@InProceedings{maas-EtAl:2011:ACL-HLT2011, author = {Maas, Andrew L. and Daly, Raymond E. and Pham, Peter T. and Huang, Dan and Ng, Andrew Y. and Potts, Christopher}, title = {Learning Word Vectors for Sentiment Analysis}, booktitle = {Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies}, month = {June}, year = {2011}, address = {Portland, Oregon, USA}, publisher = {Association for Computational Linguistics}, pages = {142--150}, url = {http://www.aclweb.org/anthology/P11-1015} (http://www.aclweb.org/anthology/P11-1015}) }

The sentiment analysis divided into four class for considering robustness of the program: 1.preprocessing_of_file 2.classification_model 3.prediction_savefile 4.mean_absolute_percentage

All the libary function used for this project

In [5]:
```
# notebook magic function to plot figures within the notebook

# import modules
import numpy as np
import pandas as pd
#TfidfVectorizer from sklean
from sklearn.feature_extraction.text import TfidfTransformer
# Load datasets in the svmlight / libsvm format into sparse CSR matrix
from sklearn.datasets import load_svmlight_file
from sklearn.svm import SVC
from time import time
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_score
print("Successfully imported all the libararies")
```

```
Successfully imported all the libararies
```

File Reading

In [6]:
```
#load train dataset and test data set
train_dataset, train_labels = load_svmlight_file('labeledBow_train.feat', 89527
)
test_dataset, test_labels = load_svmlight_file('labeledBow_test.feat', 89527)

print ("Loading Data successful")
```

```
Loading Data successful
```

Create class of Preprocessing of file

In [8]:
```
# Substitue is used as only two category either positive or negative
class preprocessing_of_file:
    # Binerize target data
    # Converting target into binary like if review rating >5, positive (1) if <
=5, negative (-1)
    def binerize (self,target):
        binary = []
        for i in range(len(target)):
            if target[i] > 5:
                binary.append(1) # Positive
            else:
                binary.append(-1) # Negative
        return binary
    # Calculating Tf-Idf for training and testing
    def tf_idf(self, training, testing): #taking trainig and testing arguments
        tf_transformer = TfidfTransformer() #Transform a count matrix to a norm
alized tf or tf-idf representation
        print("Training_data TF-IDF")
        #  It computes the TF for each review, the IDF using each review, and f
inally the TF-IDF for each review
        training_tfidf = tf_transformer.fit_transform(training)
        print(training_tfidf.shape)
        print("Testing_data TF-IDF")
        # .transform on the testing data which computes the TF for each review,
        # then the TF-IDF for each review using the IDF from the training data
        testing_tfidf = tf_transformer.transform(testing)
        print(testing_tfidf.shape)
        return [training_tfidf,testing_tfidf]
```

Run preprocessing_of_file class

In [9]:
```
# Object for preprocessing_of_file
pf=preprocessing_of_file()
print("Binerizing target ...")
train_label = pf.binerize(train_labels)
test_label = pf.binerize(test_labels)
print("Binerizing target data successfull")
print("\n")
print("Calculating Tf-Idf for training and testing")
tfidf_data = pf.tf_idf(train_dataset, test_dataset)
training_data = tfidf_data[0]
testing_data = tfidf_data[1]
```

```
Binerizing target ...
Binerizing target data successfull


Calculating Tf-Idf for training and testing
Training_data TF-IDF
(25000, 89527)
Testing_data TF-IDF
(25000, 89527)
```

Classification Model Use for This project

In [2]:
```
class classification_model:

    #Linear SVM classification Model function without cross validation

    def lSVM(self,training_data, training_target, testing_data, testing_target)
:
        start = time()
        # create a Linear SVC object
        clf_linear = LinearSVC()
        print("Data are Started Training")
        # teach the linear SVC using the training dataset
        clf_linear.fit(training_data, train_label)
        print("Training Successful")
        print("Data are Started Testing")
        # test the linear SVC accuracy using the testing dataset
        clf_linear_accuracy = clf_linear.score(testing_data, test_label)*100
        end = time()
        return [clf_linear, round(clf_linear_accuracy,2), float(round(end-start
))]

    #Linear SVM classification Model function with cross validation or c value
adjustment
    def lSVC_para(self, training_data, training_target, testing_data, testing_t
arget):
        print("Calculating best parameter for LinearSVC Classifier ...")
        # array of C values to test
        # Due to computation issue only taking value -2 to 4
        clist = 2**np.array(range(-2, 4), dtype='float') # 4**-4, 4**-1, 2**0,
4**1, 4**2, ...
        cvscores = [] #For High-Dimensional Data With Known Groups, Derive Scor
es For Plotting
        iterator=range(8)
        for c,r in zip(clist,iterator):
            print(c)
            clf= LinearSVC(C=c)
            scores = cross_val_score(clf, training_data, training_target, cv=3)
#Evaluate a score by cross-validation
            print('Iteration #{}'.format(r+1))
            print("score", scores)
            cvscores.append(scores.mean()*100)
            bestscore, bestC = max([(val, clist[idx]) for (idx, val) in enumera
te(cvscores)])
        print('Best CV accuracy =', round(bestscore,2), '% achieved at C =', be
stC)

        # Retrain on whole trainning set using best C value obtained from Cross
validation
        print("Retrain on whole trainning set using best C value obtained from
Cross validation")
        clf = LinearSVC(C=bestC)
        clf.fit(training_data, training_target)
        accu = clf.score(testing_data, testing_target)*100
        return (clf, accu, bestC)

    #Random Forest classification Model function without adjust depth and numbe
r of tress
    def random_forest(self, training_data, training_target, testing_data, testi
ng_target):
        start = time()
        # create a andom Forest object
        clf_forest = RandomForestClassifier()
        print("Data are Started Training")
        # teach the andom Forest  using the training dataset
        obj_random_forest=clf_forest.fit(training_data, train_label)
        print("Training Successful")
        print("Data are Started Testing")
        # test the andom Forest  accuracy using the testing dataset
        clf_forest_accuracy = clf_forest.score(testing_data, test_label)*100
        end = time()
        return [clf_forest, round(clf_forest_accuracy,2), float(round(end-start
```

Run Classification Model without parameter adjustment

In [44]:
```
cm = classification_model() # Object for classification_model
print("Linear SVM Classifier ")
output = cm.lSVM(training_data, train_label, testing_data, test_label)
obj_lSVM = output[0]
print("Accuracy = ", output[1], "% Time = ", output[2],"seconds")
print("\n")
print("Random Forest Classifier without adjust depth and number of tress adjust
ment")
output = cm.random_forest(training_data, train_label, testing_data, test_label)
obj_random_forest = output[0]
print("Accuracy = ", output[1], "% Time = ", output[2],"seconds")
```

```
Linear SVM Classifier
Data are Started Training
Training Successful
Data are Started Testing
Accuracy =  87.9 % Time =  1.0 seconds


Random Forest Classifier without adjust depth and number of tress adjustment
Data are Started Training
Training Successful
Data are Started Testing
Accuracy =  73.2 % Time =  9.0 seconds
```

Run Classification Model with parameter adjustment

In [47]:
```
print("Linear SVM Classifier With Parameter Selection")
start = time()
output = cm.lSVC_para(training_data, train_label, testing_data, test_label)
end = time()
obj_lSVM_para = output[0]
print("Accuracy = ", output[1], "% at Best C = ", output[2], "% Time = ", float
(round(end-start)),"seconds")
print("\n")
print("Random Forest Classifier with depth and number of tress adjustment")
print("Calculating best parameter for random  Classifier by changing tree depth
and leaf")
output = cm.random_forest_para(training_data, train_label, testing_data, test_l
abel)
obj_random_forest_para = output[0]
print("\n")
print("Best Accuracy = ", output[1])
```

```
Linear SVM Classifier With Parameter Selection
Calculating best parameter for LinearSVC Classifier ...
0.25
Iteration #1
score [0.85805136 0.86177106 0.87013922]
0.5
Iteration #2
score [0.85325174 0.85913127 0.8643783 ]
1.0
Iteration #3
score [0.84485241 0.85157187 0.85417667]
2.0
Iteration #4
score [0.83717303 0.84233261 0.84349496]
4.0
Iteration #5
score [0.82889369 0.83273338 0.83713394]
8.0
Iteration #6
score [0.82277418 0.82793377 0.83269323]
Best CV accuracy = 86.33 % achieved at C = 0.25
Retrain on whole trainning set using best C value obtained from Cross validati
on
Accuracy =  88.628 % at Best C =  0.25 % Time =  16.0 seconds


Random Forest Classifier with depth and number of tress adjustment
Calculating best parameter for random  Classifier by changing tree depth and l
eaf
Accuracy =  82.74 at Leaf Size =  5 Tree Depth 16 % Time =  6.0 seconds
Accuracy =  83.09 at Leaf Size =  10 Tree Depth 32 % Time =  10.0 seconds
Accuracy =  82.62 at Leaf Size =  25 Tree Depth 64 % Time =  7.0 seconds


Best Accuracy =  83.09
```

Create class for prediction and save the file

In [48]:
```
class prediction_savefile:
    # write Prediction function
    def prediction(self, obj_clf):
        pre = obj_clf.predict(testing_data)
        print("Done")
        prediction_result = []
        for i in range(len(pre)):
            if pre[i] == 1:
                prediction_result.append(str(i) + ", positive") #label positve
            else:
                prediction_result.append(str(i) + ", negative")
        return(pre, prediction_result)
    # Storing prediction in CSV file
    def save_csv(self, prediction_result, fileName, labels):
        print("Creating CSV file")
        # Open File
        output_file = open(fileName+".csv",'w')
        output_file.write(','.join(labels)+"\n")
        # Write data to file
        for r in prediction_result:
            output_file.write(r + "\n")
        output_file.close()
        print("File saved!")
```

Run prediction and save the file class

In [49]:
```
ps=prediction_savefile() # Object for prediction_savefile
print("Prediction for new dataset from classifier...")
print("\n")
print ("Using Linear SVM Prediction model")
pre_lsvm=ps.prediction(obj_lSVM_para)
print("Save Prediction result to CSV file")
labels = ["review","rating"]
ps.save_csv(pre_lsvm[1], "lsvm", labels)
print("\n")
print ("Using Random Forest Classification model")
pre_random=ps.prediction(obj_random_forest_para)
print("Save Prediction result to CSV file")
labels = ["review","rating"]
ps.save_csv(pre_random[1], "Random", labels)
```

```
Prediction for new dataset from classifier...


Using Linear SVM Prediction model
Done
Save Prediction result to CSV file
Creating CSV file
File saved!


Using Random Forest Classification model
Done
Save Prediction result to CSV file
Creating CSV file
File saved!
```

Create Mean Absolute Percentage Error Class

In [1]:
```
#"The mean absolute percentage error (MAPE) is a measure of prediction accuracy
of a forecasting method in statistics."
class mean_absolute_percentage:
    def mean_absolute_percentage_error(self, y_true, y_pred):
        return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
#(Cross Validated 2018)
```

In [51]:
```
#Calculate the test error for each classification model and display them on scr
een
#The MAPE (Mean Absolute Percent Error) measures
#the size of the error in percentage terms.
print ("Calculate mean absulte percentage error")
mp=mean_absolute_percentage()
print("Mean absulte percentage error of Linear SVM")
print(mp.mean_absolute_percentage_error(test_label, pre_lsvm[0]))
print("Mean absulte percentage error of Random Forest")
print(mp.mean_absolute_percentage_error(test_label, pre_random[0]))
```

```
Calculate mean absulte percentage error
Mean absulte percentage error of Linear SVM
22.744
Mean absulte percentage error of Random Forest
34.768
```

Table 1: Result of Sentiment Analysis

Model	Accuracy	MAPE	Time
Linear SVM	88.628	22.74	16 Second
Random Forest Classifier	83.09	34.768	11 Second

NB: MAPE (mean_absolute_percentage error)

From Table 1, it is noticiable that Linear SVM gives better accuracy (88.628 %) with less MAPE only (22.74 %). It is obious that random forest works faster than SVM. Althogh some model could not able to use here as required large computation power for example kernal SVM more than 10 minitues or more, more model function can be added in classification class and tested. However, Linear SVM is better than random forest model as data is categorized only two target features Positive and Negative. Therefore, it is the best model for sentiment analysis (Dr Ivan Bojicic 2018; Anon 2018).

Reference: Dr Ivan Bojicic [University of Western Sydney] 2018, 301046 lecture Lecture 10 - More Predictive Modelling, 21 May). Anon, (2018). [online] Available at: https://www.researchgate.net /post/Is_random_forest_better_than_support_vector_machines (https://www.researchgate.net /post/Is_random_forest_better_than_support_vector_machines) [Accessed 3 Jun. 2018]. Cross Validated, 2018. Mean absolute percentage error (MAPE) in Scikit-learn. [online] Cross Validated. Available at: https://stats.stackexchange.com /questions/58391/mean-absolute-percentage-error-mape-in-scikit-learn (https://stats.stackexchange.com/questions /58391/mean-absolute-percentage-error-mape-in-scikit-learn) [Accessed 3 Jun. 2018].