Lab Course Machine Learning
Exercise Sheet 9
Krithika Murugesan(277537)

Exercise 1: A spam filter using SVM

Data : Spam base dataset

The dataset is based on the e-mail text, which the the mail has to be classified as ham or spam. The first forty eight variables are the percentage of words that appear in the text and six variables are percentage of characters in the mail. There are three variables which denote the length of various sequences and the dependent variable.

The given data can be converted into LibSvm using the LibSvm and Sklearn packages. The following simple python code snippet was used to convert the data frame into the required format as LibSvm was tedious to set-up even with several tries.

```python
#Convert DF to libsvm format by normal python code
with open('libSvmTrain.dat','w') as f:
    for j in range(xTrain.shape[0]):
        for i in range(xTrain.shape[1]):
            if xTrain.iloc[j][i] != 0:
                f.write(" ".join([str(int(yTrain.iloc[j]))] + ["{}:{}".format(i,xTrain.iloc[j][i])]))

with open('libSvmTest.dat','w') as f:
    for j in range(xTest.shape[0]):
        for i in range(xTest.shape[1]):
            if xTest.iloc[j][i] != 0:
                f.write(" ".join([str(int(yTest.iloc[j]))] + ["{}:{}".format(i,xTest.iloc[j][i])]))
```

The data is split into test and train; after which it is formatted to two separate text files. Using python package is a more efficient option though

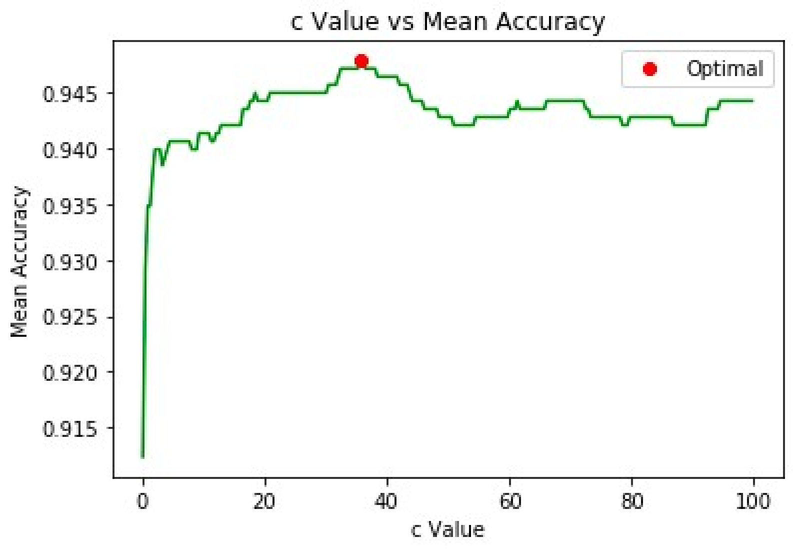
```python
#Converting input to libSvm format
dump_svmlight_file(train.iloc[:,0:53],train.iloc[:,-1],"libSvmTrain.dat")
dump_svmlight_file(test.iloc[:,0:53],test.iloc[:,-1],"libSvmTest.dat")

#Reading from libSvm format to use in scikit
dataTrain = load_svmlight_file("libSvmTrain.dat")
dataTest = load_svmlight_file("libSvmTest.dat")
```

The output files are of the following format,

```
1 0:0.52 1:0.42 2:0.35 4:0.14 5:0.03 6:0.03 7:0.1 8:0.5600000000000001 9:0.8 10:0.28 11:0.7 12:0.5600000000000001 13:1.19
14:0.03 15:0.24 16:0.45 18:3.18 20:1.47 22:0.38 23:0.63 44:0.07000000000000001 49:0.075 51:0.452 52:0.528
0 11:1.38 18:2.77 26:1.38 49:0.228
1 0:0.71 2:0.35 4:0.17 5:0.17 6:0.35 9:0.35 10:0.17 11:0.53 15:0.35 16:0.71 17:0.35 18:3.76 20:1.97 23:0.53 49:0.029 51:0.234
52:0.029
```

Where the first value is that of the dependent variable and the key value pairs are parameter and probability of parameter occurrence respectively. Once, we have this file, it can be used as input to SVM function in python to learn the model.

To obtain the best c-value, the accuracy is measured for a range of c values, and the value with maximum accuracy is used for prediction. The results are as follows:



```
accuracy      0.947864
c            35.700000
Name: 89, dtype: float64
```

Further studying the results,

```
Model summary
              precision    recall  f1-score   support

        0.0       0.95      0.97      0.96       856
        1.0       0.94      0.92      0.93       525

avg / total       0.95      0.95      0.95      1381


Confusion Matrix
 [[827  29]
 [ 43 482]]

Accuracy
 0.947863866763
```

Accuracy of 94.79% is achieved with best c Values and other default parameters.

Part B: Pre-processed a dataset and learn SVM

Data : Text Messages

It is corpus of messages, it can be classified as similar to e-mails as spam and ham. As the original text is given, it cannot be directly used in models. Pre-processing is done in following steps:

i) **Stemming** : The words are reduced to the ground form to avoid considering the same verb in different tenses as different words

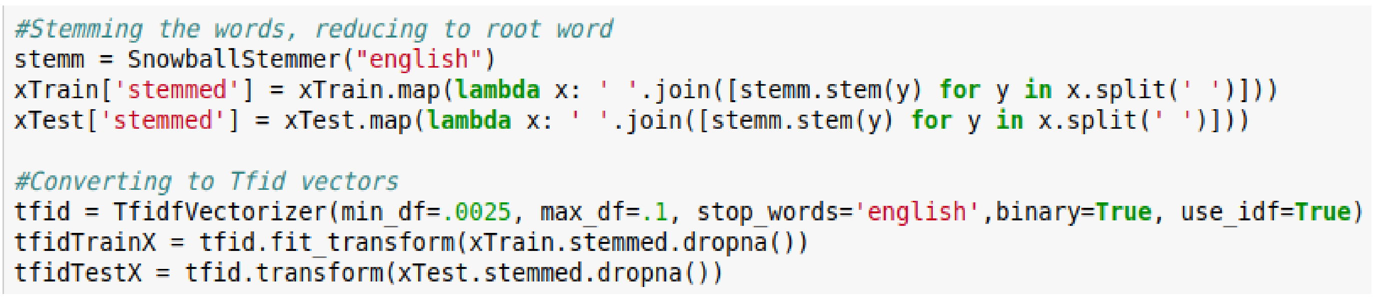
ii)**Vectorization** : As the data is only text doing hot one encoding will be space and time expensive. Text Analytics used different methods to solve this issue namely,

a) Bag of words : The distinct words are obtained from the text and its frequencies are calculated, this is a direct representation of the text in a programmable format

b)TFID : Term frequency - Inversed Document Frequency. It is one of the term-weighting schemes. The term frequency is proportional to the frequency of the important words and

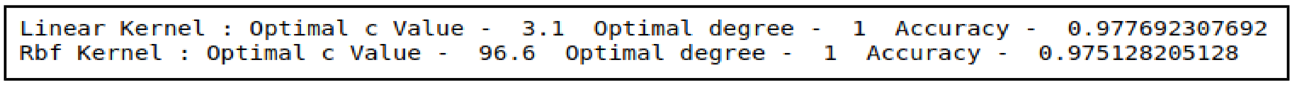
Inverse Document Frequency diminishes the weight for words with less importance. The TFID is the product of these two measures. TFID is used as it a better representation of data

iii)**Stop words** : Words like ‘is’,’a’,’the’ do not contribute much to the training of text models, such stop words are also removed to increase computational efficiency.

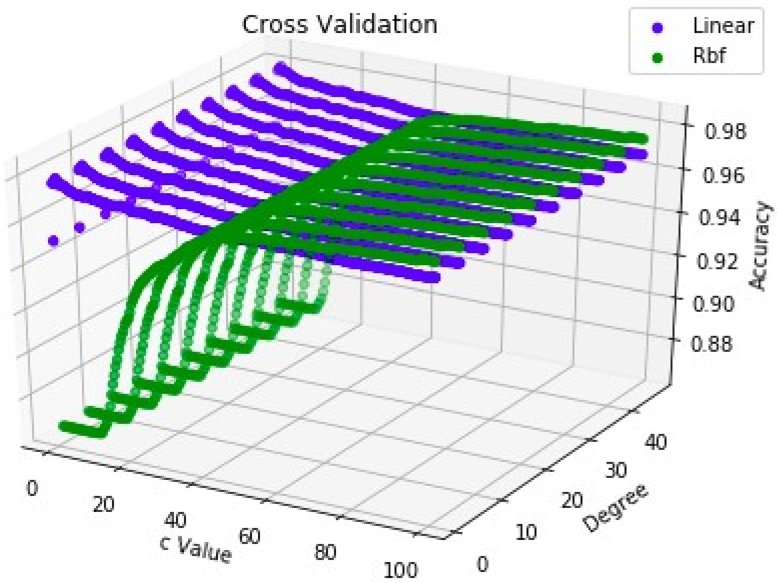


```
#Stemming the words, reducing to root word
stemm = SnowballStemmer("english")
xTrain['stemmed'] = xTrain.map(lambda x: ' '.join([stemm.stem(y) for y in x.split(' ')]))
xTest['stemmed'] = xTest.map(lambda x: ' '.join([stemm.stem(y) for y in x.split(' ')]))

#Converting to Tfid vectors
tfid = TfidfVectorizer(min_df=.0025, max_df=.1, stop_words='english',binary=True, use_idf=True)
tfidTrainX = tfid.fit_transform(xTrain.stemmed.dropna())
tfidTestX = tfid.transform(xTest.stemmed.dropna())
```

Performing 5-fold cross validation on the training dataset for c-value, degree and type of kernel, the results are



```
Linear Kernel : Optimal c Value -  3.1  Optimal degree -  1  Accuracy -  0.977692307692
Rbf Kernel : Optimal c Value -  96.6  Optimal degree -  1  Accuracy -  0.975128205128
```



It can be seen that the Linear kernel with c-value of 3.1 and degree of 1 has maximum accuracy.

Exercise 2: Compare SVM based spam filter with Decision Tree

Doing five-fold Cross validation for Decision Tree, the best parameters are

criterion = 'entropy'

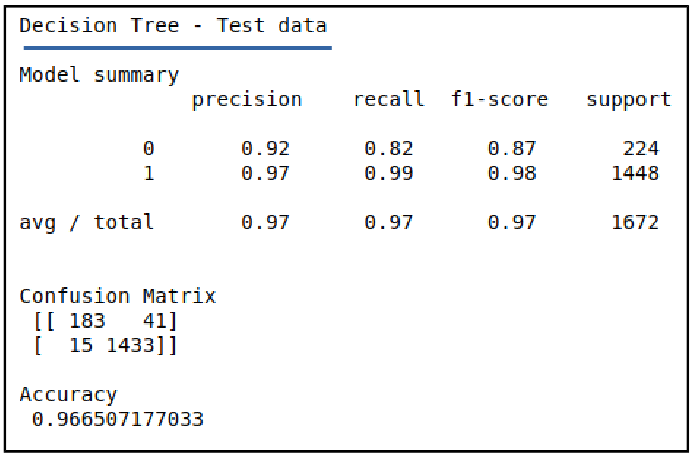
max_depth = 20

max_leaf_nodes = None

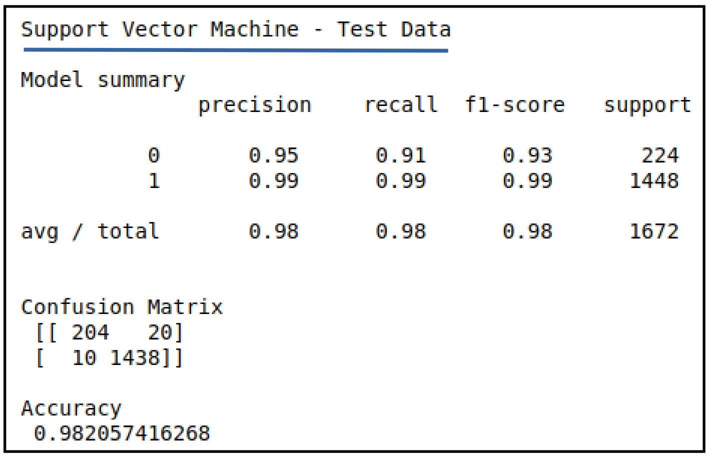
min_samples_leaf = 1

min_samples_split = 2

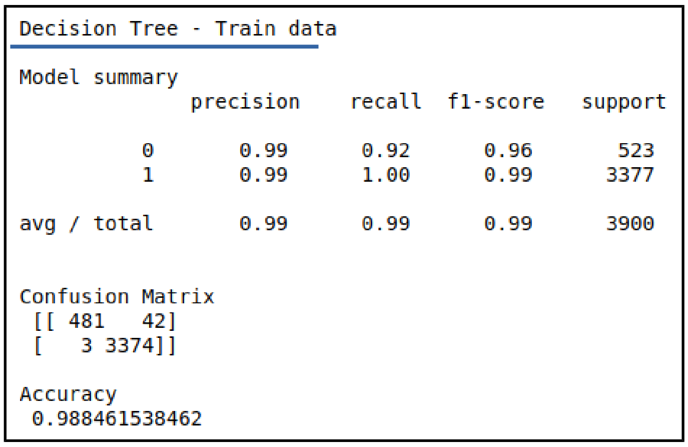
Using the above trained SVM model and decision tree model, the results are



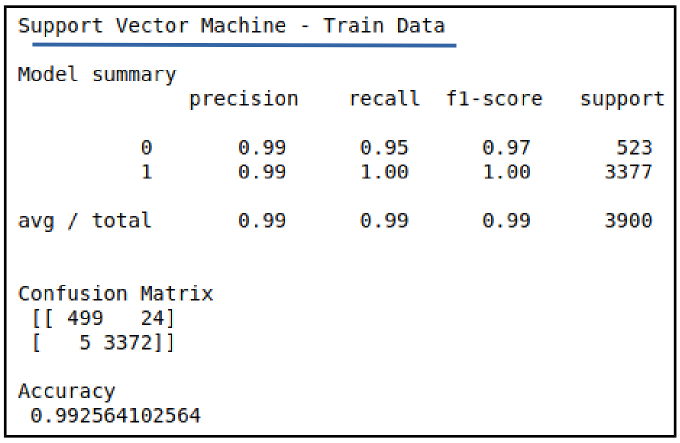
```
Decision Tree - Test data

Model summary
             precision    recall  f1-score   support

          0       0.92      0.82      0.87       224
          1       0.97      0.99      0.98      1448

avg / total       0.97      0.97      0.97      1672


Confusion Matrix
 [[ 183   41]
 [  15 1433]]

Accuracy
 0.966507177033
```



```
Support Vector Machine - Test Data

Model summary
             precision    recall  f1-score   support

          0       0.95      0.91      0.93       224
          1       0.99      0.99      0.99      1448

avg / total       0.98      0.98      0.98      1672


Confusion Matrix
 [[ 204   20]
 [  10 1438]]

Accuracy
 0.982057416268
```



```
Decision Tree - Train data

Model summary
             precision    recall  f1-score   support

          0       0.99      0.92      0.96       523
          1       0.99      1.00      0.99      3377

avg / total       0.99      0.99      0.99      3900


Confusion Matrix
 [[ 481   42]
 [   3 3374]]

Accuracy
 0.988461538462
```



```
Support Vector Machine - Train Data

Model summary
             precision    recall  f1-score   support

          0       0.99      0.95      0.97       523
          1       0.99      1.00      1.00      3377

avg / total       0.99      0.99      0.99      3900


Confusion Matrix
 [[ 499   24]
 [   5 3372]]

Accuracy
 0.992564102564
```

It can be seen that SVM performs better than Decision Tree on both test and the train data by comparing the Accuracies, also the false positives are considerably high for the Decision Tree results. It cannot be concluded that SVM is a supreme model, the decision tree is easy to interpret and easily explainable to business people. While there can be high chances of over-fitting, this can be avoided by pruning and using random forests(collection of trees built on various sub-samples of the data). Support Vector machine is a more sophisticated method, though choosing the Kernel can be tricky. Both methods cannot be placed one over other based on their performance on this dataset, the choice can be made based on the requirements.