Assignment 3: Report

Training Dataset:

Enron-Spam Datasets were used for the purpose of training the spam-ham classifier. The dataset used is the "Enron2" set obtained from http://nlp.cs.aueb.gr/software_and_datasets/Enron-Spam/index.html. The Enron2 directory consists of 1496 spam emails and 4361 ham emails both as txt files. However, some ham email text files contain email threads (i.e: forwarded,replies ,etc). The spam mails are stored in "spam" folder and the ham emails are stored in "ham" folder under a folder named "data" in the working directory of the jupyter notebook "Training.ipynb" in which the code for the training of classifier is written.

Pre-processing:

For the purpose of training the classifier, only the words in the SUBJECT and the BODY of the emails were considered. So, from both the "spam" folder and "ham" folders, txt files are read one by one, converted to lower case string elements and also removing the words - "subject:", "subject: re", "subject: re: fw" etc. as only the content of the SUBJECT and the BODY of the email are considered. As mentioned before some ham files contain threads which are also segregated and considered as separate emails/datapoints.

There were 6 steps involved in the pre-processing:

a) symbol_color_codes removal: Punctuation marks and some symbols frequently encountered in the the emails are removed and replaced by " "(spaces), as they don't contribute to the main content of the email. Some spam emails also contained hex colour codes, which were also removed and replaced by ""(empty string) ,with of color_codes.txt file containing the hex colour codes frequently encountered in the mails.

b) Number tokens: All the digits from [0-9] are also removed and replaced by " " , as both ham and spam contain fair share of digits and doesn't help much in distinguishing them apart.

c) Lemmatisation: Words with their suffices are converted back to their root words, (e.g) running,ran to run and going,went,go to go etc, with the help of lemma.txt files which contain the root words corresponding to a given word. The lemma.txt file is obtained from https://raw.githubusercontent.com/skywind3000/lemma.en/master/lemma.en.txt . Out of 84,487 lemma groups, only 20,000 groups are considered and converted to python dictionary("lemma_dictionary") as the first 20,000 lemma groups almost covers all the common/most frequent word groups.

d) The "data" folder also consists of "stop_words.txt" file with English stop words, which are removed from the email datasets. (obtained from https://www.ranks.nl/stopwords).

e) The "data" folder consists of "html_words.txt" which contain some common html tags & attributes and css attributes, compiled from the Internet. These words are also removed from the emails as a pre-processing step.

f) Considering some emails could be in rich text format, the frequently occurring commands/ rtf words are put together "rtf_words.txt" file and these words are also removed from the emails.

Feature-Extraction:

TF-IDF vectorisation:

Once each email is pre-processed, it is stored as a python list consisting of strings/relevant words which could potentially help in classifying them. The lists are converted to a vector, via TF (term frequency)-IDF (inverse document frequency) vectorisation. The list of email are converted to a python set, inorder to extract only distinct words from the emails/documents. Once the Bag of words are ready, IDF vector is computed with dimension equal

to the number of words in the python set/bag of words and each component is equal to log(total number of documents or emails/ number of documents which contain the particular word in the bag). It's a measure of rarity of occurrence of each word in the word_ bag. Rare (less number of documents in which the word is found) the word in the training email dataset, higher is the IDF component corresponding to that word.

Similarly, for each email in our dataset TF (Term Frequency) vectors where each component is equal to (number of occurrence of that particular word/Total number of words in the document). The dimension of TF vectors is also equal to the number of words in word_bag.

Finally, each email in the training set is converted to a vector by multiplying elementwise, TF[i]*IDF, where "i" denotes the ith email in our dataset.

Thus, the features are extracted and the list of words are converted to vectors for each mail in the training set. This array of vectors now forms our Dataset over which classifier is trained.

After processing and TF-IDF vectorisation:

- Number of spams=1496
- Number of hams=6118
- Feature vector dimension=no. of element in bag_of_words=34737

Handling Imbalances:

With the spam: ham emails ratio being 1:4, it is possible that the classifier learnt is not a good one and is biased towards the ham class. Inorder to handle this imbalance SMOTE (Synthetic minority Over Sampling technique) is used over the minority class, in this case the spam class.

The SMOTE finds the K=5 nearest neighbours for each datapoint in the spam class and randomly picks certain number of points(4/1 -1 in this case) from these neighbours such that at the end of process the number of datapoints in the spam/minority class is almost equal to the number points in the ham class. Once a neighbour is randomly picked new datapoint is created by linear interpolation btw the datapoint and neighbour with a random weight "c" belonging to (0,1], (i.e) new_datapoint= datapoint[i] +c*(datapoint[i]- neighbour).

After SMOTE:

- Number of spam datapoints=5984
- Total number of datapoints=12102
- Feature vector dimension=34737

Training Algorithm:

For the balanced dataset (after SMOTE), a linear SVM(withC=10) is chosen for training the classifier which is done using LinearSVC class from the "scikit-learn" python module. SVM is chosen as the hinge loss error function forms a strict upper bound on the 0-1 loss which we would like to minimise. Also, the LinearSVC offers better & faster fit on sparse datasets, like the TF- IDF dataset that we have generated from the emails. Also, other polynomial / RBF kernels might result in over fitting of the training data.

The fitted model is stored as "svm_classifier.sav" file along with the Training Dataset (TF-IDF vectorised), the labels ,the lemma_dictionary and the bag_of_words list which are saved in the working directory of the notebook as "X.npy","y.npy", "lemma_dictionary.npy" and "bagofwords.npy" respectively.

Cross-validation: (code in "Cross_validation1.ipynb" for part a," Cross_validation2.ipynb" for part b)

a) In order to determine the best possible value for the "C" (slack/bribe) parameter of the Linear SVM, cross validation is done by splitting the training dataset (TF-IDF vectorised and SMOTEd), randomly into 70% training and 30% validation set.
   Various ranges of C were used, results of which are attached below:

The best C: 10.0

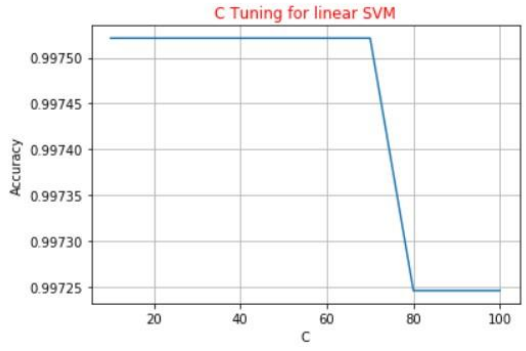


Figure 1: range: 10 to 100(steps of 10)

The best C: 100.0

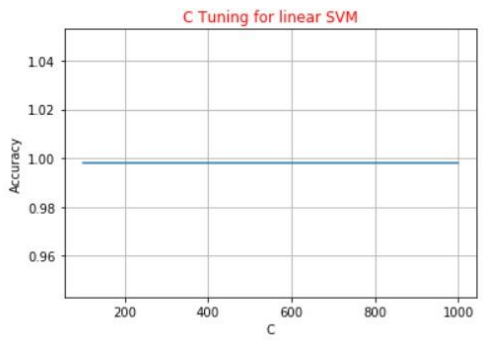


Figure 2: range 100 to 1000 (step size =100)

The best C: 4.25

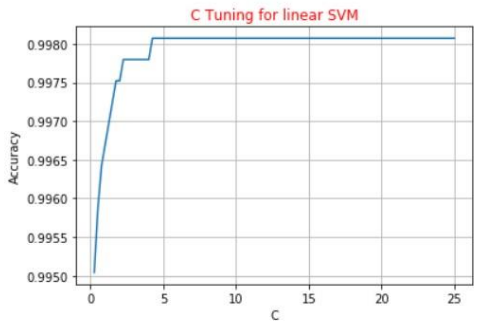
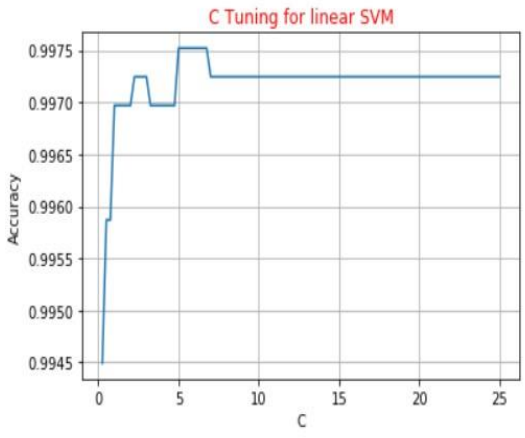


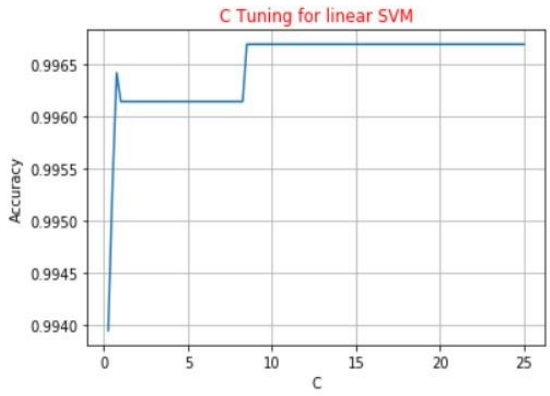
Figure 3: range 0.25 to 25 (step size=0.25)

From the results, we see that for large values of C>50 the accuracy drops and stays relatively the same, when compared to smaller values of C<=25. Because the split is done randomly, we repeat the cross-validation for C in range of 0.25 to 25 and some of the results observed are attached.

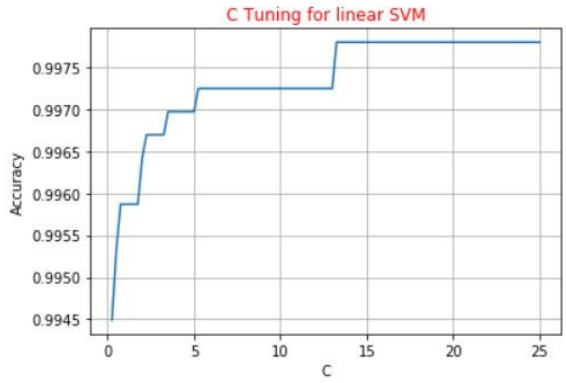
The best C: 5.0



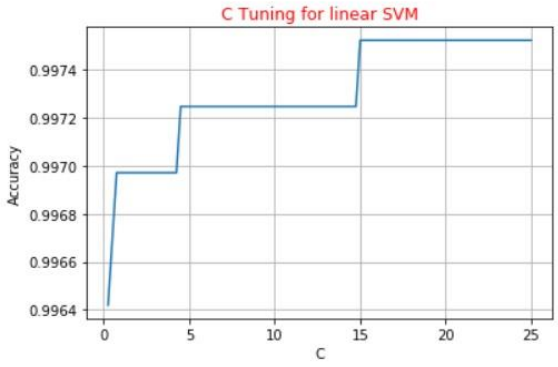
The best C: 8.5



```
The best C:  13.25
```




```
The best C:  15.0
```



On an average, we can notice that that the accuracy stays the same beyond 15 and the best possible C usually lies between 5.5 to 13.5/14.5. Therefore, "C=10 "is chosen for our classifier.

We also note the accuracies are very high like 0.99 for almost all values of C and a major reason for this kind of splitting for TF-IDF dataset is a naïve one, as the validation data's TF-IDF vectors are also computed along with the training data's TF-IDF with a common bag of words.

b) As a fix for the above validation method, we split the pre-processed emails before TF-IDF vectorisation as 70% training set and 30% validation set, in such a way that the proportion of spam and ham emails in the two sets are same as in original data. Then the bag_of_words list is created only using the training set, after which the TF-IDF and SMOTE -ing is done over the training set. Linear SVM with C=10 (as chosen above) is used to train the classifier and prediction is done over the validation set (after TF-IDF vectorisation), giving an accuracy of about 98.38%.

```
]: lsvc=svm.LinearSVC(C=10,max_iter=10000)
   lsvc.fit(X_train,y_train)
   y_pred=lsvc.predict(X_test)
   accuracy=np.sum(1-np.abs(y_pred-label_test))/len(label_test)
   print(accuracy)

   0.9938730853391685
```

Prediction: (code in "Prediction.ipynb")

This main() function in "Prediction.ipynb" notebook will perform the prediction on the new, unseen test dataset(emails), stored under the "test" folder in working directory . The main() function will load the fitted model/ Linear svm classifier(as trained over the enron dataset through "Training.ipynb" code) which is saved as "svm_classifier.sav" in the working directory of the notebook. The email#.txt files in test folder are read one by one(in the ascending email number# order) and are processed similar to the training set except now only the words in the bag_of_words list (previously saved as "bagofwords.npy" file during training) are considered and other terms/ words are removed or omitted, which are then TF-IDF vectorised using the bag_of_words. The main function then makes the prediction over the test dataset(X_test- with each feature vector having the same dimension as the training datapoints=34737) and prints the predicted labels (corresponding to each email) as a numpy array. The function also returns the predicted label array.