EE219 Project 1

Classification Analysis on Textual Data

Winter 2020

01/20/2020

Chenyu Wang	UID 805436446
Gaofang Sun	UID 104853165
Gongjie Qi	UID 805429380
Hao Xu	UID 305429561

Introduction

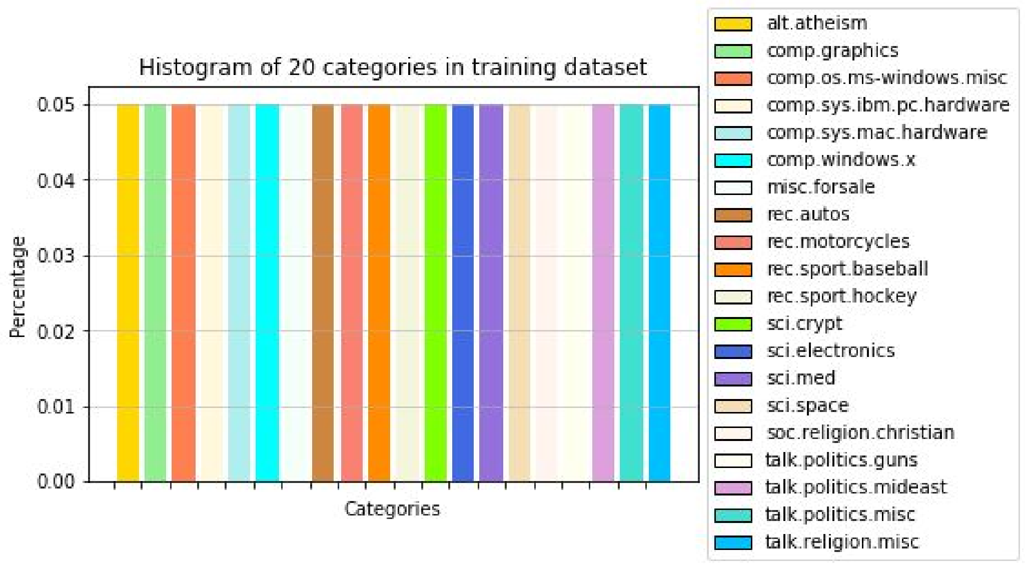
We live in a society where textual data are generated daily in an unprecedentedly high rate and it is on high demand that data engineers needs to make sense of the textual data to the extent like how we analyse the numerical data. In this project, we explore different models to classify textual data to achieve statistical classification. Firstly, we convert the textual dataset into numerical matrix. We extract feature by lemmatization, counter-vectorization and use TF-IDF to weigh them. Then we compared the performances of two dimension reduction methods LSI and NMF. Secondly, we apply binary classification model to solve the problem, including Hard/Soft Margin SVM, Multinomial Naïve Bayes and Logistic Regression, and then extend to multiclass classification with SVM and Naïve Bayes algorithms. The model performances were evaluated with confusion matrix, accuracy, precision, recall and ROC curve. Furthermore, we applied Grid search methods to estimate the best combination of different methods through the data processing pipeline. In the end we also experimented on the multiclass classification and compared the performances of multiclass classification on different classifier models.

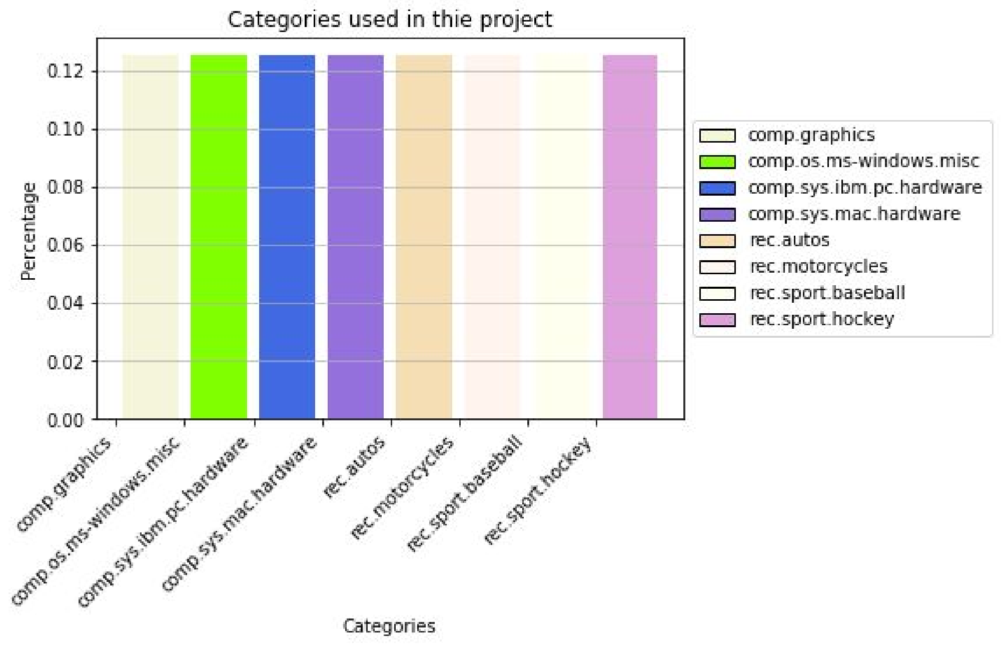
I. Data Exploration

In this project, we work with the 20 newsgroups dataset. It comprises around 20,000 newsgroups posts on 20 topics. Size balance in each category of dataset is of great importance to an effective learning algorithms. In case of imbalance in the relative sizes of the datasets with different category, we would need to modify the penalty function or down-sample the majority classes. To check if they are balanced is very important.

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

The histogram of 20 categories shows that all the categories have balanced number of training dataset. Among all the categories, there are two major classes "Computer technology" and "Recreation activity". "Computer technology" includes 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware' 4 subclasses. "Recreation activity" includes 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey' 4 subclasses. In this project we will focus on the data in the eight categories from these 2 major classes.





The second figure (above) is the histogram of documents over these concerned topics. It shows that the data is indeed well balanced as stated in the description.

II. Feature Extraction

In this section, we explored a common representation of documents, "Bag of Words", which summarize a corpus of text into a term-document matrix whose entries are some statistics of the terms. Then to avoid unnecessarily large feature vectors (vocabulary size), we use the stop words and lemmatizer to drop certain less meaningful words or terms. Then we extract the feature using TF-IDF, which captures the importance of a term to a document.

QUESTION 2: Extraction from textual data

In order to extract features from all the documents in training and testing set, we apply Part-of-Speech tagging (POS tag) function on the tokenized texts, and lemmatize the textual data based on the POS information on each of the tokens. Digits, special symbols and stop words are further removed. CountVectorizer is then used to convert the lemmatized the text to matrix of token counts. We chose Term Frequency–Inverse Document Frequency (TF-IDF) as our numerical statistic as it works well at capturing the importance of words in a document in the corpus. The result of TF-IDF shows the shape of training set and test set as following:

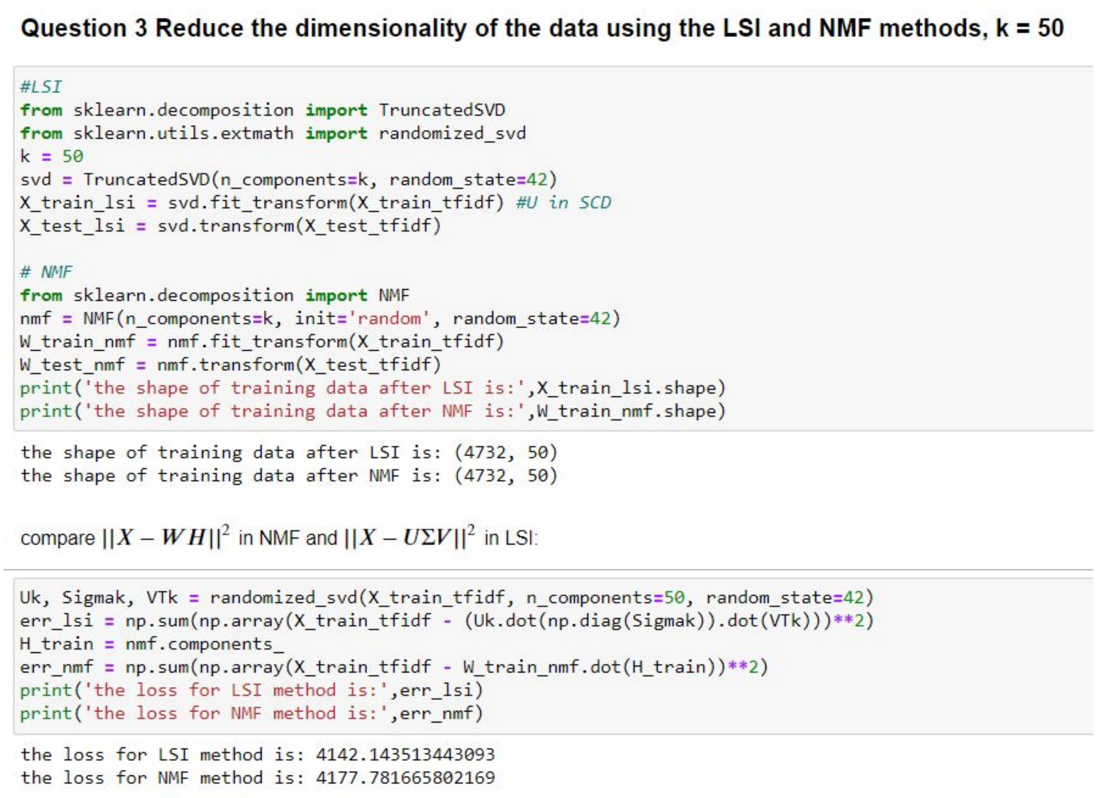
```
shape of train subset:(4732, 16741)
shape of test subset: (3150, 16741)
```

III. Dimension Reduction

In this section, to avoid poor performance in high-dimensional data, we experiment with two dimensionality reduction methods: Latent Semantic Indexing (LSI) and Non-negative Matrix Factorization (NMF), both of which minimize mean squared residual between the original data and a reconstruction from its low-dimensional approximation.

QUESTION 3: Reduce dimension of the textual data

In this problem, we applied two dimensionality reduction transform methods. LSI performs operation on SVD of TF-IDF matrix to get a lower dimensional space. NMF factorizes the matrix into two positive matrix to reduce dimension. We reduced the dimension to 50. The codes are shown below:



Question 3 Reduce the dimensionality of the data using the LSI and NMF methods, k = 50

```
#LSI
from sklearn.decomposition import TruncatedSVD
from sklearn.utils.extmath import randomized_svd
k = 50
svd = TruncatedSVD(n_components=k, random_state=42)
X_train_lsi = svd.fit_transform(X_train_tfidf) #U in SCD
X_test_lsi = svd.transform(X_test_tfidf)

# NMF
from sklearn.decomposition import NMF
nmf = NMF(n_components=k, init='random', random_state=42)
W_train_nmf = nmf.fit_transform(X_train_tfidf)
W_test_nmf = nmf.transform(X_test_tfidf)
print('the shape of training data after LSI is:',X_train_lsi.shape)
print('the shape of training data after NMF is:',W_train_nmf.shape)
```

```
the shape of training data after LSI is: (4732, 50)
the shape of training data after NMF is: (4732, 50)
```

compare $||X - WH||^2$ in NMF and $||X - U\Sigma V||^2$ in LSI:

```
Uk, Sigmak, VTk = randomized_svd(X_train_tfidf, n_components=50, random_state=42)
err_lsi = np.sum(np.array(X_train_tfidf - (Uk.dot(np.diag(Sigmak)).dot(VTk)))**2)
H_train = nmf.components_
err_nmf = np.sum(np.array(X_train_tfidf - W_train_nmf.dot(H_train))**2)
print('the loss for LSI method is:',err_lsi)
print('the loss for NMF method is:',err_nmf)
```

```
the loss for LSI method is: 4142.143513443093
the loss for NMF method is: 4177.781665802169
```

Shape of feature vector for LSI and NMF are both (4732, 50)

$\|X - WH\|^2$ in NMF : 4142.14

$\|X - U\,\Sigma\,V^T\|$ in LSI : 4177.78

From the above calculation, NMF methods is shown to have a larger loss. The formulas above can be understood as the loss of information after dimension reduction of the two methods. NMF aims to minimize the objective loss function. NMF gives non-unique and positive matrix. Different from NMF, LSI gives the singular vectors based on large singular values. This results in a generally larger or equal information loss in NMF than that in LSI.

IV. Classification Algorithms

In this section, we use the dimension-reduced training data from LSI to train classifiers, and evaluate the trained classifiers with test data, more specifically, to classify the documents into two classes “Computer Technology” vs “Recreational Activity”. Since it is a binary classification, we label the ‘Computer Technology’ and ‘Recreational Activity’ as value 0 and 1 respectively in both training and testing datasets. After training for each classification model, we generate an evaluation report including confusion matrix, accuracy, recall, precision and f1_score, and a ROC curve to exhibit the relationship between true positive rate (TPR) and false positive rate (FPR).

QUESTION 4: Hard and soft margin SVMs

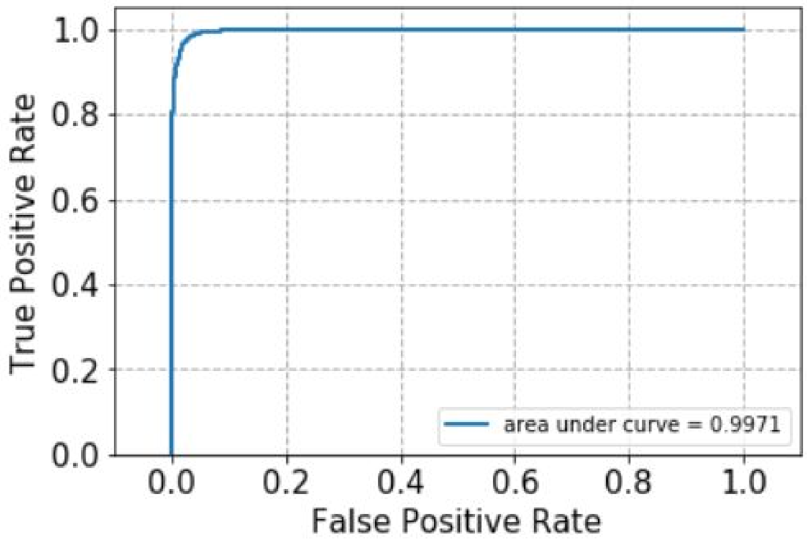
- Train two linear SVMs and compare:

To decide if a document should be classified as Computer Technology or Recreational Activity, in this problem, we train two linear SVMs: one with C = 1000 (hard margin), another with C = 0.0001 (soft margin). Then we evaluate trade-off relationship between the TPR and FPR by ROC curve. We then use cross-validation to choose C and plotted the ROC curve, reported the confusion matrix and calculate the accuracy, recall precision and F-1 score of this best SVM.

C as the cost of misclassification, trades off correct classification. For instance, when misclassification of individual points is highly penalized, the SVM model is called "Hard Margin SVM". In contrast, a "Soft Margin SVM", is very lenient towards misclassification of a few individual points as long as most data points are well separated.

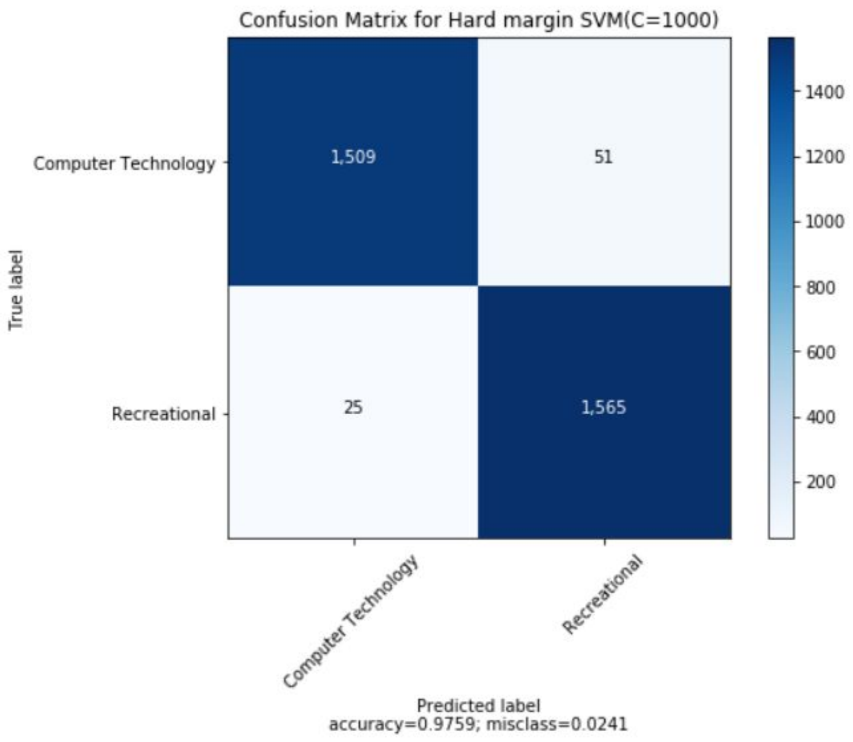
Hard margin linear SVM (C = 1000):

ROC curve:

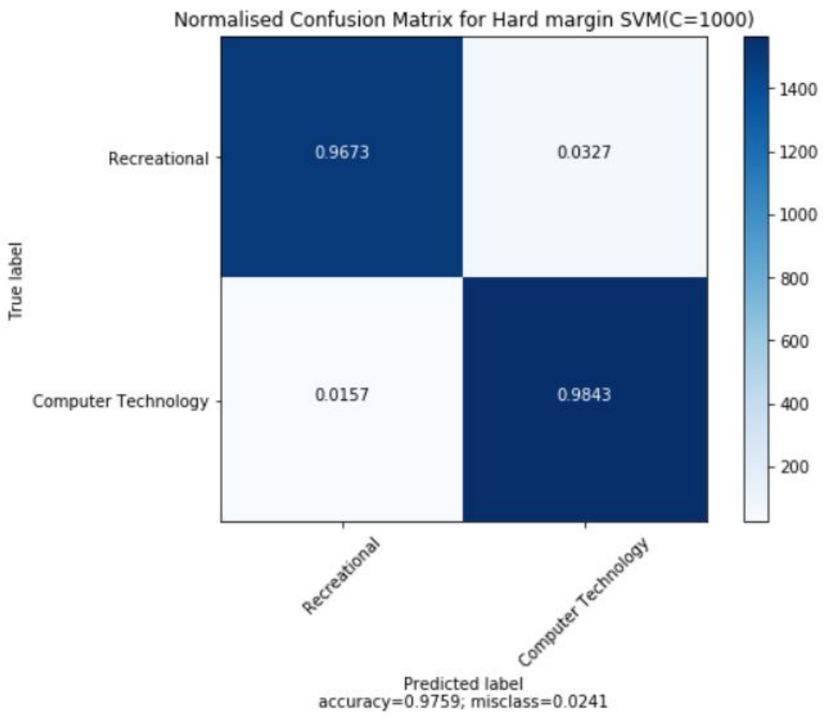


Accuracy, recall, precision, f1-score:
Accuracy = 0.9759
Recall = 0.9843
Precision = 0.9684
F1_score = 0.9763

Confusion Matrix:
Before normalization:

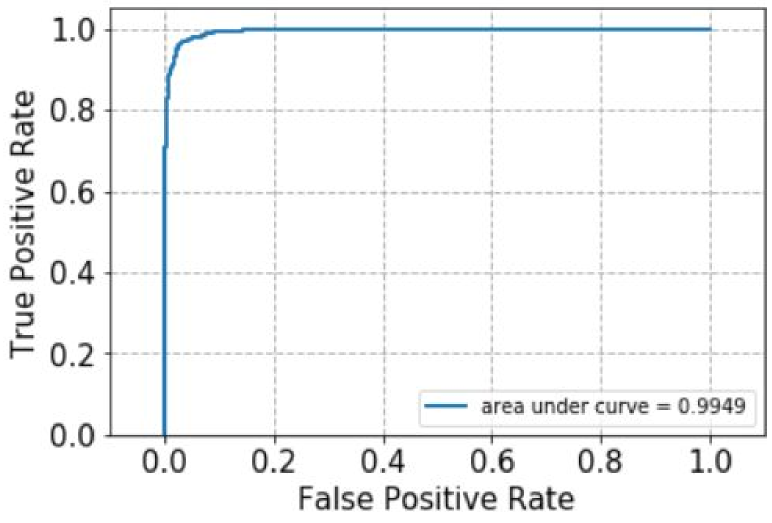


Normalized confusion matrix:



Soft margin linear SVM (C = 0.0001):

ROC curve:



Accuracy, recall, precision, f1-score:

Accuracy = 0.5048

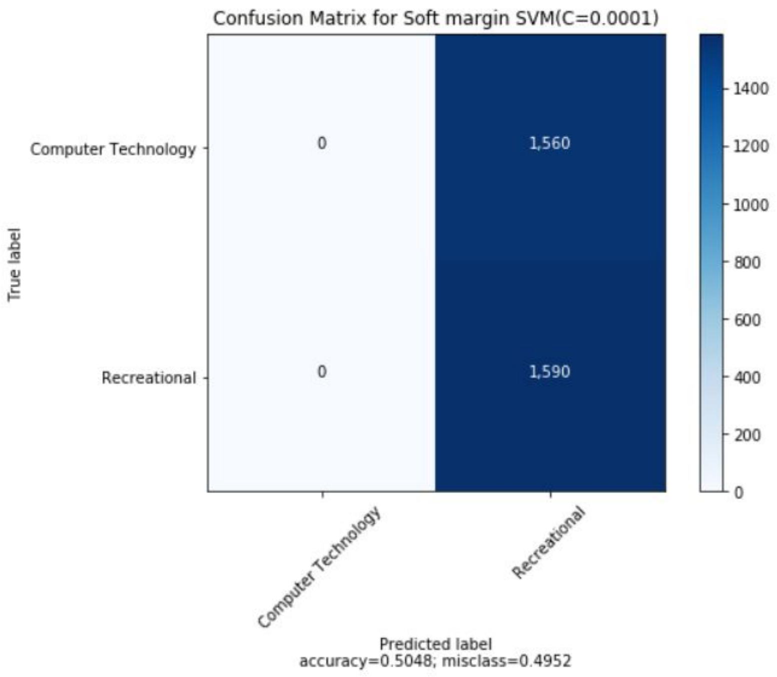
Recall = 1.0000

Precision = 0.5048

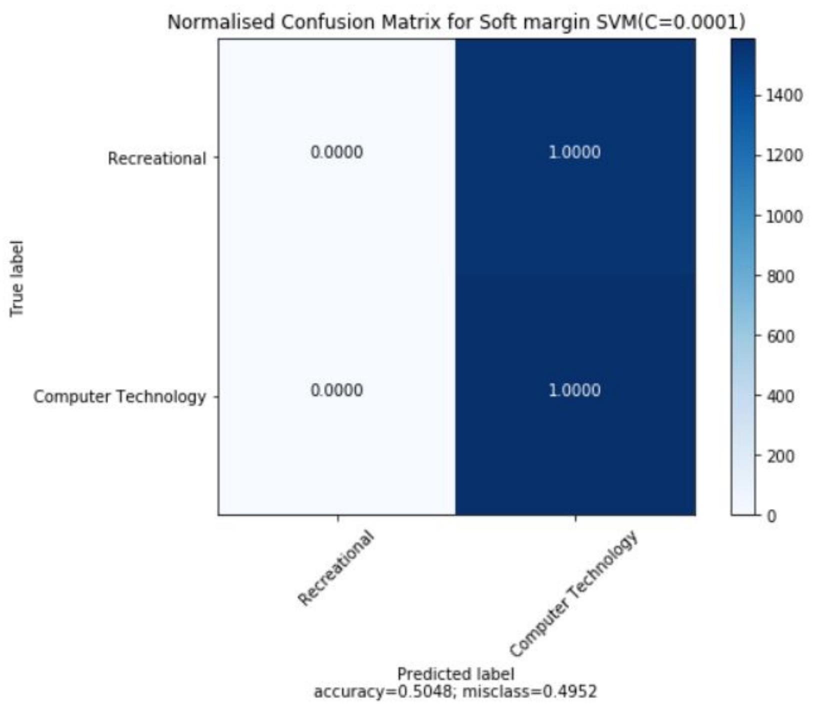
F1_score = 0.6709

Confusion Matrix:

Before nomalization:



Normalized confusion matrix:



After the comparison of the two evaluation reports, Hard margin SVM, with *C*= 1000, gives a better performance. Soft SVM, with *C* = 0.0001, performs not very well (the accuracy is only around 0.5). This is because the penalty is so small when C is too small. A very small penalty makes the algorithm tolerate more misclassification, thus a more loose classification result with lower accuracy. However the ROC curve of the hard and soft margin SVMs look very similar. This is because ROC evaluates the relationship between TPR and FPR. However ROC does not consider the FN and TN rates, which the other metrics measure.

- Use cross-validation to choose C

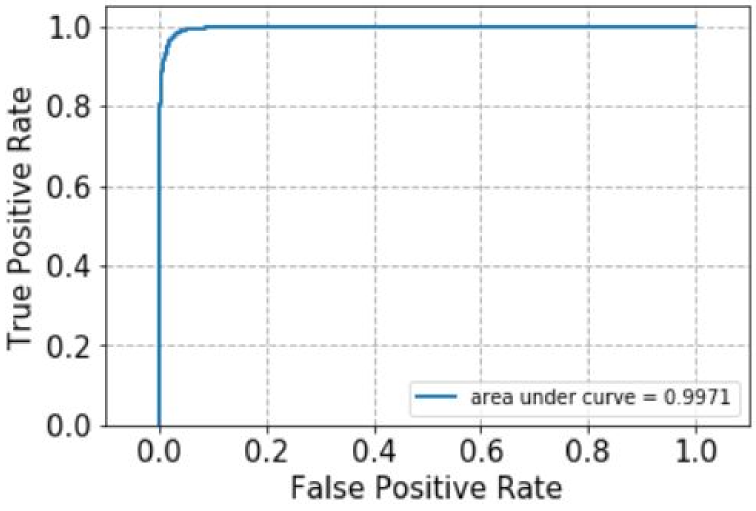
We need to find the best value of the parameter C in SVM classifier from {0.001,0.01,0.1,1,10,100,1000}. After applying 5-fold cross validation, we found the best value of the penalty C for SVM classifier is 100.

The average accuracy score of SVM cross validation for each of the C values from 0.001 to 1000 is as following:

```
[0.5048605721693318,
 0.5065512348196959,
 0.9666138207534539,
 0.9729520756332475,
 0.9780247333303496,
 0.9795035334653713,
 0.9788710709282405]
```

Hence we get the best accuracy is 0.9795, when C = 100. The evaluation reports of the model with highest performance (C = 100) are as follows:

ROC curve:



Accuracy, recall, precision, f1-score:

Accuracy = 0.9752

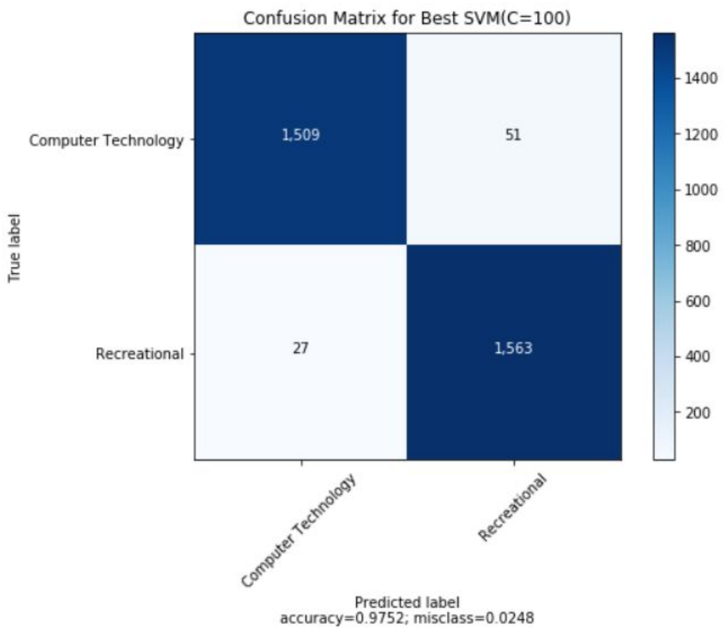
Recall = 0.9830

Precision = 0.9684

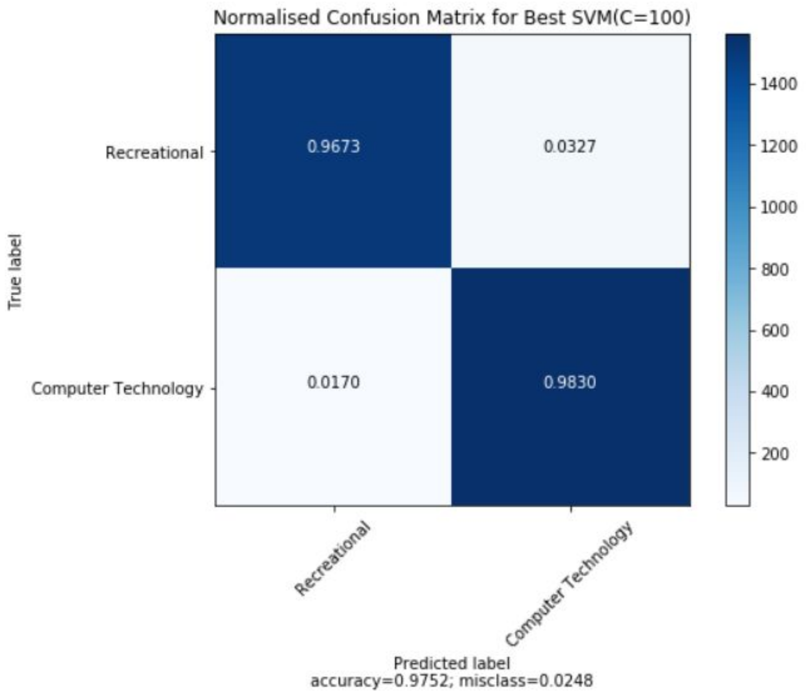
F1_score = 0.9757

Confusion Matrix:

Before normalization:



Normalized confusion matrix:



QUESTION 5: Logistic Regression

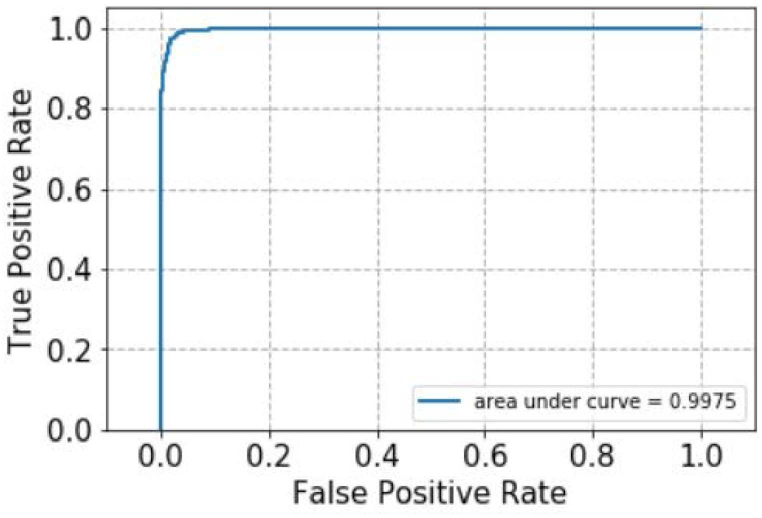
Logistic regression is a probability model that is used for binary classification. One can also add regularization term in the objective function, so that the goal of the training process is not only maximizing the likelihood, but also minimizing the regularization term, which is often some norm of the parameter vector. Adding regularization helps prevent ill-conditioned results and over-fitting, and facilitate generalization ability of the classifier. A coefficient is used to control the trade-off between maximizing likelihood and minimizing the regularization term.

In this section, we first conduct logistic classification without regularization. The parameter “C” in LogisticRegression is the inverse of regularization strength. Thus, to set strength to zero, we choose to set “C” to a very large number. In this way, the classifier is approximately logistic classification without regularization.

- Train a logistic classifier without regularization

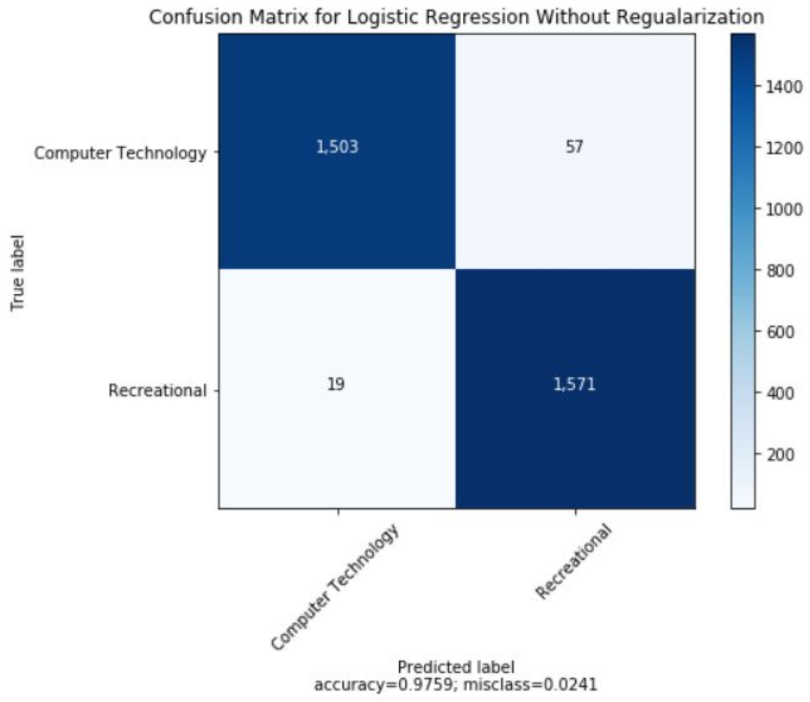
No regularization Logistic Regression

ROC curve:

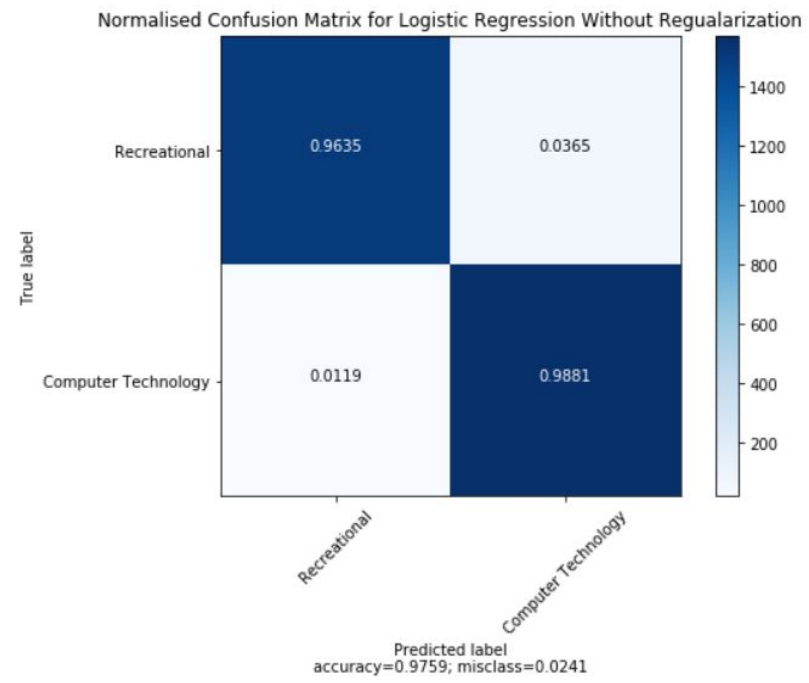


No regularization Confusion Matrix:

Confusion matrix before normalization



Normalized confusion matrix



No regularization Logistic Regression Accuracy, precision, recall and F-score:
Accuracy = 0.9759
Recall = 0.9881
Precision = 0.9650
F1_score = 0.9764

- Find the Best Regularization Parameter for L1 and L2

Then we used 5-fold cross-validation and find the best regularization strength for logistic regression with L1 regularization and L2 regularization. The average accuracies for different regularization parameters are shown below.

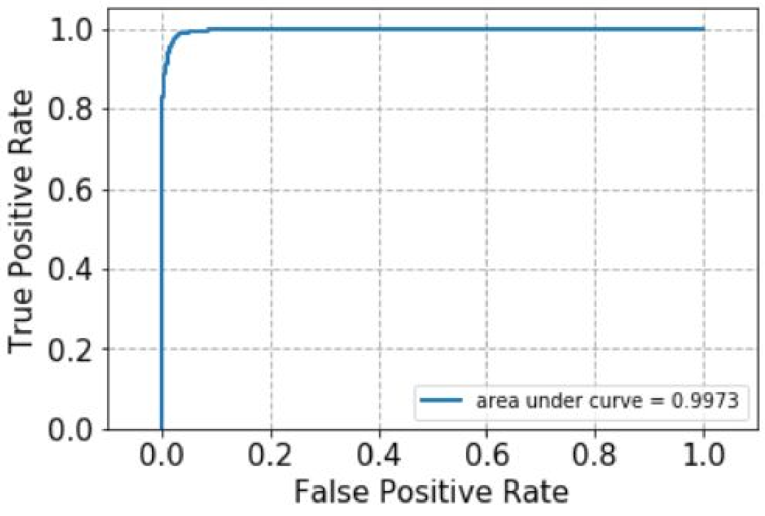
C	Regularization Strength	Average Accuracy for L1	Average Accuracy for L2
0.001	**1000**	**0.4951394278306682**	**0.7147079384711075**
0.01	**100**	**0.924774100754062**	**0.9480174212867418**
0.1	**10**	**0.9522455306827007**	**0.9649213697513102**
1	**1**	**0.974219234611661**	**0.9706278327849507**
10	**0.1**	**0.9773886969245629**	**0.9756993742387028**
100	**0.01**	**0.9769672039080944**	**0.9782348103288159**
1000	**0.001**	**0.9771788441197347**	**0.9771786208710646**

According to the table above, the best regularization strength for L1 is 0.1(C = 10) and the best regularization strength for L2 is 0.01(C = 100).

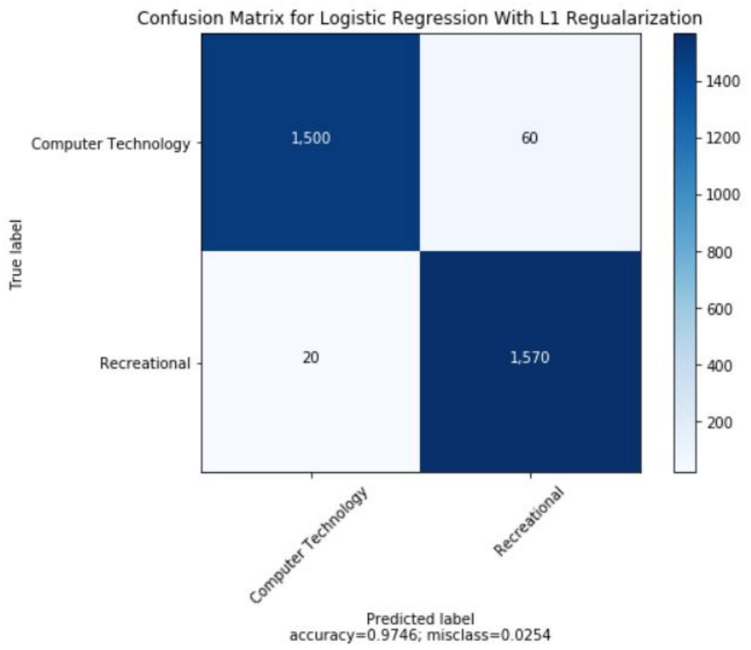
Logistic Regression with L1:

After we found the best regularization parameter for L1, we evaluated the trained model. The ROC curve, the confusion matrix, accuracy, precision, recall and F-score are shown below.

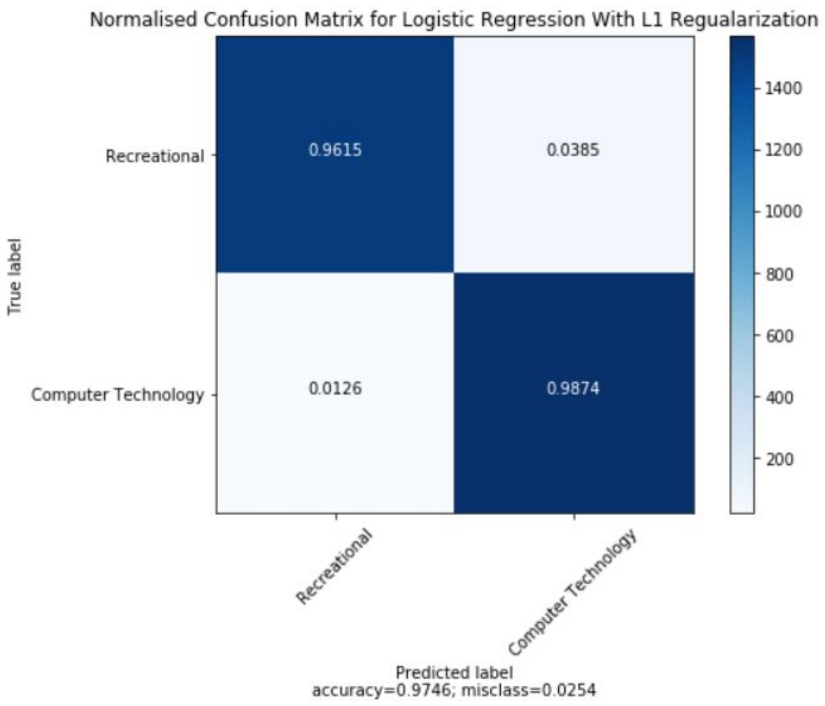
Regularization L1 ROC:



L1 Confusion Matrix:



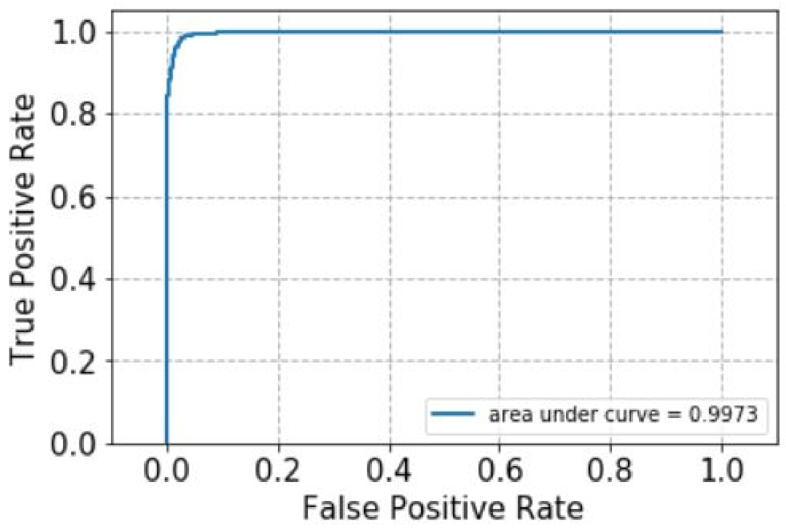
The ROC curve, the confusion matrix, accuracy, precision, recall and F-score are shown below.



L1 Accuracy, precision, recall and F-score:

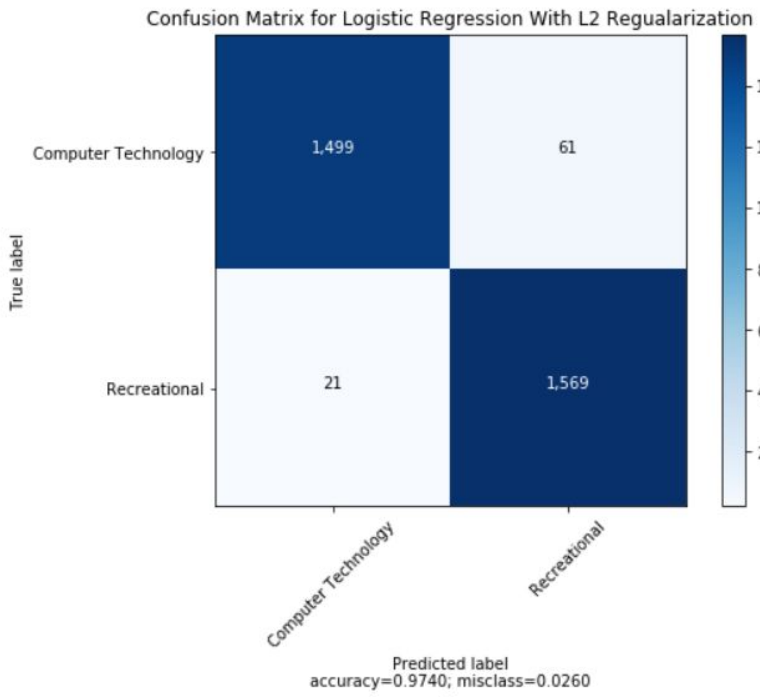
Accuracy = 0.9746
Recall = 0.9874
Precision = 0.9632
F1_score = 0.9752

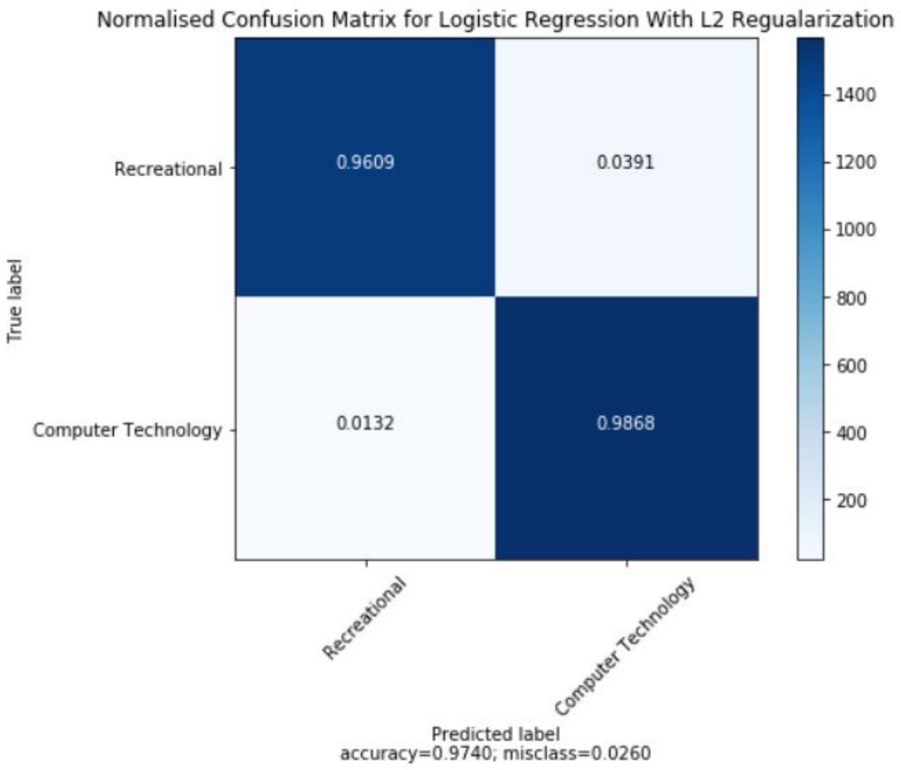
Regularization L2 ROC:



The ROC curve, the confusion matrix, accuracy, precision, recall and F-score are shown below.

L2 Confusion Matrix:





L2 Accuracy, precision, recall and F-score:

Accuracy = 0.9740
Recall = 0.9868
Precision = 0.9626

F1_score = 0.9745

- Compare the performance

	Accuracy	**Recall**	**Precision**	**F-1 score**
No regularization	**0.9759**	**0.9881**	**0.9650**	**0.9764**
L1 regularization	**0.9746**	**0.9874**	**0.9632**	**0.9752**
L2 regularization	**0.9740**	**0.9868**	**0.9626**	**0.9745**

- **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?**

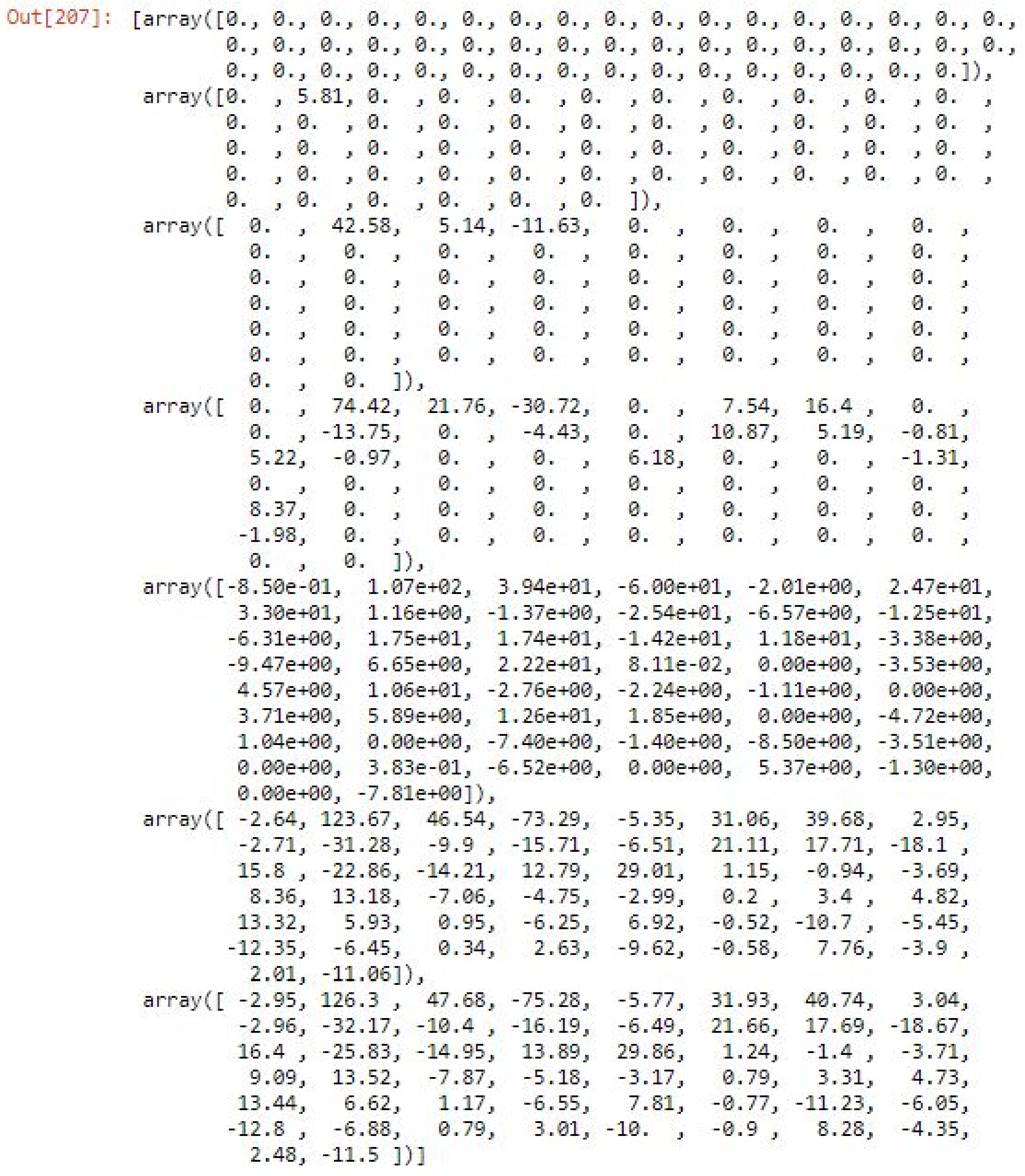
answer for part 1:

We can observe from the table listing the accuracies of different regularization strength that as the regularization strength becomes larger i.e. the parameter 'C' become smaller, the test error of the models increase sharply. As the regularization strength becomes smaller, the errors tend to be close.

answer for part 2:

We printed out the coefficient matrix of the models corresponding to L1 and L2 with different regularization strength as the figure shown below.

Coefficient Matrix of L1 regularization corresponding to regularization strength from 0.001 to 1000

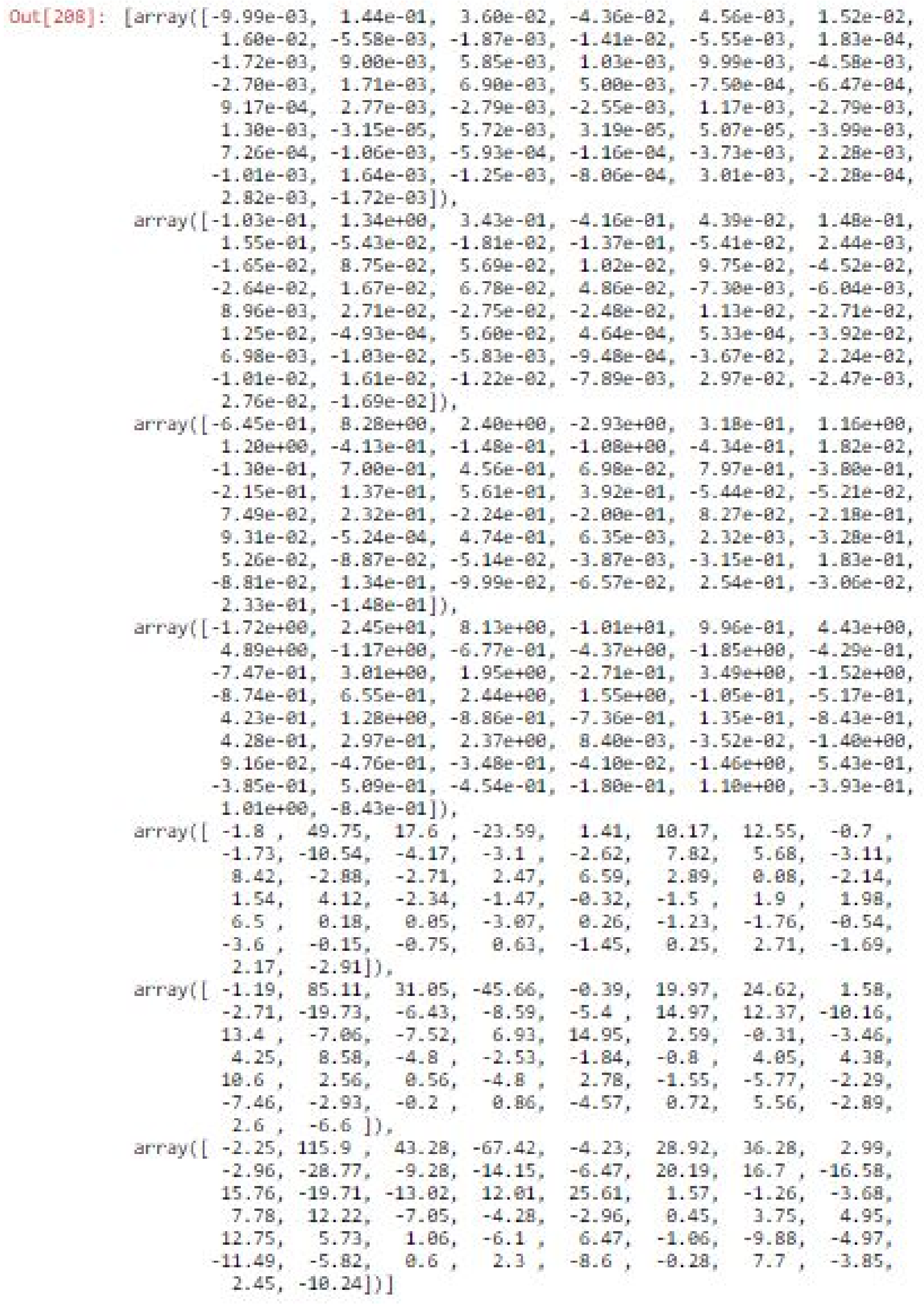


```
Out[207]: [array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                  0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                  0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
           array([0.  , 5.81, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ,
                  0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ,
                  0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ,
                  0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ,
                  0.  , 0.  , 0.  , 0.  , 0.  , 0.  ]),
           array([  0.  ,  42.58,   5.14, -11.63,   0.  ,   0.  ,   0.  ,   0.  ,
                    0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,
                    0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,
                    0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,
                    0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,
                    0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,
                    0.  ,   0.  ]),
           array([  0.  ,  74.42,  21.76, -30.72,   0.  ,   7.54,  16.4 ,   0.  ,
                    0.  , -13.75,   0.  ,  -4.43,   0.  ,  10.87,   5.19,  -0.81,
                    5.22,  -0.97,   0.  ,   0.  ,   6.18,   0.  ,   0.  ,  -1.31,
                    0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,
                    8.37,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,
                   -1.98,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,
                    0.  ,   0.  ]),
           array([-8.50e-01,  1.07e+02,  3.94e+01, -6.00e+01, -2.01e+00,  2.47e+01,
                   3.30e+01,  1.16e+00, -1.37e+00, -2.54e+01, -6.57e+00, -1.25e+01,
                  -6.31e+00,  1.75e+01,  1.74e+01, -1.42e+01,  1.18e+01, -3.38e+00,
                  -9.47e+00,  6.65e+00,  2.22e+01,  8.11e-02,  0.00e+00, -3.53e+00,
                   4.57e+00,  1.06e+01, -2.76e+00, -2.24e+00, -1.11e+00,  0.00e+00,
                   3.71e+00,  5.89e+00,  1.26e+01,  1.85e+00,  0.00e+00, -4.72e+00,
                   1.04e+00,  0.00e+00, -7.40e+00, -1.40e+00, -8.50e+00, -3.51e+00,
                   0.00e+00,  3.83e-01, -6.52e+00,  0.00e+00,  5.37e+00, -1.30e+00,
                   0.00e+00, -7.81e+00]),
           array([ -2.64, 123.67,  46.54, -73.29,  -5.35,  31.06,  39.68,   2.95,
                   -2.71, -31.28,  -9.9 , -15.71,  -6.51,  21.11,  17.71, -18.1 ,
                   15.8 , -22.86, -14.21,  12.79,  29.01,   1.15,  -0.94,  -3.69,
                    8.36,  13.18,  -7.06,  -4.75,  -2.99,   0.2 ,   3.4 ,   4.82,
                   13.32,   5.93,   0.95,  -6.25,   6.92,  -0.52, -10.7 ,  -5.45,
                  -12.35,  -6.45,   0.34,   2.63,  -9.62,  -0.58,   7.76,  -3.9 ,
                    2.01, -11.06]),
           array([ -2.95, 126.3 ,  47.68, -75.28,  -5.77,  31.93,  40.74,   3.04,
                   -2.96, -32.17, -10.4 , -16.19,  -6.49,  21.66,  17.69, -18.67,
                   16.4 , -25.83, -14.95,  13.89,  29.86,   1.24,  -1.4 ,  -3.71,
                    9.09,  13.52,  -7.87,  -5.18,  -3.17,   0.79,   3.31,   4.73,
                   13.44,   6.62,   1.17,  -6.55,   7.81,  -0.77, -11.23,  -6.05,
                  -12.8 ,  -6.88,   0.79,   3.01, -10.  ,  -0.9 ,   8.28,  -4.35,
                    2.48, -11.5 ])]
```

For L1 regularization, with the regularization strength becomes larger i.e. the parameter 'C' become smaller, the sparsity of the matrix increases, it is obvious that when the regularization strength is 1000, almost all entries in the matrix are zero, the model is underfitting. For L2 regularization, with the regularization strength becomes larger i.e. the parameter 'C' become smaller, the entries in the matrix become smaller and the model is underfitting.

For Logistic Regression model with both L1 and L2 regularization, when the regularization strength becomes smaller, the coefficient matrices tend to be similar to the model with no regularization. The reason is that the regularization does little on the model.

Coefficient Matrix of L2 regularization corresponding to regularization strength from 0.001 to 1000



```
Out[208]: [array([-9.99e-03,  1.44e-01,  3.60e-02, -4.36e-02,  4.56e-03,  1.52e-02,
                   1.60e-02, -5.58e-03, -1.87e-03, -1.41e-02, -5.55e-03,  1.83e-04,
                  -1.72e-03,  9.00e-03,  5.85e-03,  1.03e-03,  9.99e-03, -4.58e-03,
                  -2.70e-03,  1.71e-03,  6.90e-03,  5.00e-03, -7.50e-04, -6.47e-04,
                   9.17e-04,  2.77e-03, -2.79e-03, -2.55e-03,  1.17e-03, -2.79e-03,
                   1.30e-03, -3.15e-05,  5.72e-03,  3.19e-05,  5.07e-05, -3.99e-03,
                   7.26e-04, -1.06e-03, -5.93e-04, -1.16e-04, -3.73e-03,  2.28e-03,
                  -1.01e-03,  1.64e-03, -1.25e-03, -8.06e-04,  3.01e-03, -2.28e-04,
                   2.82e-03, -1.72e-03]),
            array([-1.03e-01,  1.34e+00,  3.43e-01, -4.16e-01,  4.39e-02,  1.48e-01,
                   1.55e-01, -5.43e-02, -1.81e-02, -1.37e-01, -5.41e-02,  2.44e-03,
                  -1.65e-02,  8.75e-02,  5.69e-02,  1.02e-02,  9.75e-02, -4.52e-02,
                  -2.64e-02,  1.67e-02,  6.78e-02,  4.86e-02, -7.30e-03, -6.04e-03,
                   8.96e-03,  2.71e-02, -2.75e-02, -2.48e-02,  1.13e-02, -2.71e-02,
                   1.25e-02, -4.93e-04,  5.60e-02,  4.64e-04,  5.33e-04, -3.92e-02,
                   6.98e-03, -1.03e-02, -5.83e-03, -9.48e-04, -3.67e-02,  2.24e-02,
                  -1.01e-02,  1.61e-02, -1.22e-02, -7.89e-03,  2.97e-02, -2.47e-03,
                   2.76e-02, -1.69e-02]),
            array([-6.45e-01,  8.28e+00,  2.40e+00, -2.93e+00,  3.18e-01,  1.16e+00,
                   1.20e+00, -4.13e-01, -1.48e-01, -1.08e+00, -4.34e-01,  1.82e-02,
                  -1.30e-01,  7.00e-01,  4.56e-01,  6.98e-02,  7.97e-01, -3.80e-01,
                  -2.15e-01,  1.37e-01,  5.61e-01,  3.92e-01, -5.44e-02, -5.21e-02,
                   7.49e-02,  2.32e-01, -2.24e-01, -2.00e-01,  8.27e-02, -2.18e-01,
                   9.31e-02, -5.24e-04,  4.74e-01,  6.35e-03,  2.32e-03, -3.28e-01,
                   5.26e-02, -8.87e-02, -5.14e-02, -3.87e-03, -3.15e-01,  1.83e-01,
                  -8.81e-02,  1.34e-01, -9.99e-02, -6.57e-02,  2.54e-01, -3.06e-02,
                   2.33e-01, -1.48e-01]),
            array([-1.72e+00,  2.45e+01,  8.13e+00, -1.01e+01,  9.96e-01,  4.43e+00,
                   4.89e+00, -1.17e+00, -6.77e-01, -4.37e+00, -1.85e+00, -4.29e-01,
                  -7.47e-01,  3.01e+00,  1.95e+00, -2.71e-01,  3.49e+00, -1.52e+00,
                  -8.74e-01,  6.55e-01,  2.44e+00,  1.55e+00, -1.05e-01, -5.17e-01,
                   4.23e-01,  1.28e+00, -8.86e-01, -7.36e-01,  1.35e-01, -8.43e-01,
                   4.28e-01,  2.97e-01,  2.37e+00,  8.40e-03, -3.52e-02, -1.40e+00,
                   9.16e-02, -4.76e-01, -3.48e-01, -4.10e-02, -1.46e+00,  5.43e-01,
                  -3.85e-01,  5.09e-01, -4.54e-01, -1.80e-01,  1.10e+00, -3.93e-01,
                   1.01e+00, -8.43e-01]),
            array([ -1.8 ,  49.75,  17.6 , -23.59,   1.41,  10.17,  12.55,  -0.7 ,
                   -1.73, -10.54,  -4.17,  -3.1 ,  -2.62,   7.82,   5.68,  -3.11,
                    8.42,  -2.88,  -2.71,   2.47,   6.59,   2.89,   0.08,  -2.14,
                    1.54,   4.12,  -2.34,  -1.47,  -0.32,  -1.5 ,   1.9 ,   1.98,
                    6.5 ,   0.18,   0.05,  -3.07,   0.26,  -1.23,  -1.76,  -0.54,
                   -3.6 ,  -0.15,  -0.75,   0.63,  -1.45,   0.25,   2.71,  -1.69,
                    2.17,  -2.91]),
            array([ -1.19,  85.11,  31.05, -45.66,  -0.39,  19.97,  24.62,   1.58,
                   -2.71, -19.73,  -6.43,  -8.59,  -5.4 ,  14.97,  12.37, -10.16,
                   13.4 ,  -7.06,  -7.52,   6.93,  14.95,   2.59,  -0.31,  -3.46,
                    4.25,   8.58,  -4.8 ,  -2.53,  -1.84,  -0.8 ,   4.05,   4.38,
                   10.6 ,   2.56,   0.56,  -4.8 ,   2.78,  -1.55,  -5.77,  -2.29,
                   -7.46,  -2.93,  -0.2 ,   0.86,  -4.57,   0.72,   5.56,  -2.89,
                    2.6 ,  -6.6 ]),
            array([ -2.25, 115.9 ,  43.28, -67.42,  -4.23,  28.92,  36.28,   2.99,
                   -2.96, -28.77,  -9.28, -14.15,  -6.47,  20.19,  16.7 , -16.58,
                   15.76, -19.71, -13.02,  12.01,  25.61,   1.57,  -1.26,  -3.68,
                    7.78,  12.22,  -7.05,  -4.28,  -2.96,   0.45,   3.75,   4.95,
                   12.75,   5.73,   1.06,  -6.1 ,   6.47,  -1.06,  -9.88,  -4.97,
                  -11.49,  -5.82,   0.6 ,   2.3 ,  -8.6 ,  -0.28,   7.7 ,  -3.85,
                    2.45, -10.24])]
```

answer for part 3:

For L1 regularization, the penalty term is the absolute value of the magnitude which is the square of the magnitude. From the matrix above we know that L1 tend to shrink the coefficient matrix by making the entries zeros, which do not produce a large computation complexity, thus L1 regularization can reduce the memory usage and speed up the computation speed. However, L2 regularization is expert at dealing with high correlated features and multi classification but the computation complexity and memory usage performance will be worse than L1 regularization.

- **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?**

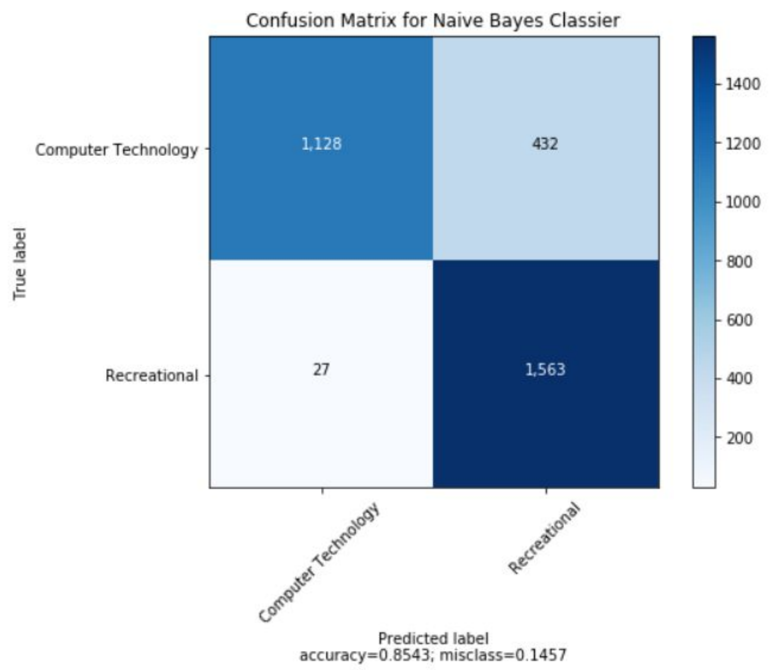
What SVM does is an attemptation to find a decision boundary that reduces the risk of error on the test data. It tries to find a decision boundary that has the same distance from the boundary points of both classes. Logistic Regression is to maximize the probability for correct classification. To use a silly analogy , LR tries to put the function 'through the points' while SVMs attempt to put support vectors 'between the points'.

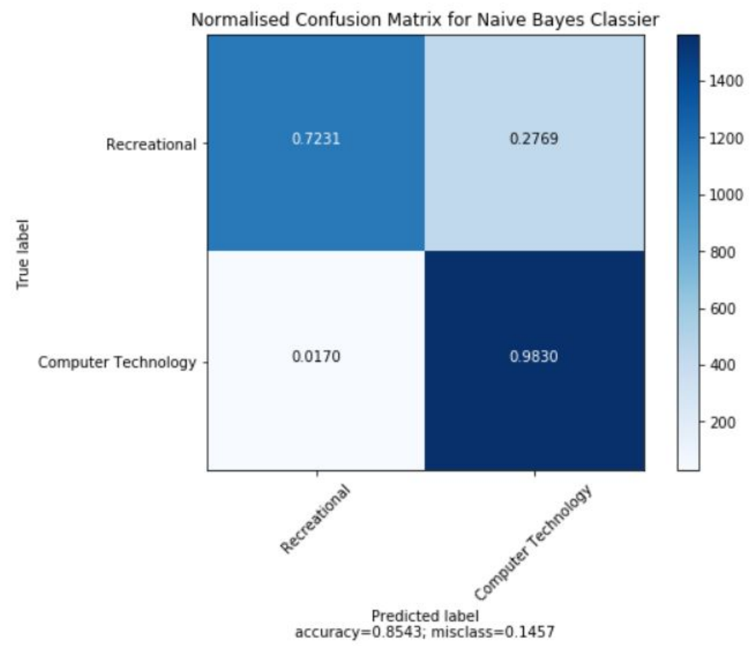
As for difference in performance, SVM is better at dealing with high dimensional data and LR is better at low dimensional data. In addition, SVM performs better than LR when high correlation structures are observed in the data. Furthermore, LR used all data to perform the classification but SVM only used those data that are support vectors.

QUESTION 6: Naive Bayes

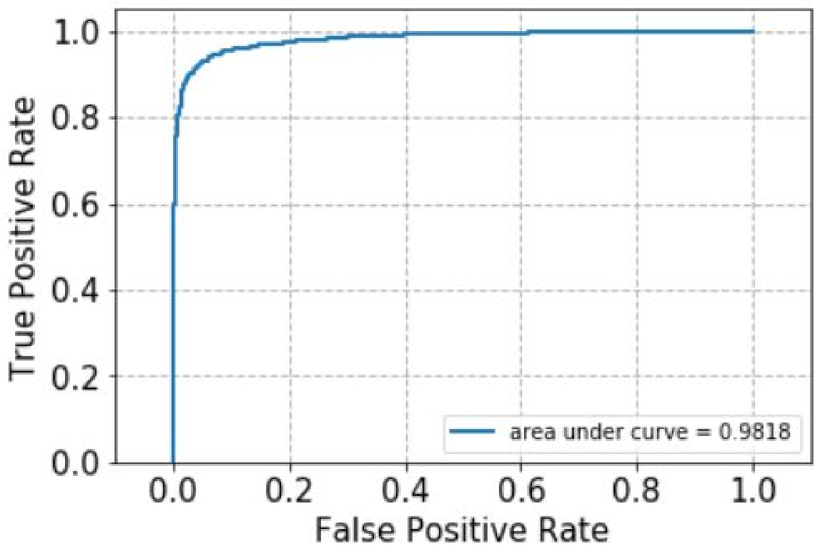
The following are the evaluations of the trained model of naive bayes classifier.

Confusion matrix:





ROC curve



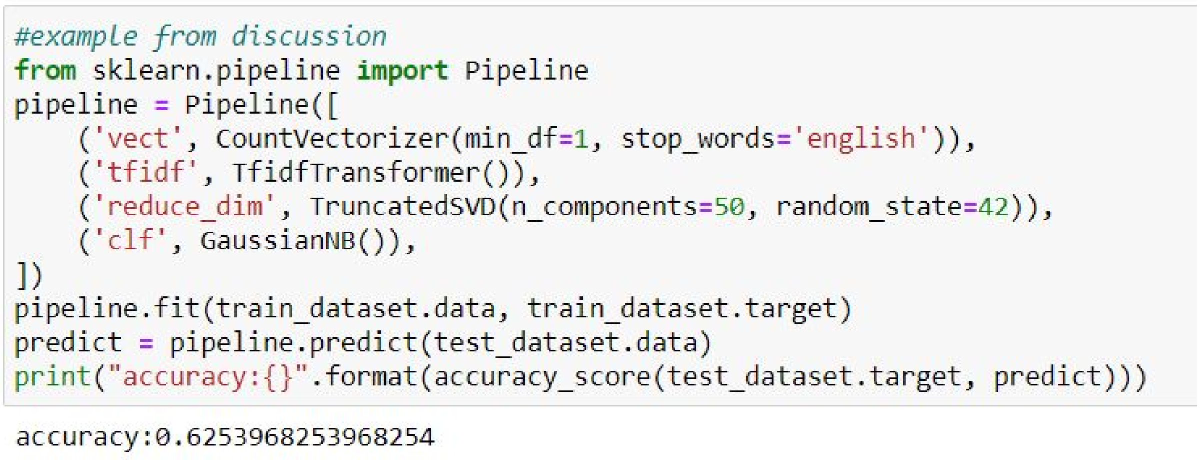
Gaussian Accuracy, precision, recall and F-score:

Accuracy = 0.8543
Recall = 0.9830
Precision = 0.7835
F1_score = 0.8720

V. Grid Search of parameters

QUESTION 7: Grid Search of parameters

Below is how the pipeline is constructed, including vectorization, feature extraction, dimensionality reduction and classification.



```
#example from discussion
from sklearn.pipeline import Pipeline
pipeline = Pipeline([
    ('vect', CountVectorizer(min_df=1, stop_words='english')),
    ('tfidf', TfidfTransformer()),
    ('reduce_dim', TruncatedSVD(n_components=50, random_state=42)),
    ('clf', GaussianNB()),
])
pipeline.fit(train_dataset.data, train_dataset.target)
predict = pipeline.predict(test_dataset.data)
print("accuracy:{}".format(accuracy_score(test_dataset.target, predict)))
```

```
accuracy:0.6253968253968254
```

In order to obtain the complete comparison in the table. Besides the vectorization, dimension reduction, the lemmatized vs not lemmatized, and keeping "headers" and "footers" vs having them removed are not presented in the pipeline above. Thus we chose to implement 4 training dataset with the combination of the 2 mentioned variables and train them through the pipeline separately. The 4 different training datasets formed by combinations are as follows:

#1. headremoved, lemmatized

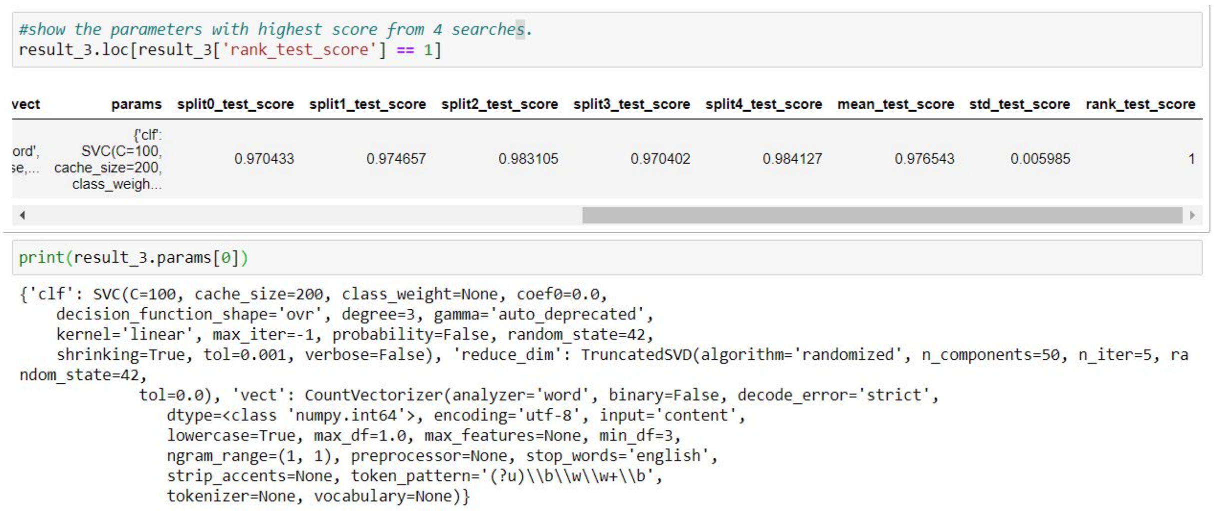
#2. headremoved, not lemmatized

#3. head not removed, lemmatized

#4. head not removed, not lemmatized

With each of the 4 datasets, we made a grid search with different min_df values (3 vs 5); different dimension reduction methods (LSI vs NMF); different classification methods (SVM vs L1 Logistic Regression vs L2 Logistic Regression vs Naive Bayesian). Thus each training dataset results in 2*2*4 = 16 combinations. And then in total there are 16*4 = 64 combinations.

After the 4 grid searches, we obtain the accuracy scores and each of their rankings presented in dataframes. Comparing all the 64 accuracy scores, we find the combination for highest score is:



```
#show the parameters with highest score from 4 searches.
result_3.loc[result_3['rank_test_score'] == 1]
```

vect	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	mean_test_score	std_test_score	rank_test_score
ord', se,...	{'clf': SVC(C=100, cache_size=200, class_weigh...	0.970433	0.974657	0.983105	0.970402	0.984127	0.976543	0.005985	1

```
print(result_3.params[0])
```

```
{'clf': SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=42,
    shrinking=True, tol=0.001, verbose=False), 'reduce_dim': TruncatedSVD(algorithm='randomized', n_components=50, n_iter=5, ra
ndom_state=42,
             tol=0.0), 'vect': CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                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='english',
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)}
```

- Keeping the ‘headers’ and ‘footers’
- Applying Lemmatization
- Vectorization with min_df = 3
- LSI Dimension Reduction
- SVM classification model with C = 100

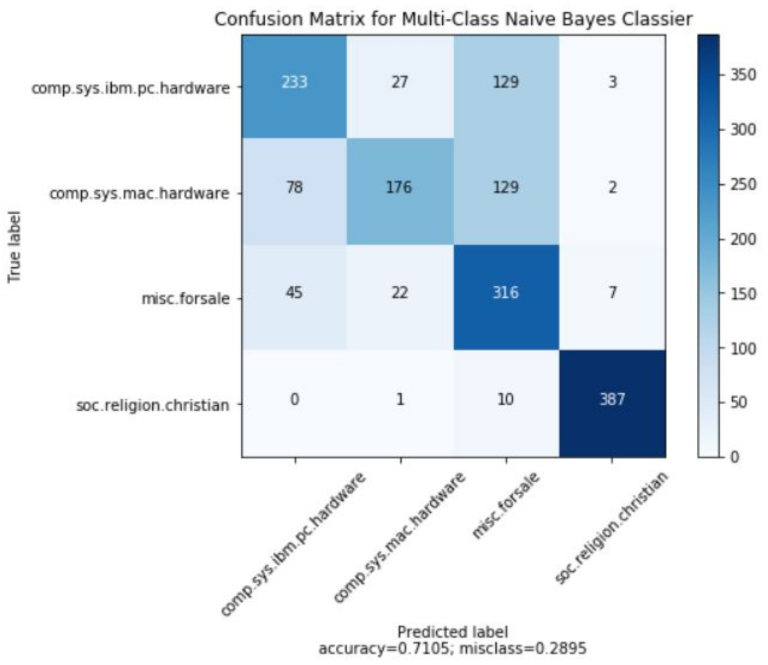
QUESTION 8: Multiclass classification

In this part we train Naive Bayes, SVM One vs One and SVM One vs Rest classifiers on the multiclass dataset. We then compare the performances of the three models on the scenario where multiple classes are present.

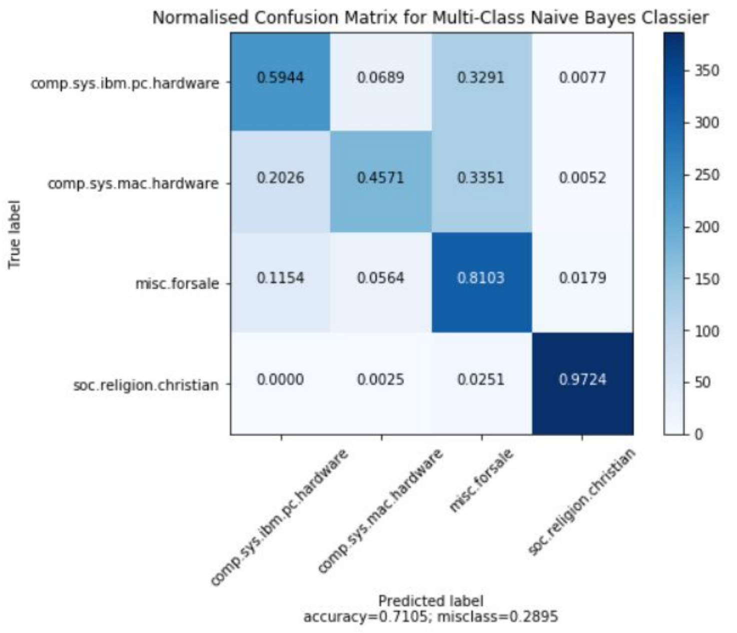
Naive Bayes:

For Naive Bayes, we still use the GaussianNB() classifier to train the dataset.

```
Confusion matrix, without normalization
[[233  27 129   3]
 [ 78 176 129   2]
 [ 45  22 316   7]
 [  0   1  10 387]]
```



```
Normalized confusion matrix
[[0.5944 0.0689 0.3291 0.0077]
 [0.2026 0.4571 0.3351 0.0052]
 [0.1154 0.0564 0.8103 0.0179]
 [0.     0.0025 0.0251 0.9724]]
```



Accuracy = 0.7105

Recall = 0.7105

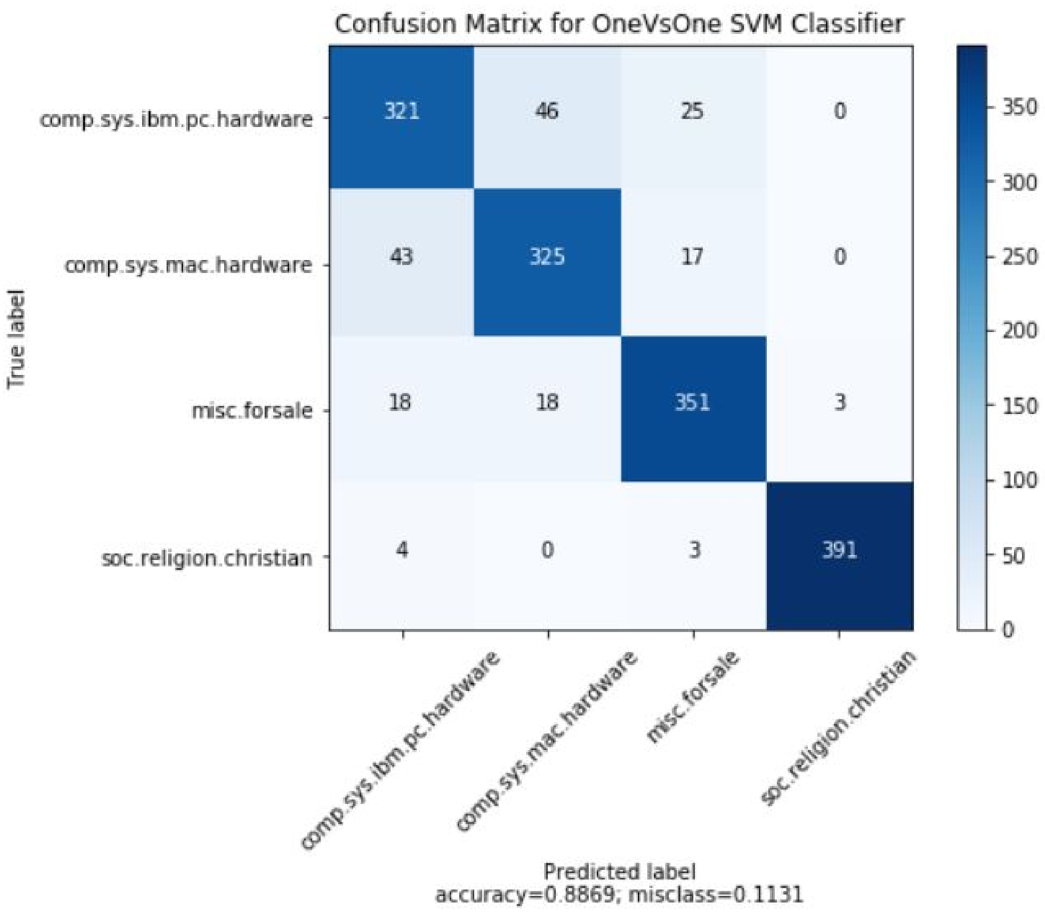
Precision = 0.7370

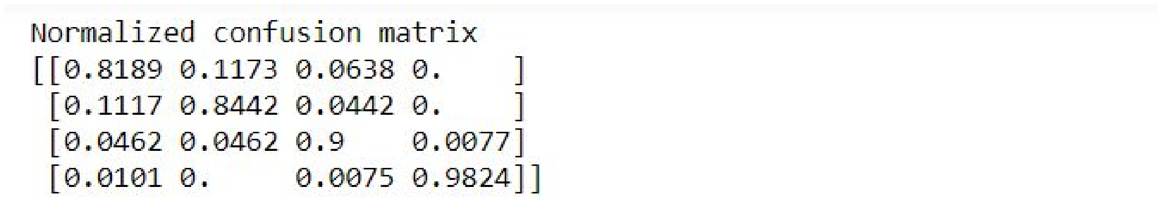
F1_score = 0.7064

SVM One vs One:

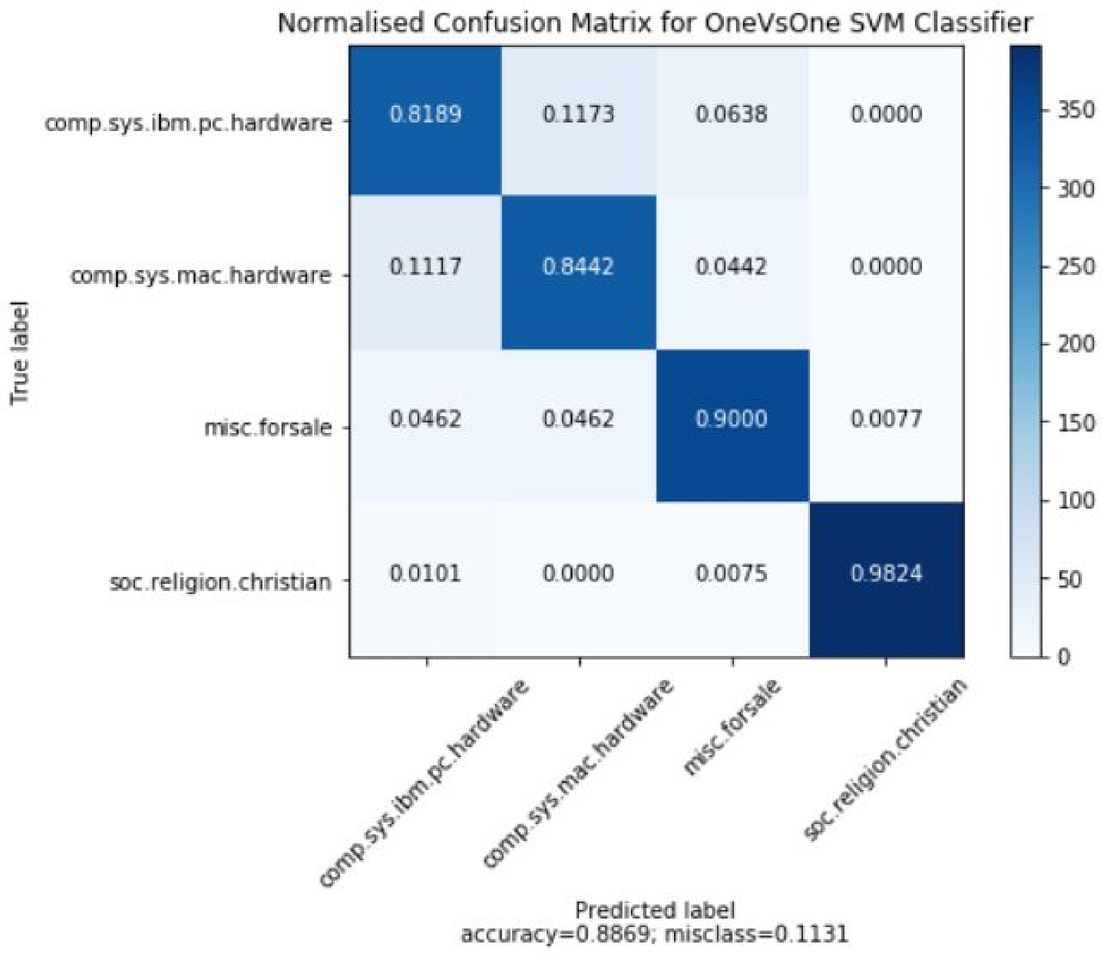
For SVM One vs One, as the name suggests, two pairs of classes are selected at a time and a binary classifier trained for them. This is done for every possible pair of classes thus there are n(n-1)/2 of them where n is the total number of classes. During the classification phase, all the binary classifiers are tested. For each of them, a "win" for one class is a vote for that class. The class with the most votes wins.

```
Confusion matrix, without normalization
[[321  46  25   0]
 [ 43 325  17   0]
 [ 18  18 351   3]
 [  4   0   3 391]]
```





```
Normalized confusion matrix
[[0.8189 0.1173 0.0638 0.    ]
 [0.1117 0.8442 0.0442 0.    ]
 [0.0462 0.0462 0.9    0.0077]
 [0.0101 0.     0.0075 0.9824]]
```

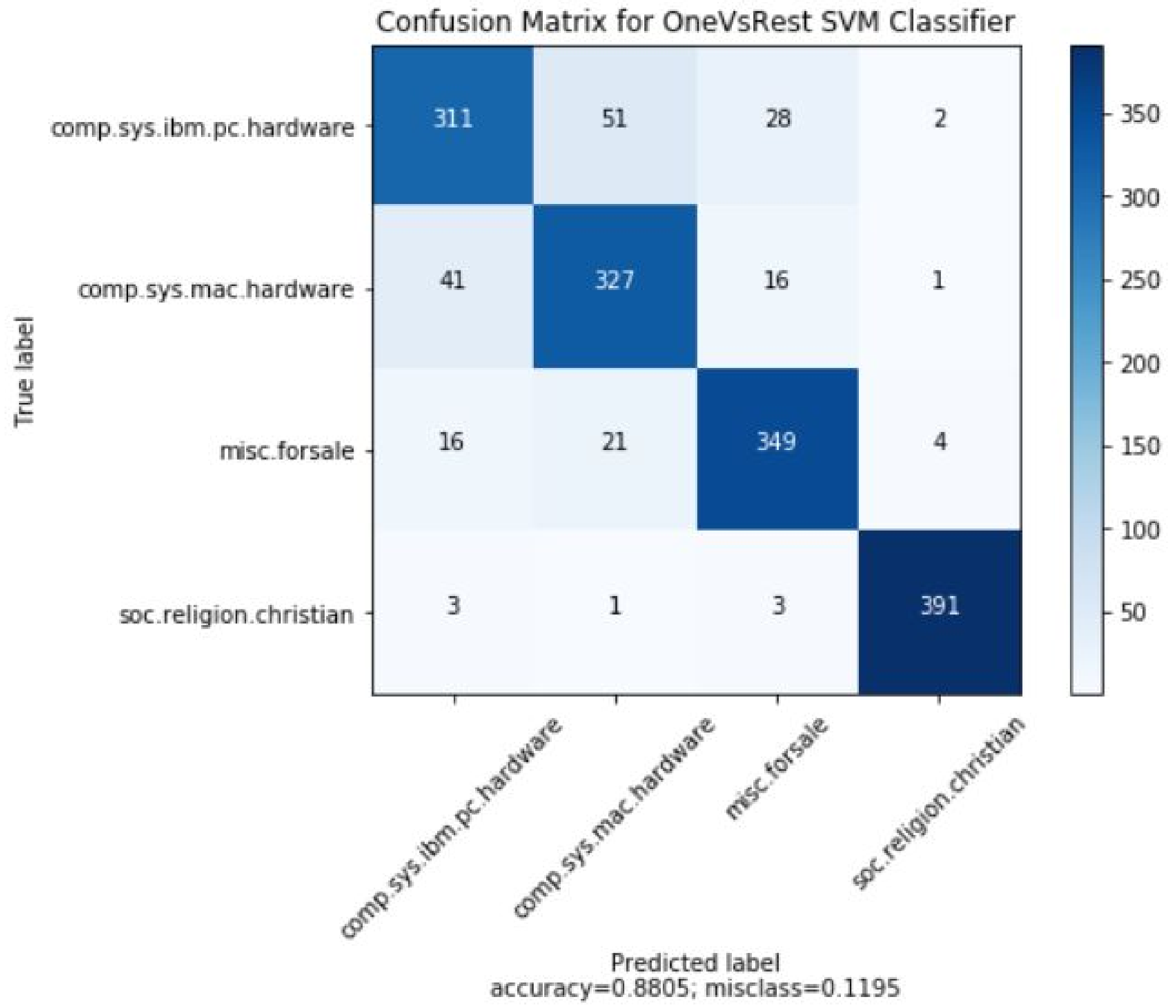


Accuracy = 0.8869
Recall = 0.8869
Precision = 0.8871
F1_score = 0.8870

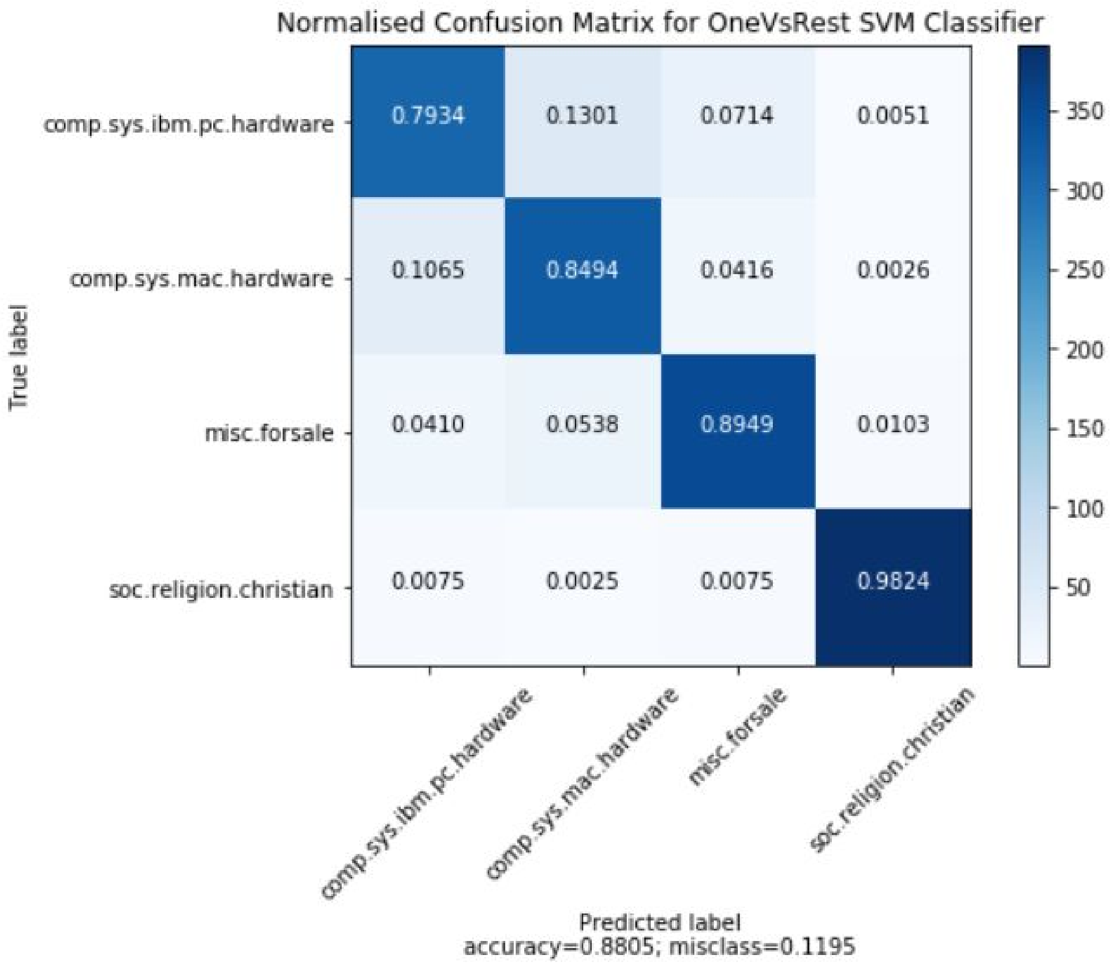
SVM One vs Rest:

For SVM One vs Rest, this approach takes one class as positive and rest all as negative to train the classifier. Thus for the data having n classes, n classifiers will be trained. During the classification phase, the n-classifier predicts probability of particular class and the class with the highest probability is selected.

```
Confusion matrix, without normalization
[[311  51  28   2]
 [ 41 327  16   1]
 [ 16  21 349   4]
 [  3   1   3 391]]
```



```
Normalized confusion matrix
[[0.7934 0.1301 0.0714 0.0051]
 [0.1065 0.8494 0.0416 0.0026]
 [0.041  0.0538 0.8949 0.0103]
 [0.0075 0.0025 0.0075 0.9824]]
```



Accuracy = 0.8805
Recall = 0.8805
Precision = 0.8805
F1_score = 0.8803

By comparison, SVM One vs One has the highest accuracy and thus is the most suitable for this multiclass classification. SVM One vs Rest also shows a better performance compared to Naive Bayes Classification.

VI. Conclusion

Through this project we became more familiar with the step-by-steps of how textual data are converted into matrix through tokenizing, lemmatization, vectorization and TF-IDF. We then learned to implement dimension reduction methods including LSI and NMF functions and verified that LSI generally has lower loss compared to NMF. We then tested and evaluated different classification models including SVM, logistic regression and Naive Bayes classifier. We learned to use various metrics to evaluate the model, such as accuracy, precision, f1_scroe etc. We also implemented Grid search methods to find the optimized model in a larger scale. Finally we trained different models for multiclass classification where the classification is not binary.