-margin	0.9953825995807128
Optimum-margin	0.9956337687469763

Table 5: Area under the curve (AUC) for the soft-margin, hard-margin, and optimum-margin SVM. The optimum margin has the maximum AUC which demonstrates its optimal performance.

Question 5

Logistic regression is a predictive analysis for linear binary classification that attempts to find the probability that a document belongs to a given class. To do this it describes a "hyperplane" that lies between the two categories of data while decreasing the error of incorrect classification.

The function used to calculate the probability that the data point belongs to a given class are below:

$$\sigma(\phi) = \frac{1}{1+e^{-\phi}} \quad \text{(logistic function)} \tag{14}$$

where $\phi(x) = w^T x + c$.

Here, $\sigma(\phi)\mathrm{E}(0, 1)$ is the probability measure and w is the hyperplane dividing the two data classes.

The parameters w and c are found by maximizing the margin between classes (or the likelihood of categorizing a class properly). [**?**]

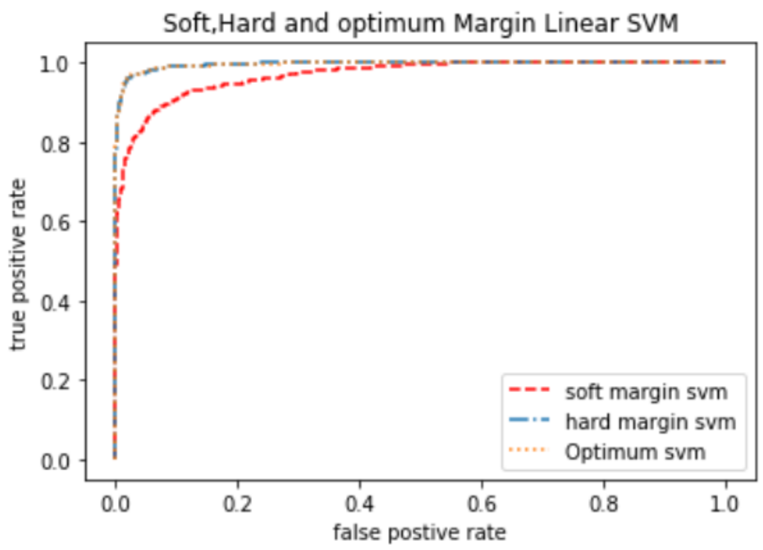


Figure 4: The ROC curves of the optimized SVM (orange) vs. hard-margin SVM (blue) and soft-margin SVM (red).

SVM Vs Logistic Regression

While SVM's goal is to maximize the "gap" between two classes of data while Logistic Regression focuses on minimizing the probability of classifying a document into the incorrect category. In addition, SVM relies strongly on the documents that lie along the boundary between the two classes and is not affected by outliers. [10]

Logistic Regression

Logistic Repression can improve the testing performance. Regularization can be thought of as the concept of constraining a model to prevent overfitting. What regularization effectively does is reduce the degrees of freedom when trying to optimize the fit. [1]

Intuitively, imagine regularization as a penalty against complex predictions. If your data is fit to a more complex model of the training data, it can be "overfit" and this can backfire when you apply it to testing data. "The key thing to note is the cost function penalizes confident and wrong predictions more than it rewards confident and right predictions!" [13]

L2 regression (also known as ridge regression)[11] optimizes the following cost function:

$$\min_{w,c} \lambda w^T w + C \sum_{i=1}^{n} \log(e^{-y_i(X_i^T w + c)} + 1) \tag{15}$$

where λ is the regularization strength.

As we can see, the difference between the two is in the penalty term. L1 regression adds the absolute value of magnitude of the coefficient w as a penalty term to the cost function. L2 regression adds a “squared magnitude” of the coefficient w as penalty term to the cost function.

The key difference between these techniques is that L1 removes some of the less important features and thus works well when there are a very large number of features. [11]

In the LogisticRegression scikit function, there are various “solvers” that can be used to obtain the fit to the training data. The solvers implemented in the class LogisticRegression are “liblinear”, “newton-cg”, “lbfgs”, “sag” and “saga”.

For this project we used the “lbfgs” solver for the unregularized LIBLINEAR solver to maintain consistency across L1 and L2 as it allowed us to use the same solver for both.

Results

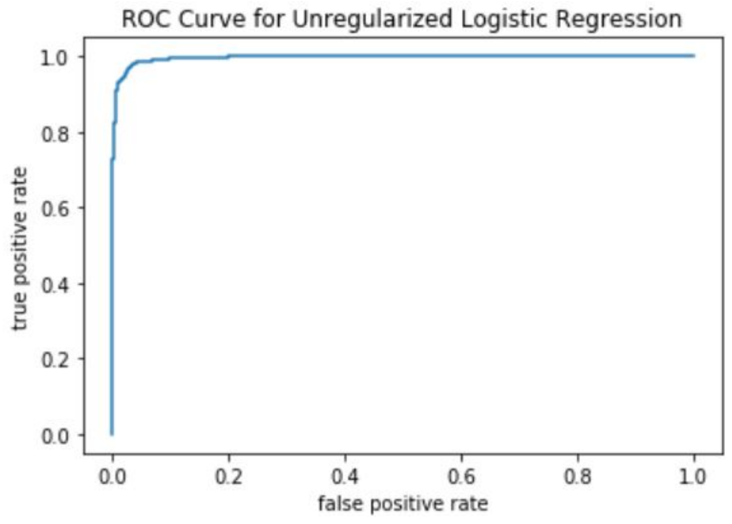
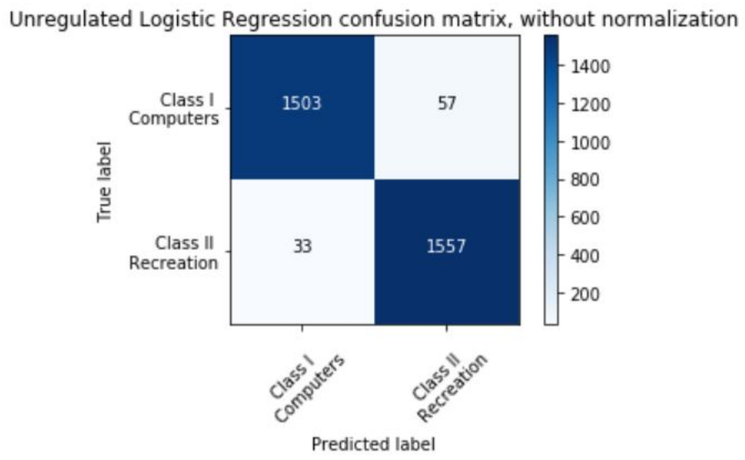


Figure 5: Area under unregularized logistic regression curve: 0.9958676019996774



For the L1 and L2 logistic regression penalties we were asked to find the best inverse of regularization strength:

$$C = \{10^k | -3 \leq k \leq 3, k \in \mathbb{Z}\} \tag{16}$$

	L1	L2
Optimal k-value	1.0	2.0
Optimal C-value	10.0	100.0
Optimal Regularization Strength (inverse of C=10^k)	0.1	0.01

Table 6: Metrics for Logistic Regression

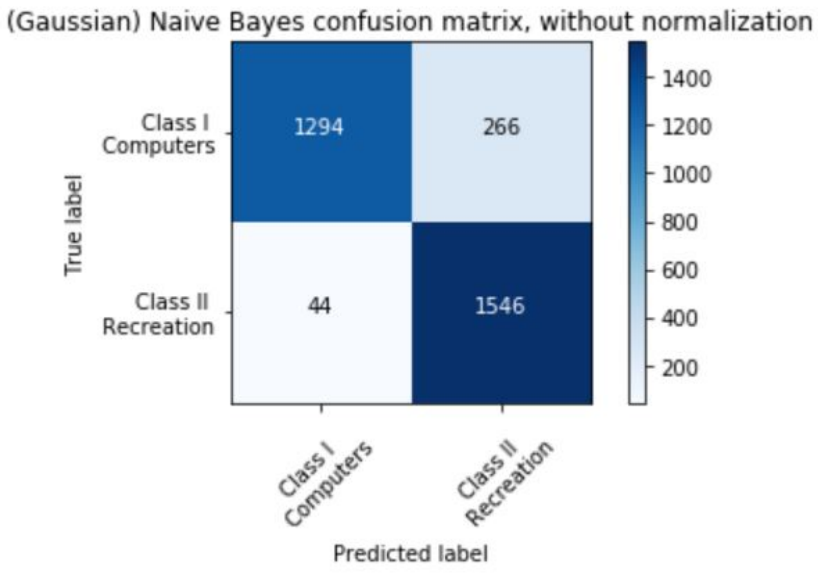
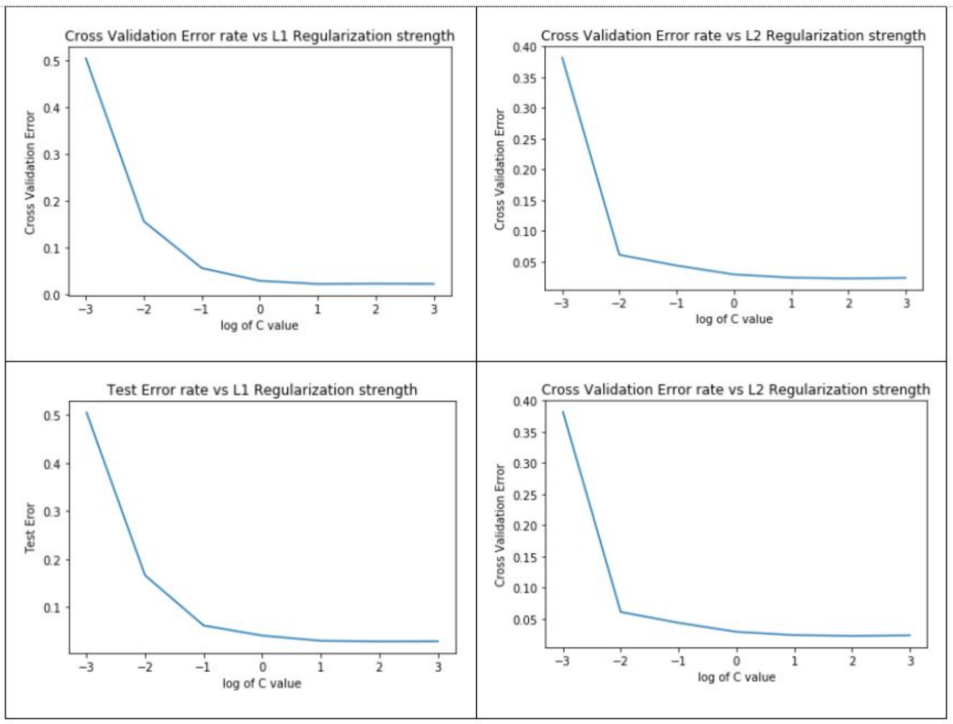


Figure 6: Confusion Matrix for Logistic Regression.



We see in the plots above that the tradeoff between test error decreased exponentially at first as k increased (as the regularization strength decreased). Then the error increased slightly; for L1 at k=2 and for L2 at k=3.

	Accuracy	Recall
Unregularized	0.9714285714285714	0.9792452830188679
L1	0.9704761904761905	0.9817610062893082
L2	0.9701587301587301	0.9817610062893082
Winner	Unregularized	Tie between L1 and L2
SVM (for comparison)	0.9707936507936508	0.980503144654088

Table 7: Metrics for Logistic Regression

As we can see, the unregularized logistic regression dominated most of the classification metrics on the test data. This indicates that there is no over-fitting in the model that we would want removed by regularization.

Our results for SVM and Logistic Regression were very similar. We believe this is because our model is very simple as we only used 50 terms to describe

	Precision	F1 Score
Unregularized	0.9646840148698885	0.9719101123595505
L1	0.9606153846153846	0.971073094867807
L2	0.9600246002460024	0.9707711442786069
Winner	Unregularized	Unregularized
SVM (for comparison)	0.9623456790123457	0.9713395638629283

Table 8: Metrics for Logistic Regression Continued

the data. If the model was more complex, we would have seen that the logistic regression models with regularization perform better.

Question 6

In Naive Bayes Classifier the Model assumption is, given the class label C, the feature vectors x_i are independent. In our example we have the class label C can take values 0 or 1 (Bernoulli random variable), So given C=k (k is 0 or 1) for a given training sample its feature vectors are independent that is we can model our data as if drawn from 50 independent univariate gaussians , which has only 50 mean and 50 variance parameters (total 100) to be learned instead of 50 dimensional gaussian distribution which has 50x1 mean vector and 50x50 full covariance matrix which has 1325 number of parameters. So Naive bayes assumption reduces the number of parameters required to be learned drastically, hence we can learn a good model without overfitting with less amount of training data.

$$\begin{aligned}
p(C_k)|x_1, \cdots, x_n) &\propto p(C_k, x_1, \cdots, x_n) \\
&= p(C_k)p(x_1|C_k)p(x_2|C_k)p(x_3|C_k)\cdots \\
&= p(C_k)\prod_{i=1}^{n} p(x_i|C_k)
\end{aligned} \tag{17}$$

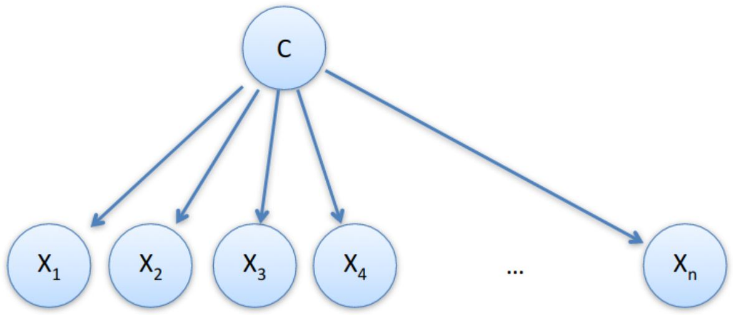


Figure 7: Naive bayes Classifier as a probabilistic graphical Model .

Results

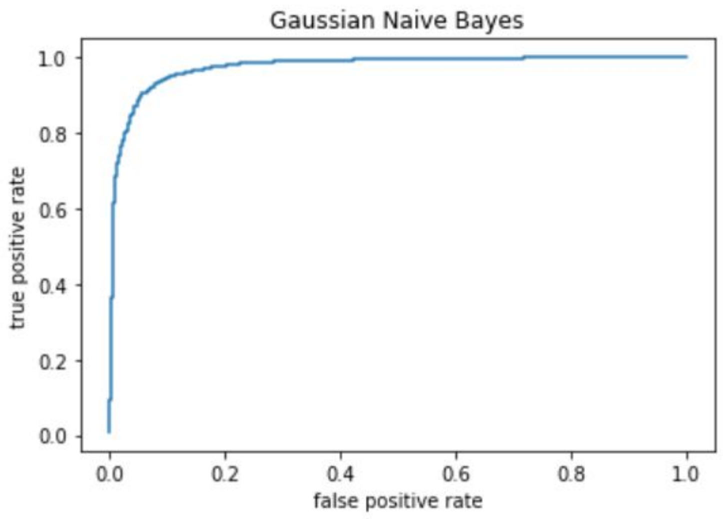


Figure 8: ROC for Gaussian Naive Bayes performance.

Metric	Score
Accuracy of (Gaussian) Naive Bayes:	0.9015873015873016
Recall of (Gaussian) Naive Bayes:	0.9723270440251572
Precision of (Gaussian) Naive Bayes:	0.8532008830022075
F1 score of (Gaussian) Naive Bayes:	0.9088771310993533

Table 9: Metrics for Gaussian Naive Bayes performance.

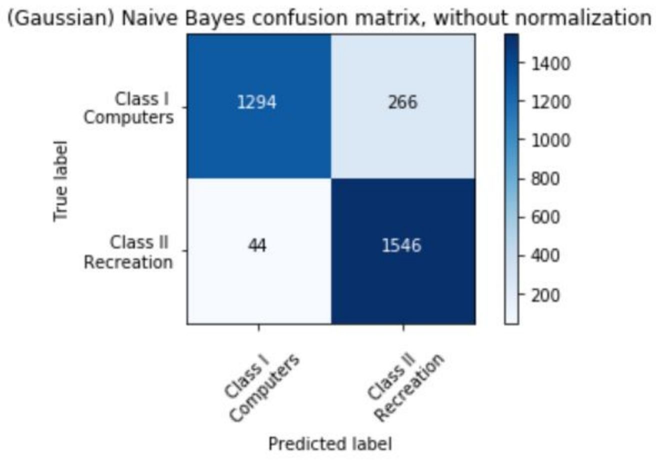


Figure 9: Confusion Matrix for Gaussian Naive Bayes performance.

Question 7

There are 3 basic steps in a classification problem: feature extraction, dimensionality reduction, and classification. In each step, we have multiple hyper parameters to tune, such as the minimum document frequency in feature extraction, or which technique to use for dimensionality reduction. Each of these decisions affect the performance of our final result. Scikit-Learn provides Pipeline and GridSearch functions to help us tune our hyperparameters in a structured and automated way.

Before discussing the Pipeline, let us note that Scikit-Learn has a library of transformers and classifiers that we have previously been using. Transformers are classes that have a “fit” and “transform” method. Each of the objects we used for feature extraction and dimensionality reduction were transformers. Classifiers are objects that have a “fit” and a ‘predict’ method, as we used for our classifiers. These transformer and classifier objects are both considered estimator classes in Scikit-learn.

The Pipeline class takes in a series of estimators. When the pipeline object’s “fit” method is called, each of the estimators “fit” objects is called, and the concatenated result is returned. The same thing occurs for its “transform” and “fit_transform” method. If the final estimator in the pipeline is a classifier, then we can call a “predict” and “fit_predict” method to execute the entire pipeline of estimators from beginning to end.

This Pipeline on its face makes our lives easier by reducing a few lines of code and making what we’re doing more human-readable. Its true power comes through when combined with the GridSearchCV object. The grid search object similarly can hijack another estimator’s “fit” and “predict” methods and act as an estimator object itself. However, the grid search object also receives a list of variables and their corresponding parameters. The grid search object will then iterate through each parameter and “fit” and “predict” for each possible combination of parameters. This automates testing the performance of different hyperparameters and choices for our classifier. When combined with the pipeline, we can swap out entire estimators, as well as their individual parameters, in the grid search. Each combination of estimators is subjected to a cross-validation, which means that a grid search with very few parameters will still take a very long time to run.

Results

We tested 32 combinations of parameters on our dataset. We did this twice — once including headers and footers in our dataset, once without.

Parameter	With Headers	Without Headers
Rank	1	1
Accuracy	97.506	97.3584
Classifier	Logistic Reg. with C=10 L1 Reg	Linear SVC with C=10
TF-IDF	LSA	LSA
$\min_d f$	5	3
Analyzer	Custom	Default

Table 10: The top performing combinations in the grid searches.

I discuss the specifics of "with headers" here, because this condition performed better overall.

We choose to score the grid search based on accuracy, and under this metric the top performers were the Linear SVC classifier with the LSI dimensionality reduction, min_df=3, and our custom analyzer, as well as the Logistic Regression classifier L1 using the LSI dimensionality reduction, min_df=3, and our custom analyzer.

The worst performer was the Gaussian Naive Bayes classifier with the LSI dimensionality reduction, min_df=5, and our custom analyzer. Because the second worst performer was this same set of parameters except for min_df=3, we can assume that min_df is not the culprit of this poor performance. Likewise, the next two worse performers still use Naive Bayes classification and LSI reduction, just with the default stop words list — thus we can conclude the count vectorizer is not the culprit. Finally we can note that the next two worst performers are the still Naive Bayes, but with the NMF dimensionality reduction. From this, we can conclude that the Gaussian Naive Bayes classifier is not the optimal choice for classifying our dataset.

In contrast, going through the top handful of performers, Linear SVC and both L1 and L2 logistic regression, when combined with LSI reduction, all perform very well. This tells us that dimensionality reduction is a major component in the accuracy of this data pipeline, but also that both Linear SVC and logistic regression are great classifiers to choose for this dataset.

Question 8

Until now we have solved binary classification problem (e.g. decide on whether it will rain or not rain). In this last part of the project we will tackle multi-class classification problem (e.g. Will tomorrow's weather be sunny, cloudy, rainy, foggy etc.)

Some of the Machine learning techniques like Naive Bayes Classifier, Logistic regression can be naturally be extended to multiclass classification. But there are a few approaches like SVM which can only perform Binary classification. To extend these kind of algorithms to multi class setting, two strategies are commonly used; one over rest (OVR) and one over one (OVO).

In the OVR approach, If we have N classes we need to train N classifiers, each classifier predicting whether the given point belongs to class-k or does not belong to class-k. The class-k which has the maximum confidence level is our prediction for a given test point. In this regime, each classifier is trained using the whole dataset. Some algorithms, such as support vector machines, scale poorly with the size of the training which represents a limitation for OVR strategy. However, it has the advantage of using small number of classifiers compared with OVR.

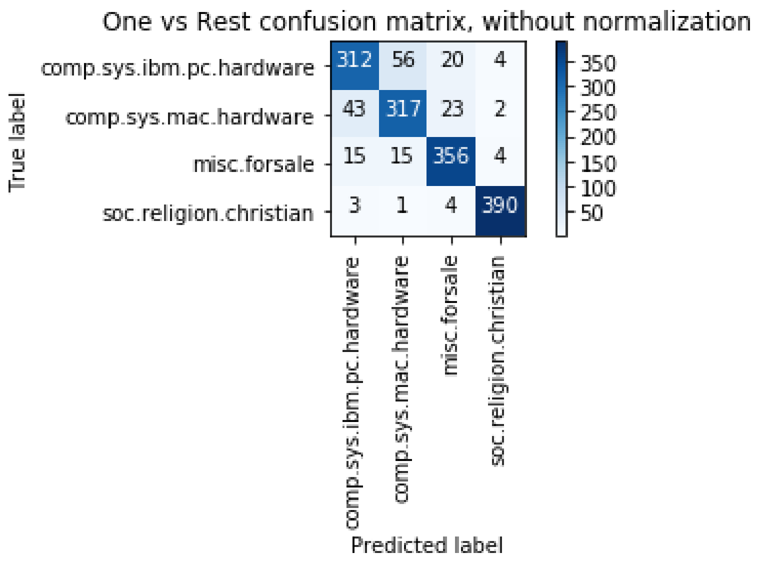
$$\hat{y} = \arg \max_{k \in 1 \cdots K} f_k(x) \tag{18}$$

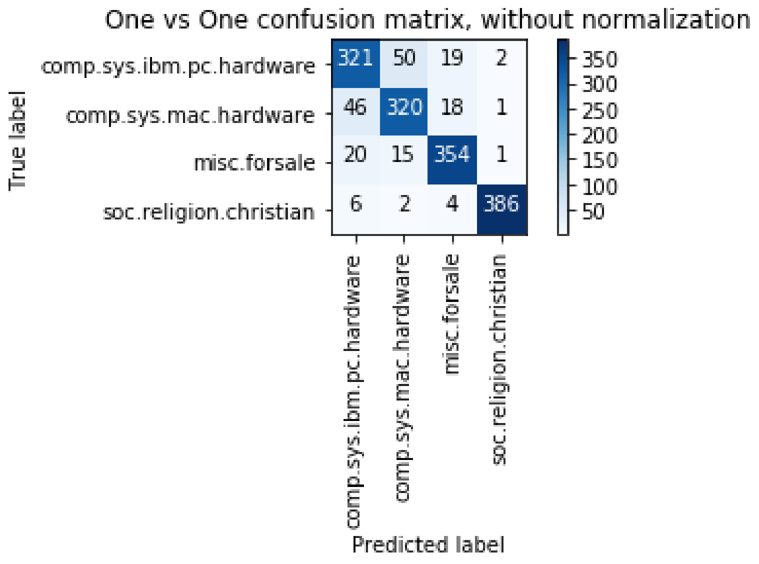
In the OVO approach, if there are N classes we train N*(N-1)/2 classifiers. Each classifier takes into account a possible pair of classes Ci and Ck. For a given test point we employ a voting based scheme to decide which class the given point belongs to. The advantage of this method is that each classifier uses only the the training data of only two classes which means that training set does not scale with the number of classes. The downside is its utilization to large number of classes. If we are training only 10 classes (e.g. MNIST dataset), this requires using 45 classifiers.

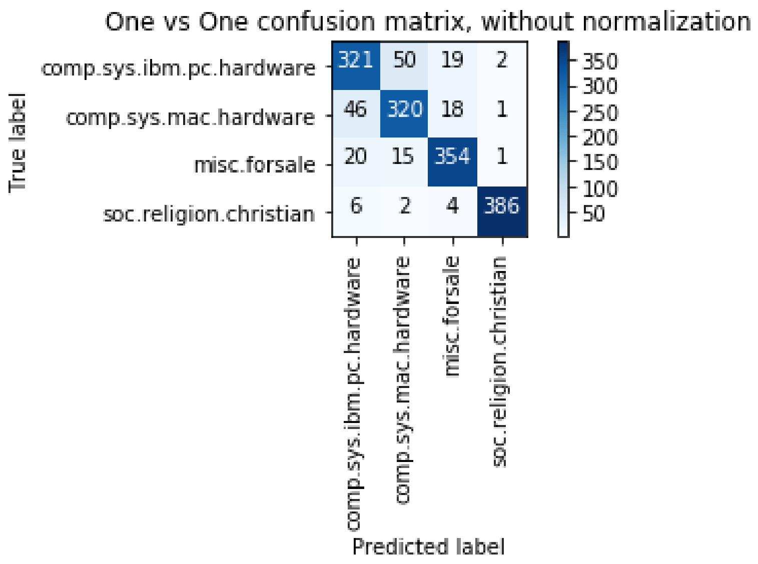
The performance parameters of OVO and OVR are summarized in talbe 11 and more figures are provided in the appended jupyter notebook. The performance of the OVO is better than the OVR in all performance parameters. This includes accuracy, recall and precision. The F1-score which summarizes the previous metrics is higher for the OVO which confirms the better overall performance of the the OVO classifier.

Algorithm→ Metric↓	OVO	OVR
Confusion Matrix	$\begin{array}{cccc} 322 & 49 & 17 & 3 \\ 50 & 313 & 21 & 1 \\ 19 & 15 & 355 & 1 \\ 7 & 2 & 4 & 385 \end{array}$	$\begin{array}{cccc} 309 & 59 & 17 & 7 \\ 47 & 313 & 23 & 2 \\ 19 & 14 & 355 & 2 \\ 5 & 2 & 3 & 388 \end{array}$
Test accuracy	0.8792332268370607	0.8722044728434505
Recall	0.8786399246243108	0.8715957753062922
Precision	0.8791918414026507	0.8710624505884719
F-1 score	0.878844231712086	0.871254094793351

Table 11: Evaluating the performance metrics of OVO and OVR classifiers.


One vs Rest confusion matrix, without normalization
True label
comp.sys.ibm.pc.hardware 312 56 20 4
comp.sys.mac.hardware 43 317 23 2
misc.forsale 15 15 356 4
soc.religion.christian 3 1 4 390
comp.sys.ibm.pc.hardware
comp.sys.mac.hardware
misc.forsale
soc.religion.christian
Predicted label
350
300
250
200
150
100
50


One vs One confusion matrix, without normalization
True label
comp.sys.ibm.pc.hardware 321 50 19 2
comp.sys.mac.hardware 46 320 18 1
misc.forsale 20 15 354 1
soc.religion.christian 6 2 4 386
comp.sys.ibm.pc.hardware
comp.sys.mac.hardware
misc.forsale
soc.religion.christian
Predicted label
350
300
250
200
150
100
50



Appendix

References

[1] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn & TensorFlow*. O'Reilley Media, Inc., 2017

@articlescikit-learn, title=Scikit-learn: Machine Learning in Python, author=Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E., journal=Journal of Machine Learning Research, volume=12, pages=2825–2830, year=2011

[2] Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E. "Multipath Propagation." *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research. Volume 12: 2825–2830, 2011.

[3] Matloff, Norman. "Unbalanced Data is a problem? No, balanced data is worse." *Mad (Data) Scientist.* `https://matloff.wordpress.com/2015/09/29/unbalanced-data-is-a-problem-no-balanced-data-is-worse/`

[4] Paarlberg, Simon. "Latent Semantic Analyses (LSA)." *Latent Semantic Analyses (LSA).* `https://simonpaarlberg.com/post/latent-semantic-analyses/`

[5] Deerwester, Scott, and Harshman, Richard and Dumais, Susan T., and Furnas, George W., and Landauer, Thomas K. *"Indexing by latent semantic analysis."* JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE, Columе 41:6, 391–407, 1990

[6] Dumais, Susan T. *Latent semantic analysis.* 10.1002/aris.1440380105

[7] Murphy, Kevin P. *Machine Learning: A Probablistic Perspective.* Massachusetts Institute of Technology, 2012.

[8] "The Why and How of Nonnegative Matrix Factorization". *Annual Review of Information Science and Technology.* Volume 38: 188-230, 2004. `https://onlinelibrary.wiley.com/doi/abs/10.1002/aris.1440380105`

[9] Gillis, Nicolas. *Department of Mathematics and Operational Research.* Faculte Polytechnique, Universite de Mons. March 2014.

[10] Drakos, Georgios and Drakos, Georgios. "Support Vector Machine vs Logistic Regression – Towards Data Science." *Towards Data Science.* Aug., 2018. `https://towardsdatascience.com/support-vector-machine-vs-logistic-re`

[11] Raschka. "Regularization in Logistic Regression: Better Fit and Better Generalization?" 2014. `https://www.kdnuggets.com/2016/06/regularization-logistic-regressio`

[12] assignment) Roychowdhury, Vwani. *Project 1: Classification Analysis on Textual Data* Assignment. `ECE 219 Winter 2019`.

[13] `https://people.eecs.berkeley.edu/ russell/classes/cs194/f11/lecture`

[14] Raschka, Sebastian. ”L1 and L2 Regularization Methods – Towards Data Science”. *Analytics Big Data Data Mining and Data Science.* Michigan State University. June 2014. https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25