EE219 Project 1
Classification Analysis on Textual Data
Winter 2018

Group Members: Asavari Limaye; UID: 605224431
Pooja Janagal Nagaraja; UID: 405222664
Rupa Mahadevan; UID: 005225216
Vaishnavi Ravindran; UID: 805227216

1. INTRODUCTION:

In this project we deal with Classification on textual data. In machine learning and statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. In this case of our project, we apply this concept of classification to textual data that is first preprocessed by means of an efficient representation (tf-idf), reduced in dimensions (NMF and LSI). We then apply various classification algorithms, analyse and compare their performances. We ultimately build the complete pipeline for the classification task on text data.

2. DATASET:

For our dataset, we use scikit-learn's "20 Newsgroups" dataset. This is a collection of over 20K documents that are partitioned across these 20 different topics/categories. The topics in this dataset are closely related to one another and hence can be grouped into parent or super classes. For the majority part of our project we consider the following two super categories - "Computer Technology" and "Recreational Activity". Each of these two super-classes consists of 4 different topics from the "20 Newsgroups" as shown below:

Table 1: Two class distribution between "Computer Technology" and "Recreational Activity"

Super-class	Class1	Class 2	Class 3	Class 4
Computer Technology	comp.graphics	comp.os.ms-windows.misc	comp.sys.ibm.pc.hardware	comp.sys.mac.hardware
Recreational Activity	rec.autos	rec.motorcycles	rec.sport.baseball	rec.sport.hockey

QUESTION 1: ***Plot a histogram of the number of training documents for each of the 20 categories to check if they are evenly distributed.***

In any classification problem, we need to be cautious of the problem of "imbalanced data" which refers to the scenario when one class has significantly more data than the other class. In the case that is imbalanced, there three ways to deal with it:

1. Oversampling: resampling of data from the minority class.
2. Under-sampling: randomly eliminate tuples from the majority/dominating class
3. Synthesizing new data points for the minority class

However, for our project the data is already evenly distributed across the 20 categories as shown in the histogram below.

Result:

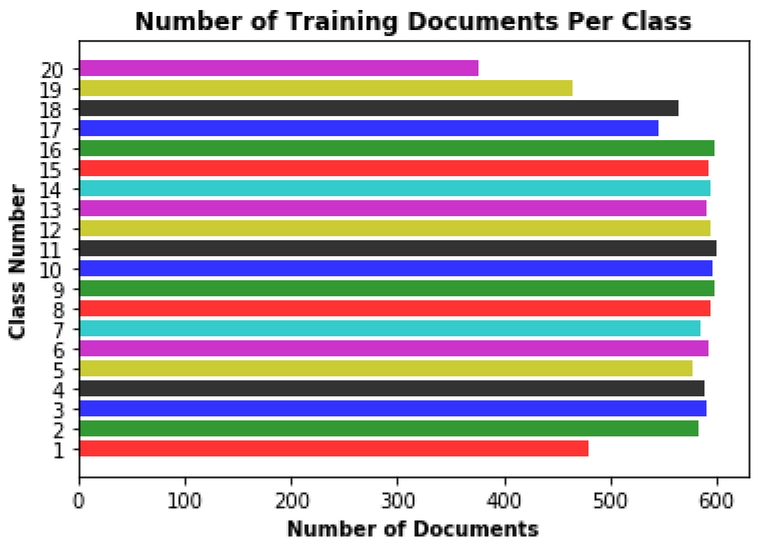


Fig. 1 Number of training documents per class

Observation:

The above histogram plot of the number of training documents present in each of the 20 documents shows us that the distribution of documents is close to being evenly distributed in each category.
Class 20 and class 1 however have fewer documents than the remaining categories.

3. FEATURE EXTRACTION

We use bag-of-words representation in our project for the document representation. This type of representation both avoids computational intractability and over-fitting. The entire corpus is thus represented as a document-term matrix, where each entry (i,j) is the count of the term j in the document i. We also apply the following specs to extract features from the textual data:

• Use the "Term Frequency-Inverse Document Frequency (TF-IDF)" metric to normalize the count of the vocabulary words in each document.
• Use the "english" stopwords of the CountVectorizer
• Exclude terms that are numbers (e.g. "123", "-45", "6.7" etc.)
• Perform lemmatization with nltk.wordnet.WordNetLemmatizer and pos_tag
• Use min_df=3. Here min_dif is the threshold such that when building the vocabulary we ignore terms that have a document frequency strictly lower than the given threshold.

***QUESTION 2:* Report the shape of the TF-IDF matrices of the train and test subsets respectively.**

Results:

Table 2: Shape of X_train_tfidf and X_test_tfidf for different combinations of min_dif and lemmatization=true/false

Operations performed	Train_dataset	Test_Dataset
Tfdif transformation after feature extraction using CountVectorizer with **min_dif = 3** , excluding numbers, performing lemmatization ***with*** nltk.wordnet.WordNetLemmatizer and pos_tag	X_train_tfidf = (4732, 16292)	X_test_tfidf = (3150, 16292)
Tfdif transformation after feature extraction using CountVectorizer with **min_dif = 4** , excluding numbers, performing lemmatization ***with*** nltk.wordnet.WordNetLemmatizer and pos_tag	X_train_tfidf =(4732, 12640)	X_test_tfidf = (3150, 12640)

Tfdif transformation after feature extraction using CountVectorizer with min_dif = 3 , excluding numbers, performing lemmatization ***without*** nltk.wordnet.WordNetLemmatizer and pos_tag	X_train_tfidf =(4732, 18768)	X_test_tfidf = (3150, 18768)

Observation and Inference: The shapes of TF-IDF matrices of the train and test subsets have been reported in table 2 with different combinations for min_dif and lemmatization configs.

a. The shape of X_train_tfidf and X_test_tfidf increases in feature length when min_dif value decreases. That is, X_train_tfidf = (4732, 16292) for min_dif = 3 and X_train_tfidf =(4732, 12640) for min_dif = 4. This makes sense since min_dif is the threshold below which we ignore terms that have a document frequency strictly lower than the given threshold.
b. The shape of X_train_tfidf and X_test_tfidf increases in feature length when we do not perform lemmatization with nltk.wordnet.WordNetLemmatizer and pos_tag.. That is, X_train_tfidf = (4732, 16292) for lemmatization = True and X_train_tfidf =(4732, 18768) for lemmatization = False. This makes sense since **lemmatization** is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form. As a result with lemmatization, we will be reducing the number of documents in the vocabulary which reflects in the decrease in feature size of document-term matrix.

4. DIMENSIONALITY REDUCTION

In order to overcome the “Curse of Dimensionality”, we use two dimensionality reduction methods: Latent Semantic Indexing (LSI) and Non-negative Matrix Factorization (NMF). These techniques are individually applied to the TF-IDF matrix, whose rows correspond to TF-IDF representation of the documents.

QUESTION 3: Dimensionality Reduction using NMF and LSI.

a) First LSI is applied to the TF-IDF matrix corresponding to the 8 categories with k = 50; hence mapping each document to a 50-dimensional vector.

For LSI, we perform SVD on the data matrix X as shown below:

$$X_k = U_k \Sigma_k V_k^T$$ ----eqn(1)

b) Second, NMF is applied to the TF-IDF matrix also corresponding to the categories with k = 50; hence mapping each document to a 50-dimensional vector.

For NMF, we try to approximate the data matrix by describing the documents in the TF-DIF matrix by a linear combination of the topics present as shown below:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\mathsf{T} \\ \vdots \\ \mathbf{x}_n^\mathsf{T} \end{bmatrix} \approx \mathbf{WH} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1r} \\ \vdots & \vdots & \vdots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{mr} \end{bmatrix} \begin{bmatrix} \mathbf{h}_1^\mathsf{T} \\ \vdots \\ \mathbf{h}_r^\mathsf{T} \end{bmatrix}$$

----eqn(2)

Results:

We calculate the square of the frobenius norm of the reconstruction error for both LSI and NMF and report them.

The value of $\|\mathbf{X} - \mathbf{U}_k\Sigma_k\mathbf{V}_k^\mathsf{T}\|_F^2$ in LSI is 4108.291966953609

The value of $\|\mathbf{X} - \mathbf{WH}\|_F^2$ in NMF is 4146.201714043186

Observation and Inference: The value of $\|\mathbf{X} - \mathbf{WH}\|_F^2$ is larger than $\|\mathbf{X} - \mathbf{U}_k\Sigma_k\mathbf{V}_k^\mathsf{T}\|_F^2$, that is, LSI performs better than NMF for our dataset. Some of the possible reasons are:

1. NMF converges to a local minima, in fact, different initializations can lead to different local optimas. Where LSI is more robust as it has a unique factorization without letting the initialization affect its results.
2. LSI is more deterministic compared to that of NMF. That is, LSI computes the eigenvectors which are deterministic for a given matrix. As a result of this the results of LSI doesn't change much based on which package we use or what the initial conditions are. On the other hand, in case of NMF, it tries to get best fit decomposed matrices,

which are trained on some training data and then done an evaluation on test data. Based on the technique used to get these smaller matrices we may end up with different results.

3. Another point is that we should also consider regularization in NMF while we don't need to worry about it in SVD.
4. SVD is a more 'insightful' factorization technique. NMF gives us only U and V matrices, where as SVD gives us a Sigma matrix along with these two. The Sigma matrix gives us insights into the amount of information that each of the eigen vectors hold. This type of information is not available in NMF.

4. LEARNING MODELS:

In this section, the models that have been used for classifying the news groups has been discussed. The dataset is divided into two groups: "Computer Technology" and "Recreational Activities". The dataset has target values ranging from 0 to 7. The values from 0-3 correspond to "Computer Technology" and 4-7 correspond to "Recreational Activities". Several binary classification models have been implemented in the following sections. They have been discussed in detail below

4. 1. SUPPORT VECTOR MACHINES:

Linear Support Vector Classifiers are used to classify data using a linear hyperplane.

QUESTION 4: Hard margin and soft margin linear SVMs

In this part, two linear svms were trained and compared. One is a hard margin svm (γ = 1000) and the other is soft margin SVM (γ = 0:0001)

Result:

Hard margin SVM:

ROC curve for hard margin SVM (γ = 1000)

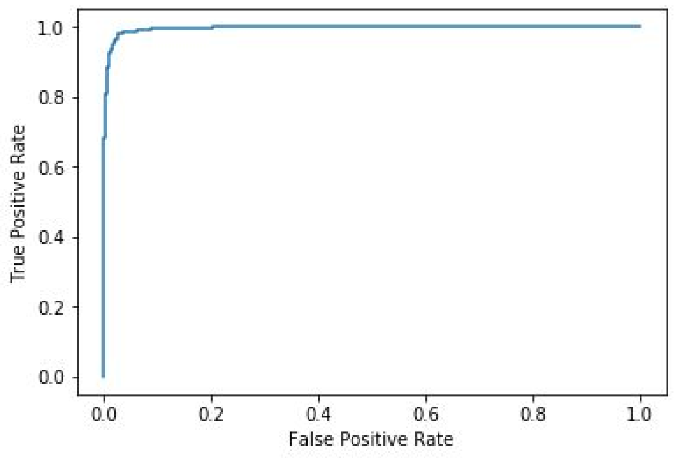


Table 3: Confusion Matrix for hard margin SVM (γ = 1000)

	Predicted = NO	Predicted =Yes
Actual = NO	TN = 1514	FP= 46
Actual = Yes	FN = 30	TP = 1560

Table 4: Metric values for Hard Margin SVM

Metric	Value
Accuracy	0.9758730158730159
Precision	0.9811320754716981
Recall	0.9713574097135741
F1-score	0.9762202753441802

Soft margin SVM:

ROC curve for soft margin SVM (γ = 0.0001)

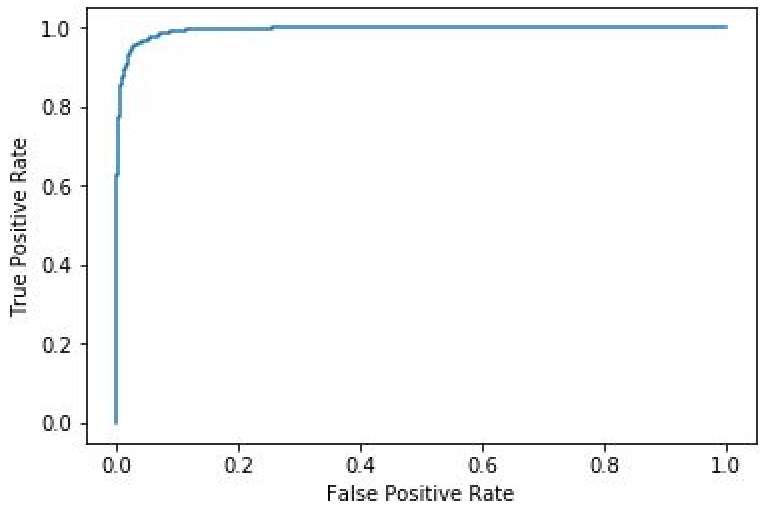


Table 5: Confusion Matrix for soft margin SVM (γ = 0.0001)

	Predicted = NO	Predicted =Yes
Actual = NO	TN = 74	FP= 1486
Actual = Yes	FN = 0	TP = 1590

Table 6: Metric values for Soft Margin SVM

Metric	Value
Accuracy	0.5282539682539683
Precision	0.5169050715214565
Recall	1.0
F1-score	0.6815259322760395

Observations and Inference:

1. Which SVM performs better? Hard vs Soft Margin SVM:

   - From Tables 4 and 6 it is clear that the Hard Margin SVM is doing much better than the Soft margin SVM. This is because the Soft Margin SVM has a very high false positive rate, i.e., it classifies many data point as belonging to class Recreational Activity when the data point is actually of the class Computer Technology
   - Though the soft margin svm has a high recall of 1, all the other metrics such as accuracy, precision, f1-score are worse than the hard margin svm. The high value for recall can be attributed to the fact that the data samples that do belong to the class Recreational Activity have been correctly classified into that class and since recall is the fraction of "positive classified" samples that are in fact "positive" (see eqn 3). we get a value of 1 for recall.

- Instead of recall, for our particular assessment, other metrics like accuracy, precision, and F1-score are better to evaluate the soft margin svm. Using these metrics, it is clear that the hard margin svm performs better.

2. What happens for the soft margin SVM? Why is the case? Does the ROC curve of the soft margin SVM look good? Does this conflict with other metrics?

- An ROC curve is a plot of true positive rate vs false positive rate, where true positive rate and false positive rate are defined as below:

$$\textbf{TPR /Recall / Sensitivity} = \frac{\textbf{TP}}{\textbf{TP + FN}}$$

----eqn(3)

$$\textbf{FPR = 1 - Specificity}$$
$$= \frac{\textbf{FP}}{\textbf{TN + FP}}$$

----eqn(4)

- The ROC curve is a plot of the True Positive Rate to the False Positive Rate. The formulae for the two are as stated above. The recall for the soft margin classifier is 1. And the false positive rate at different thresholds is also high. Hence, the ROC curve is similar to that of Hard Margin classifier. It is seen that the soft margin SVC predicts most of the data points as positive class. Therefore, the true positives and false positives is high and a good ROC curve is achieved.
- However the other metrics like accuracy and precision are low since these values also consider the number of points being classified in the negative class. Hence, ROC is not a suitable metric for soft margin classifier.

3. Use cross-validation to choose (use average validation accuracy to compare): Using a 5-fold cross-validation, find the best value of the parameter in the range. Again, plot the ROC curve and report the confusion matrix and calculate the accuracy, recall precision and F-1 score of this best SVM.

Results:

Table 7: Value of γ vs Avg. validation accuracy

Value of γ	**Average validation accuracy**
10 ^ -3	0.6371830985915492
10 ^ -2	0.6373239436619718
10 ^ -1	0.9722535211267607
10 ^ 0	0.9746478873239438
10 ^ 1	0.9757746478873239
10 ^ 2	0.9750704225352113
10 ^ 3	0.9723943661971832

From table 7, we can see that $\boldsymbol{\gamma} = \mathbf{10}$ gives us the best accuracy.

We now report the ROC curve, the confusion matrix, the accuracy, recall, precision and F-1 score of this best SVM with $\boldsymbol{\gamma} = \mathbf{10}$

ROC curve for best SVC (γ = 10)

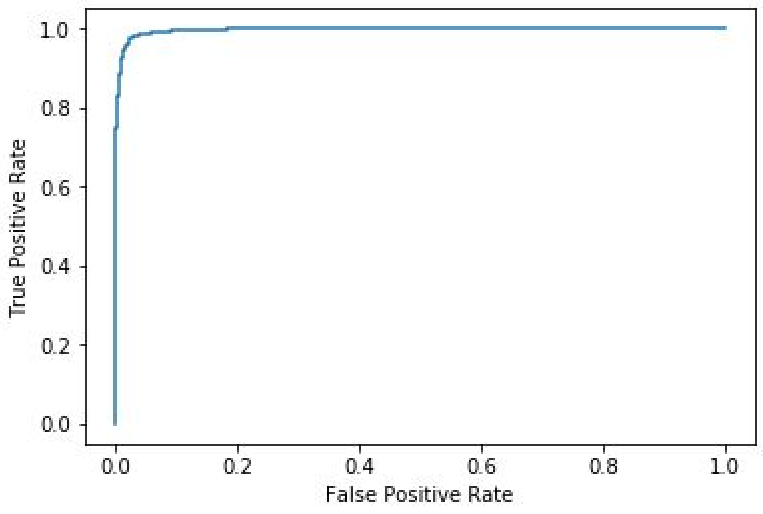


Table 8: Confusion Matrix for the best SVC ($\gamma = 10$)

	Predicted = NO	Predicted =Yes
Actual = NO	TN = 1508	FP= 52
Actual = Yes	FN = 31	TP = 1559

Table 9: Metric values for best SVC ($\gamma = 10$)

Metric	Value
Accuracy	0.9736507936507937
Precision	0.9677219118559901
Recall	0.980503144654088
F1-score	0.9740706029365823

Observation:

The above results show us that we need to tune the value of γ to improve the performance of the SVM.

4. 2. LOGISTIC REGRESSION:

Logistic Regression is a linear model that learns the best fit or relationship between the data points and their labels.

QUESTION 5:

1. Train a logistic classifier without regularization; plot the ROC curve and report the confusion matrix and calculate the accuracy, recall precision and F-1 score of this classifier.

In order to train a logistic classifier without regularization, we use sklearn.linear_model.LogisticRegression. To implement without regularization, we set the value of the parameter C as high as possible. (Since, C is the inverse of λ in the equation 5 below)

$$\widehat{\boldsymbol{\beta}} = \min_{\beta} \; -LL(\boldsymbol{\beta}; \boldsymbol{y}, \boldsymbol{X}) + \lambda\, R(\boldsymbol{\beta}).$$

----eqn(5)

where LL stands for the logarithm of the likelihood function, β for the coefficients, for the dependent variable and X for the independent variables.

The *l1* norm is defined as:

$$R(\boldsymbol{\beta}) = \| \boldsymbol{\beta} \|_1 = \sum_{i=0}^{n} |\beta_i|,$$

----eqn(6)

The regularization term for the L2 regularization is defined as:

$$R(\boldsymbol{\beta}) = \frac{1}{2} \| \boldsymbol{\beta} \|_2^2 = \frac{1}{2} \sum_{i=0}^{n} \beta_i^2,$$

----eqn(7)

Eqn. 5 gives us how to to calculate the regression coefficients of a logistic regression: using the negative of the Log Likelihood function, also called the objective function, is minimized.
Eqn. 6 and Eqn.7 are L1 and L2 regularization respectively.
By setting lambda to a value as small as possible, we can achieve Logistic Regression without regularization.

Results:

ROC curve for Logistic Regression without regularization (λ = 0)

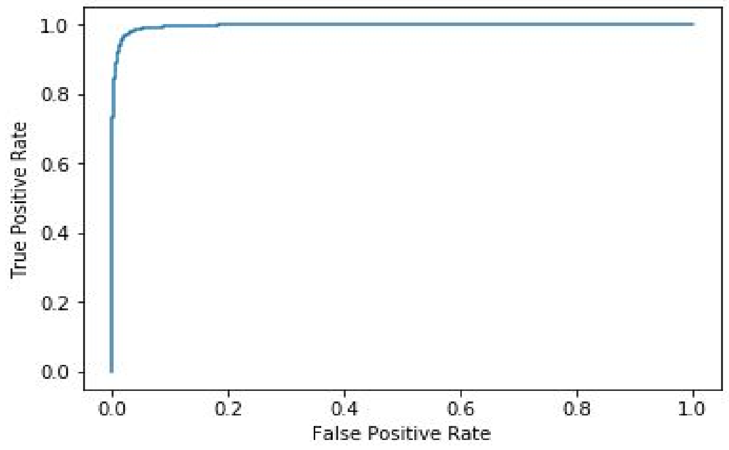


Table 10: Confusion Matrix for Logistic Regression without regularization ($\lambda = 0$)

	Predicted = NO	Predicted =Yes
Actual = NO	TN = 1508	FP= 52
Actual = Yes	FN = 31	TP = 1559

Table 11: Metric values for Logistic Regression without regularization ($\lambda = 0$)

Metric	Value
Accuracy	0.9736507936507937
Precision	0.9677219118559901
Recall	0.980503144654088
F1-score	0.9740706029365823

2. Using 5-fold cross-validation on the dimension-reduced-by-svd training data, find the best regularization strength in the range {10 ^k | -3 <= k <= 3, k ∈ Z} for logistic regression with L1 regularization and logistic regression L2 regularization, respectively.

Results:

Logistic Regression models for different regularization strengths have been computed **for L1 norm. The best performance was obtained for λ = 10.** The ROC curve, confusion matrix and performance metrics for the best model has been reported below:

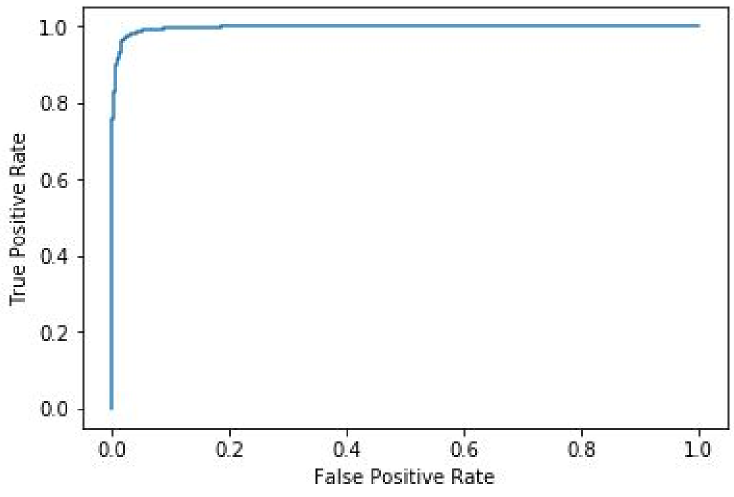


Table 12: Confusion Matrix for Logistic Regression with L1 regularization ($\lambda = 10$)

	Predicted = NO	Predicted =Yes
Actual = NO	TN = 1504	FP= 56
Actual = Yes	FN = 31	TP = 1559

Table 13: Metric values for Logistic Regression with L1 regularization ($\lambda = 10$)

Metric	Value
Accuracy	0.9723809523809523
Precision	0.9653250773993808
Recall	0.980503144654088
F1-score	0.9728549141965678

Logistic Regression models for different regularization strengths have been computed **for L2 norm. The best performance was obtained for λ = 100.** The ROC curve, confusion matrix and performance metrics for the best model has been reported below:

ROC Curve for Logistic Regression with L2 regularization

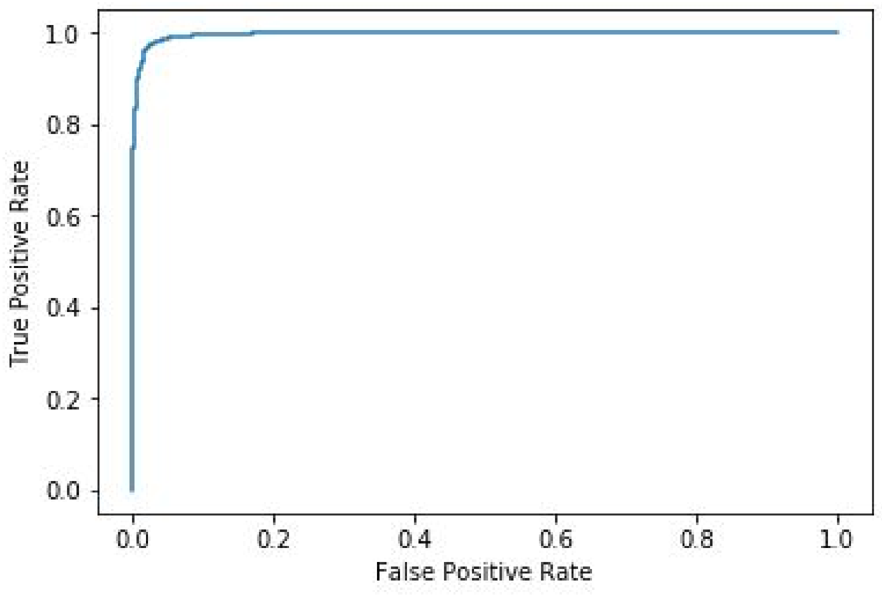


Table 14: Confusion Matrix for Logistic Regression with L2 regularization (λ = 100)

	Predicted = NO	Predicted =Yes
Actual = NO	TN = 1508	FP= 52
Actual = Yes	FN = 29	TP = 1561

Table 15: Metric values for Logistic Regression with L2 regularization (λ = 100)

Metric	Value
Accuracy	0.9742857142857143
Precision	0.967761934283943
Recall	0.9817610062893082
F1-score	0.9747112082422729

Observation:
From the tables 13 and 15, it can be seen that L2 regularization performs slightly better than L1 regularization. The accuracy, precision, recall and F1-score are better in L2 regularization.

3. Compare the performance (accuracy, precision, recall and F-1 score) of 3 logistic classifiers: w/o regularization, w/ L1 regularization and w/ L2 regularization (with the best parameters you found from the part above), using test data.

The role of regularization is that it tries to reduce overfitting on unseen data. Intuitively, it is clear that the bigger the values of the coefficients, the lesser will be the training error which also means that error on new unseen data can be high (this is because the value of the coefficients represents how much the model "fits" to the training data). We will now compare the performance of our logistic regression model under each metric value for the 3 classifiers using test data and the best parameters found:

- Accuracy: We can see from tables 11, 13, and 15 that the accuracy of the classifier with L2(97.43%) > without regularization(97.36)> L1(97.24%) .
- Precision :
  L2(96.78%) > without regularization(96.77)> L1(96.53%):
- Recall:

L2(98.18%) > without regularization(`98.0503144654088%`)=L1(`98.0503144654088%`):

- f1-score:
  L2(97.47%) > without regularization(`97.40706029365823%`)>L1(`97.28549141965678%`):

Reasoning: Though usually the model should have a better performance for l1 and l2 compared to without regularization, this can also depend on the dataset at hand. Since regularization ties to prevent the coefficients from fitting the "noise" in the problem, that is, overfitting , it is possible that there is not that much noise in our training data. As a result, without regularization> L1. However, L2> L1(97.24%) and this is as expected since L2 is always known to perform better than L1. Typically ridge or L2 penalties are better for minimizing prediction error rather than *L1* penalties. This is because that when two predictors are highly correlated, *L1* regularizer will simply pick one of the two predictors. Where as, the *ℓ2* regularizer will keep both of them and jointly shrink the corresponding coefficients a little bit. Thus, while the ***ℓ1*** penalty can certainly reduce overfitting, it can lead to a loss in predictive power.

4. How does the regularization parameter affect the test error? How are the learnt coefficients affected? Why might one be interested in each type of regularization?

Results:

Table 16: Testing error for Logistic regression models with and without regularization

	Without Regularization	With L1 Regularization	With L2 Regularization
Testing Error	2.64	2.76%	2.57%

Table 17: Weights for different features with and without regularization

Feature Number	Without Regularization	With L1 Regularization	With L2 Regularization
1	-6.91304298	-2.49874735	-2.38422442
2	124.84857102	106.45320738	84.38635704
3	-26.23988705	-19.74255818	-15.42250465
4	90.54208461	74.17114882	59.06476417
5	15.04675017	10.78259556	8.75271808
6	-17.78867016	-10.86096733	-7.57085773
7	-6.92527905	-4.1550245	-3.9011007
8	-1.03777747	0.	-1.0747365
9	19.75220887	16.35728411	14.07437215
10	14.92965149	12.80917863	11.60645078

Observation:

Regularization helps in improving generalization or preventing overfitting by penalizing the model for increasing model complexity. That is, it tries to reduce the value of coefficients which might have fitted to well to the "noise" in the training data.

From Table 16 , we can see the effect of regularization on the testing error. Error of L2<Error without regularization< error of L1. As explained earlier, the testing error of L2 is the least.

From the above sets of results it can be seen that the second coefficient is quite high in general. The value is 124.8 for a model without regularization but this coefficient is lower in L1 and L2 regularization- 106.45 and 84.38 respectively.

Why might one be interested in each type of regularization?

- L1 regularization is used to "built in feature selection". This is because L1 can produce sparse coefficients, that is, many of the coefficients end up to be zero. This provides a means of selecting the features that actually add value. However, L1-norm does not have an analytical solution and as a result can have multiple solutions.

- L2 regularization on the other hand does not produce sparse coefficients . However, it has only one solution. It also almost always performs better than L1.

5. Both logistic regression and linear SVM are trying to classify data points using a linear decision boundary, then what's the difference between their ways to find this boundary? Why their performance differ?

Linear SVM tries to classify the data points by finding the hyperplane with the maximum margin. This implies that the classifier not only tries to classify the points correctly but also tries to find the orientation of the hyperplane such that the data points are as far away as possible from the hyperplane.
Logistic regression finds the best fit for a dataset and reduces the output to a value in the range of [0, 1]. A threshold is chosen which classifies these outputs to class 0 and 1. So logistic regression maximizes the chance of correct classification based on this threshold.
Since SVMs try to find not just the best classifier but also the maximum margin, SVM performs better than logistic regression. This can also be seen in the results above.

QUESTION 6: Gaussian Naive Bayes Classifier

Naive Bayes Classifier is supervised learning algorithm based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Bayes theorem is given by the equation below:

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots x_n \mid y)}{P(x_1, \dots, x_n)}$$

For this question, a Gaussian Naive Bayes was trained with default parameters. 'GaussianNB' from Scikit-learn was used.

Results:

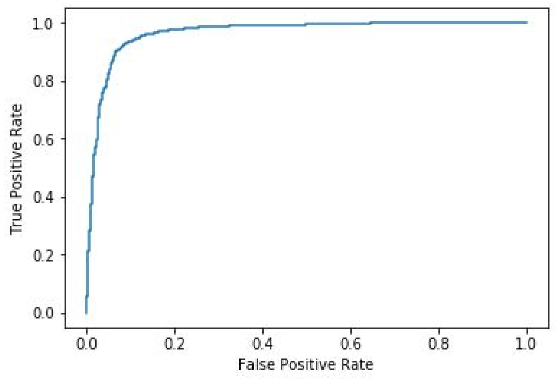


Table 18: Confusion Matrix for Gaussian Naive Bayes Classifier

	Predicted = NO	Predicted =Yes
Actual = NO	TN = 912	FP= 263
Actual = Yes	FN = 209	TP = 1766

Table 19: Metric values for Gaussian Naive Bayes Classifier

Metric	Value
Accuracy	0.8501587301587301
Precision	0.8703794972893051
Recall	0.8941772151898734
F1-score	0.882117882117882

QUESTION 7: Grid Search of parameters

Throughout the process of training and testing there are a lot of parameters that can be tuned. These include value for 'min_df', using lemmatization, LSI or NMF for dimensionality reduction and so on. Along with this, we can also run different models: SVC, Logistic Regression and Naive Bayes. Within each of these models, there are parameters that can be tuned.
We try out different such combinations to understand which combination works the best for us. To search over these combinations , we use scikit-learn's **GridSearchCV** to perform a grid search over the range of parameters for a pipeline. The **pipeline** involves the processes responsible for count vectorization, TF-IDF transformation, dimensionality reduction and classification model.
Parameters considered for tuning:

1. Loading data - We perform the task by first using training data that has headers and footers. Then we perform the same task for the training data from which headers and footers have been eliminated at the time of loading.
2. Feature extraction - When count vectorization builds the vocabulary, it ignores terms that have a document frequency strictly lower than the value of 'min_df'. So, we perform the task for min_df = 3 and min_df = 5.

We also consider using lemmatization and not using lemmatization.

3. Dimensionality Reduction - We also try different dimensionality reduction techniques. We use LSI and NMF both with number of components to be reduced to as 50.
4. Classifier-We use Linear SVC with best C previously found which is 10 and loss as hinge loss.

   We use Logistic Regression with best C previously found. We try Logistic Regression with L1 Regularization and C as 10. We use Logistic Regression with L2 Regularization and C as 100.

   We use Gaussian Naive Bayes with default parameters.

Results for different combinations:
First we run grid search for training set with headers and footers.

Table 20: Observations for training set with headers and footers

Combination Number	**min_df**	**Lemmatization**	**Dimensionality reduction**	**Classifier**	**Mean test accuracy**
0	3	Yes	LSI	LinearSVC	0.977599324
1	5	Yes	LSI	LinearSVC	0.9727388
2	3	No	LSI	LinearSVC	0.975063398
3	5	No	LSI	LinearSVC	0.975063398
4	3	Yes	NMF	LinearSVC	0.960693153
5	5	Yes	NMF	LinearSVC	0.960059172
6	3	No	NMF	LinearSVC	0.956043956
7	5	No	NMF	LinearSVC	0.954353339
8	3	Yes	LSI	LogisticRegression - L1	0.976754015
9	5	Yes	LSI	LogisticRegression - L1	0.975486052

10	3	No	LSI	LogisticRegression - L1	0.974852071
11	5	No	LSI	LogisticRegression - L1	0.974852071
12	3	Yes	NMF	LogisticRegression - L1	0.970414201
13	5	Yes	NMF	LogisticRegression - L1	0.969568893
14	3	No	NMF	LogisticRegression - L1	0.967666948
15	5	No	NMF	LogisticRegression - L1	0.96682164
16	3	Yes	LSI	LogisticRegression - L2	0.976754015
17	5	Yes	LSI	LogisticRegression - L2	0.974006762
18	3	No	LSI	LogisticRegression - L2	0.976754015
19	5	No	LSI	LogisticRegression - L2	0.975274725
20	3	Yes	NMF	LogisticRegression - L2	0.966610313
21	5	Yes	NMF	LogisticRegression - L2	0.965765004
22	3	No	NMF	LogisticRegression - L2	0.962172443
23	5	No	NMF	LogisticRegression - L2	0.963440406

24	3	Yes	LSI	GaussianNB	0.911665258
25	5	Yes	LSI	GaussianNB	0.906170752
26	3	No	LSI	GaussianNB	0.894125106
27	5	No	LSI	GaussianNB	0.90046492
28	3	Yes	NMF	GaussianNB	0.946322908
29	5	Yes	NMF	GaussianNB	0.946956889
30	3	No	NMF	GaussianNB	0.941251057
31	5	No	NMF	GaussianNB	0.94103973

Next, we run grid search for training set without headers and footers.

Table 21: Observations for training set without headers and footers

Combination Number	**min_df**	**Lemmatization**	**Dimensionality Reduction**	**Classifier**	**Mean Test Accuracy**
0	3	Yes	LSI	LinearSVC	0.96555368
1	5	Yes	LSI	LinearSVC	0.96639899
2	3	No	LSI	LinearSVC	0.96703297
3	5	No	LSI	LinearSVC	0.96893491
4	3	Yes	NMF	LinearSVC	0.94885883
5	5	Yes	NMF	LinearSVC	0.95245139
6	3	No	NMF	LinearSVC	0.94759087
7	5	No	NMF	LinearSVC	0.95033812

8	3	Yes	LSI	LogisticRegression - L1	0.9680896
9	5	Yes	LSI	LogisticRegression - L1	0.96914624
10	3	No	LSI	LogisticRegression - L1	0.97189349
11	5	No	LSI	LogisticRegression - L1	0.97252747
12	3	Yes	NMF	LogisticRegression - L1	0.965765
13	5	Yes	NMF	LogisticRegression - L1	0.96407439
14	3	No	NMF	LogisticRegression - L1	0.96449704
15	5	No	NMF	LogisticRegression - L1	0.96428571
16	3	Yes	LSI	LogisticRegression - L2	0.9680896
17	5	Yes	LSI	LogisticRegression - L2	0.96851226
18	3	No	LSI	LogisticRegression - L2	0.97083686
19	5	No	LSI	LogisticRegression - L2	0.97231615
20	3	Yes	NMF	LogisticRegression - L2	0.95773457
21	5	Yes	NMF	LogisticRegression - L2	0.95900254

22	3	No	NMF	LogisticRegression - L2	0.95710059
23	5	No	NMF	LogisticRegression - L2	0.95815723
24	3	Yes	LSI	GaussianNB	0.84298394
25	5	Yes	LSI	GaussianNB	0.84530854
26	3	No	LSI	GaussianNB	0.85608622
27	5	No	LSI	GaussianNB	0.85418428
28	3	Yes	NMF	GaussianNB	0.94357566
29	5	Yes	NMF	GaussianNB	0.94294167
30	3	No	NMF	GaussianNB	0.94146238
31	5	No	NMF	GaussianNB	0.94737954

The best test accuracy is **0.977599**
The best combination is for training set where headers and footers were not removed.
Other Parameters of the best combination are:
min_df = 3
Lemmatization = Yes
Dimensionality Reduction = LSI with n_components as 50
Classifier = LinearSVC with C as 10

QUESTION 8: Multiclass Classification using Naive Bayesian Classification, One-vs-One and One-vs-Rest SVM.

The following 4 categories of data were fetched from newsgroups:

a. ‘comp.sys.ibm.pc.hardware’
b. 'comp.sys.mac.hardware'
c. 'misc.forsale'
d. 'soc.religion.christian'

The ‘english’ stop words from sklearn and nltk were combined with punctuation symbols from Python, to form a combined set of stopwords, as given in the discussion session.

Each word was lemmatized using the nltk WordNetLemmatizer and pos_tags.
The data was then transformed to the tf-idf form, and then reduced to 50 features per document using SVD transformation.

Following this, the naive bayesian, One-vs-One and One-vs-Rest were implemented using scikit learn.

A. The Naive Bayesian Classifier used was the GaussianNB classifier from scikit learn. The classifier was trained on the train data set. The hyperparameter which was tuned for GaussianNB was the 'var_smoothing' which is a measure of the portion of variance of the all the features which is added. The F1 Score of the classification on the train data set was chosen as the metric to compare the choices of the hyperparameter. F1 score was chosen over precision, accuracy and recall because in the problem of categorizing test data, reducing false positives and false negatives are given equal importance.

   The F1 Score on classifying the train dataset of the Naive Bayes Classifier using different values of var_smoothing are given:

Table 22: F1-scores of Naive Bayes Classifier

Var Smoothing	**F1 Score on Train**
1e-09	0.6717528520739757
1e-06	0.6717528520739757
1e-03	0.6722411710178989
1e-01	0.6376358288175102

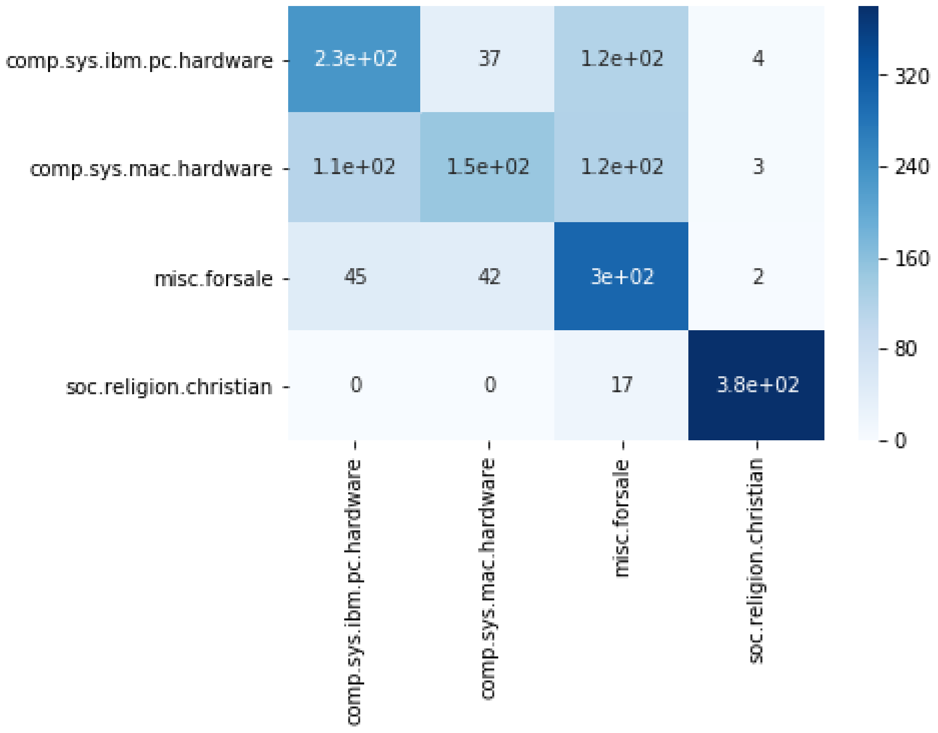
The default value of var_smoothing is 1e-09 but the value 1e-03 gives a larger F1 Score on the train data and is chosen. The results of classification using a Gaussian naive Bayes Classifier with var_smoothing = 1 e-03 are given below. The measured values are based on predictions on the test data.

Table 23: Performance Metrics for Gaussian Naive Bayes Classifier

Accuracy	67.8594249201278 %
Precision	69.06627031198585 %
Recall	67.63334057662844 %

F1 Score	66.97373643301569 %

The following is the confusion matrix of the classification on the test data.



B. For One-Vs-One SVM Classifier, the LinearSVC() Classifier from scikit learn was used with the OnevsOneClassifier, also from scikit learn. The hyperparameter C for LinearSVC was tuned by varying it and using the value which gave the best F1 Score on the train data. The F1 Score was used to give equal importance to the recall and precision. The hyperparameter C controls the amount of misclassification which is tolerated. A higher value of C will lead to smaller margins and fewer misclassifications.

The results of hyperparameter tuning are given below:

Table 24: F1 score for different C values

Values of C	**F1 Score on Train**
1e-05	0.3289583593377138
1e-02	0.8503158795030694
1	0.8848021467127868
100	0.8856870146466452

10000	0.7943184416737072

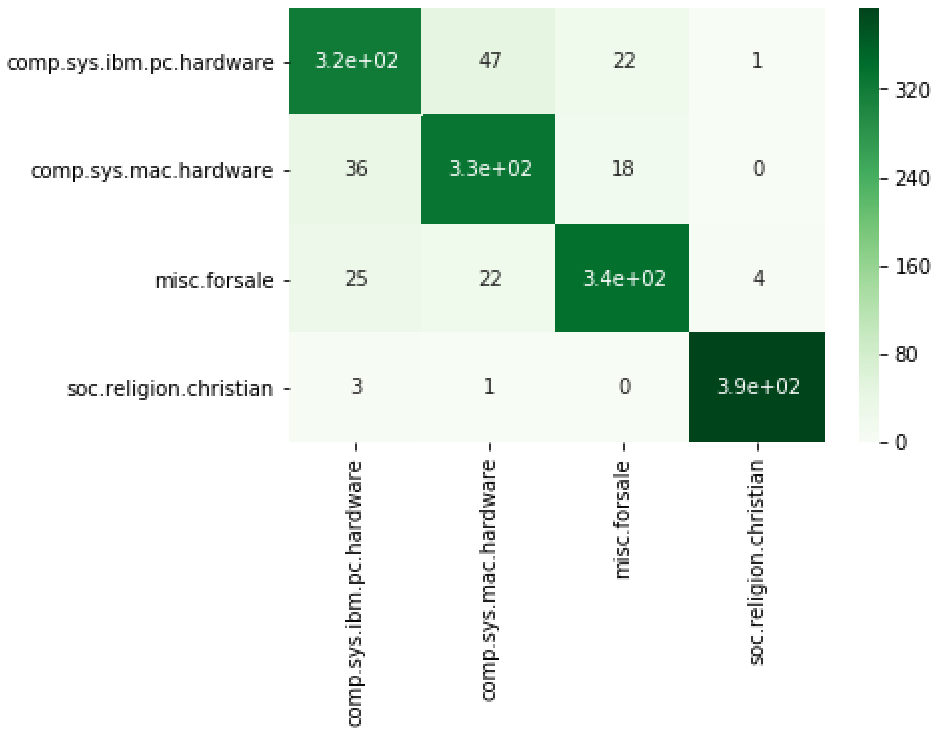
The value of hyperparameter C = 100 gave the best F1 Score on train data, and was chosen. The recall, precision, accuracy and F1 Score using a One-vs-One Classifier for LinearSVC with random state=42, C=100 are reported below:

Table 25: Performance metrics for LinearSVC with C = 100

Accuracy	88.56230031948881 %
Precision	88.60381982516016 %
Recall	88.56230031948881 %
F1	88.56870146466453 %

Confusion Matrix for One-vs-One Classifier:

```
[[322  47  22   1]
 [ 36 331  18   0]
 [ 25  22 339   4]
 [  3   1   0 394]]
```



C. For One-Vs-Rest SVM Classifier, the LinearSVC() Classifier from scikit learn was used with the OnevsRestClassifier from scikit learn. The hyperparameter C for LinearSVC was tuned by varying it and using the value which gave the best F1 Score on the train data. The F1 Score was used because equal importance is to be given to the recall and precision.

The results of hyperparameter tuning are given below:

Table 26: F1 score for different C values

C	**F1 Score on Train**
1e-05	0.33239065515123123
1e-02	0.8530828041050972
1	0.8767975550963494
100	0.8790877146635673
10000	0.8733028897021407

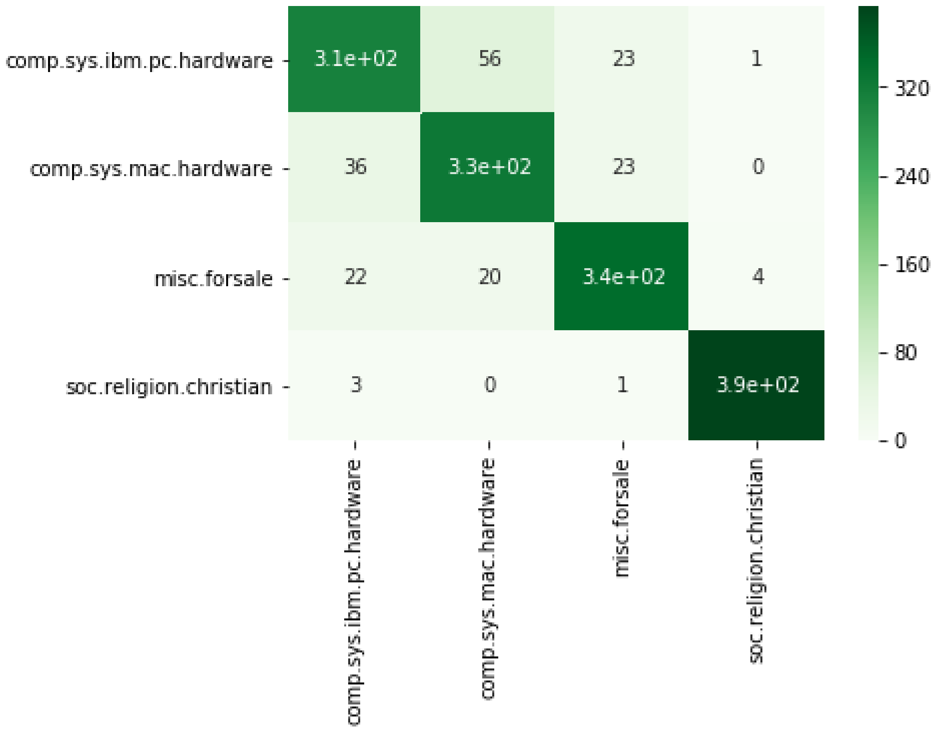
The value of hyperparameter C = 100 gave the best F1 Score on train data, and was chosen. The recall, precision, accuracy and F1 Score using a One-vs-Rest Classifier for LinearSVC with random state=42, C=100 are reported below:

Table 27:Performance metrics for LinearSVC with C = 100

Accuracy	87.92332268370608 %
Precision	87.93862157663015 %
Recall	87.92332268370608 %
F1	87.90877146635673 %

Confusion Matrix for One-vs-Rest Classifier:

```
[[312  56  23   1]
 [ 36 326  23   0]
 [ 22  20 344   4]
 [  3   0   1 394]]
```


comp.sys.ibm.pc.hardware
comp.sys.mac.hardware
misc.forsale
soc.religion.christian
3.1e+02
3.3e+02
3.4e+02
3.9e+02