CSCI 630 : Foundations of Intelligent Systems

Project 2 – Exploring discourse in Reddit

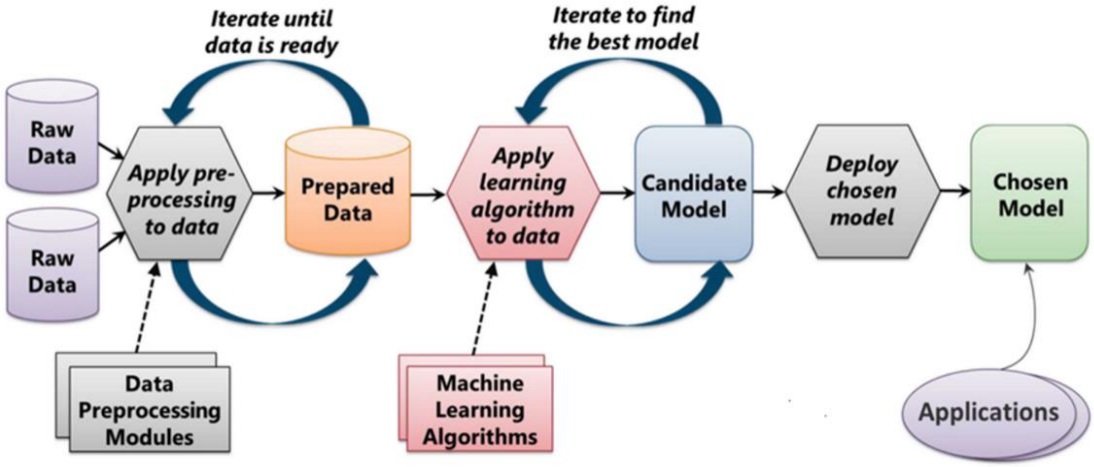
Author : Omkar Vaidya (ov5232@rit.edu)

INDEX

INTRODUCTION

As new technology comes into picture, ways of generating data are increasing exponentially. This is when data mining and machine learning techniques comes into picture. The main objective of this project is to build a machine learning agent that can predict from which subreddit an unlabelled post comes from. Subreddits chosen for the task are Computer Science (r/computerscience) , Data Science (r/datascience) and GRE (r/GRE). Latest 500 posts of these three subreddits are chosen for the task.

As it can be observed from the standard diagram of a machine learning task given below, the project proceeds through several steps, such as data pre-processing , feature engineering, developing and applying classifiers such as Random Forest Classifier and Support Vector Machine. The project deals with basic implementation theory of given two classifiers for two or more subreddits which have balanced and almost equal number of posts. The data can be visualized by implementing Word Clouds for each of the data set taken into consideration for classification.



One of the most crucial tasks after implementing classifier is to check it's performance
parameters and compare them in order to get better future results for classification. This project deals with performance measurement parameters like Accuracy Score, Confusion Matrix, Classification Report (Precision and Recall), Receiver Operating Characteristic Curve(ROC) etc.

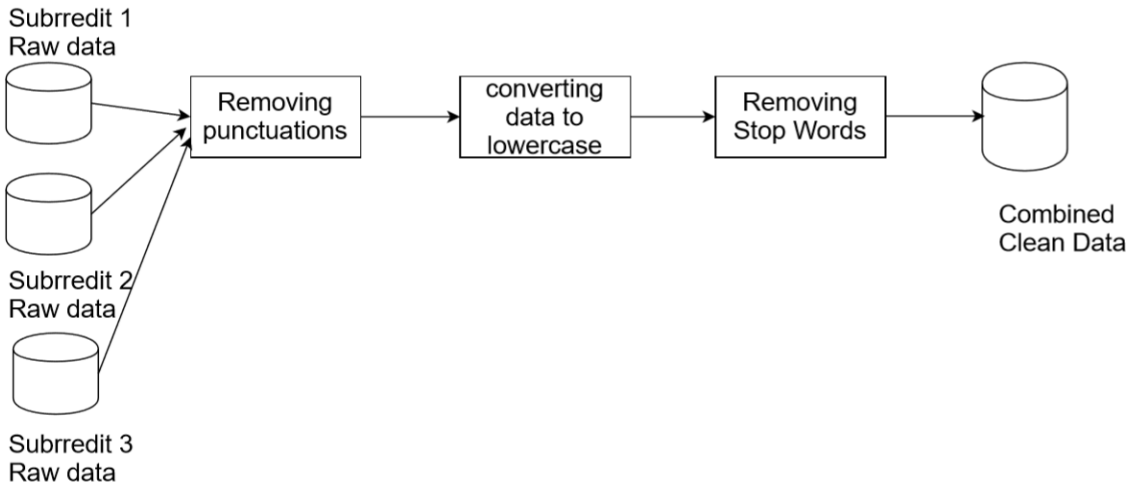
METHODS & RESULTS

Data collection and splitting :

```
# Connecting to the reddit script app through PRAW API
reddit = praw.Reddit(client_id='Nuhb2V_ONfbRAg',\
                     client_secret='ao8oVLq-ZEz1a-9leqg1FqGdCIg',\
                     user_agent='redditApp1',\
                     username='ketand2017',\
                     password='passwordfis'
                     )

# The subreddits from which data is collected
subreddit1 = reddit.subreddit('computerscience')
subreddit2 = reddit.subreddit('datascience')
subreddit3 = reddit.subreddit('GRE')

# Collecting the latest 500 posts from each subreddit
new_subreddit1 =subreddit1.new(limit=500)
new_subreddit2 =subreddit2.new(limit=500)
new_subreddit3 =subreddit3.new(limit=500)

# Storing the subreddits data into json format
first_subreddit_data = {"body":[],"subreddit":[]}
second_subreddit_data = {"body":[],"subreddit":[]}
third_subreddit_data = {"body":[],"subreddit":[]}

# Appending the posts and label for first subreddit data
for post in new_subreddit1:
    first_subreddit_data["body"].append(post.selftext)
    first_subreddit_data["subreddit"].append(post.subreddit.display_name)

# Appending the posts and label for second subreddit data
for post in new_subreddit2:
    second_subreddit_data["body"].append(post.selftext)
    second_subreddit_data["subreddit"].append(post.subreddit.display_name)

# Appending the posts and label for third subreddit data
for post in new_subreddit3:
    third_subreddit_data["body"].append(post.selftext)
    third_subreddit_data["subreddit"].append(post.subreddit.display_name)


# Appending the posts and label for third subreddit data
for post in new_subreddit3:
    third_subreddit_data["body"].append(post.selftext)
    third_subreddit_data["subreddit"].append(post.subreddit.display_name)

# Converting the collected data into dataframe
data1 = pd.DataFrame(first_subreddit_data)
data2 = pd.DataFrame(second_subreddit_data)
data3 = pd.DataFrame(third_subreddit_data)

# Storing the data for each subreddit into json file
data1.to_json("subredditdata1.json",orient='split')
data2.to_json("subredditdata2.json",orient='split')
data3.to_json("subredditdata3.json",orient='split')

# Merging all the data
data = data1.append(data2)
data = data.append(data3)

# Storing the data into json file
data.to_json("subredditdata.json",orient='split')
data['subreddit'].replace({'computerscience': 0, 'datascience': 1,'GRE':2}, inplace=True)

#Shuffling the data
data = data.sample(frac=1)

#Splitting the data into 50, 25, 25
print('Splitting the data in training 50%, development 25%, testing 25%')
training_data = data.iloc[:(len(data)//2)-1,:]
development_data = data.iloc[(len(data)//2)-1:int(0.75*len(data)),:]
testing_data = data.iloc[int(0.75*len(data)):,:]
```

Data Pre-processing - Basic

- In order to perform any machine learning task, the data must be in clean format.
- Defining cleanliness of data: Words that do not carry any significance importance in the text such as stop words (articles in English Language), removing punctuations and whitespaces.
- In order to process data in efficient manner, it is advisable that it must be uniform. Hence all the data is converted to lowercase.
- All the data is combined and mixed randomly at the end of pre-processing as, in order to perform training, development and testing tasks efficiently.



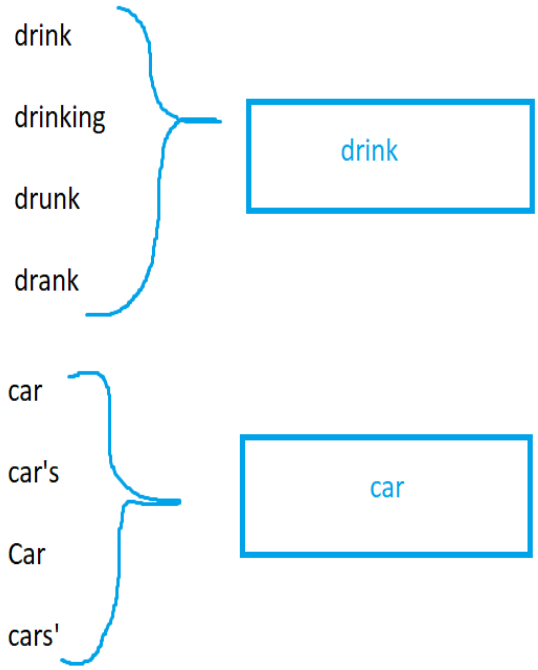
Code :

```
# Transforming to lower case
corpus['body'] = corpus['body'].apply(lambda x: " ".join(x.lower() for x in x.split()))
# Removing Punctuations
corpus['body'] = corpus['body'].str.replace('[^\w\s]','')

# Removing Stop words
corpus = {'body':[]}
for word in development_data['body']:
    tokenized_subreddit = word_tokenize(word)
    for each in tokenized_subreddit:
        if each in stopwords.words('english'):
            tokenized_subreddit.remove(each)
    tokenized_subreddit = ' '.join(tokenized_subreddit)
    corpus['body'].append(tokenized_subreddit)
```

Data Pre-processing - Advanced

Background :

- Language in day to day use is generally consists of words in their derived format.
- When language consists of words that are derived from each other it is known as “inflected language”. Inflection consists of one or more grammatical categories with a prefix, suffix, or infix or another internal modification.
- The degree of inflection can vary language to language. Inflected words will have common roots. Let us consider an example,

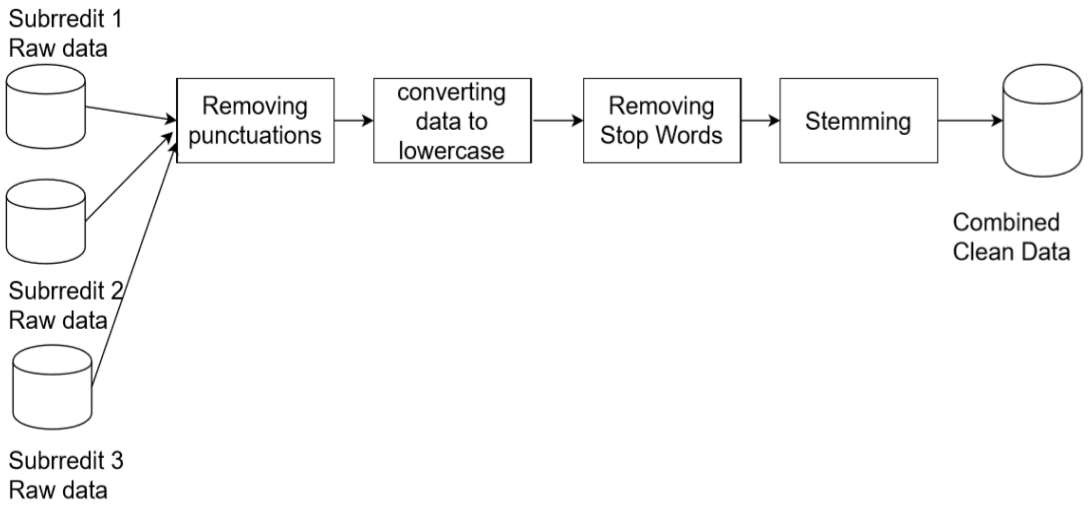


It can be observed from given example that drink, drinking, drank, drunk are the forms of word drink.

Hence stemming and lemming is performed to clean the data and to get the exact context of data by normalizing it.

a. Data Pre-processing -Basic : Combined with Stemming

- Stemming: Stemming is a Text Normalization technique under the field of Natural Language Processing.
- It is a process of reducing inflection in the words to their root forms such as mapping a group of words to the same stem. It can be a possibility that stem may not be an actual word in the language.
- Parts such as (-ed , -ize , -s , -de , -ing , -tion) are removed in the process.
- We used nltk.stemmer to find out different stemmers available in English Language.
- Porterstemmer and LancesterStemmer are popular stemmers used for working on stems of English Language.
- We have used PorterStemmer in the project.

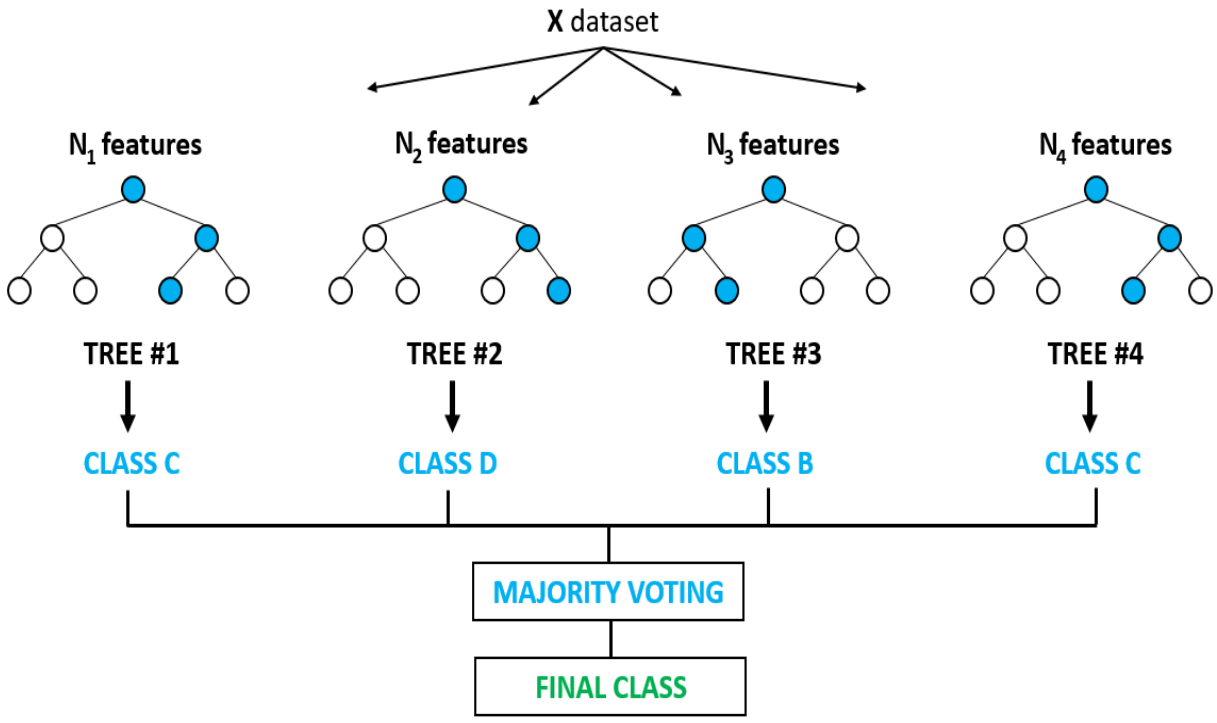


Code :

```
# Stemming
stemmer = PorterStemmer()
for i in range(len(tokenized_subreddit)):
    tokenized_subreddit[i] = stemmer.stem(tokenized_subreddit[i])
tokenized_subreddit = ' '.join(tokenized_subreddit)
corpus['body'].append(tokenized_subreddit)
```

I. Random Forest Classifier :

- Random Forest is a machine learning algorithm which can be used for both classification and regression tasks. Random forest is an ensemble algorithm which means, it combines more than one algorithm of same of different type for classification of objects.
- This may include considering predictions from Naïve Bayes, SVM, Decision tree and taking vote for final consideration of class of test object.
- Random Forest classifier generates a set of decision trees from randomly selected subset of training set. It aggregates votes from every decision tree in order to predict the final class of the object.
- This technique is effective because a single decision tree can be prone to noise but aggregate of number of decision trees marginally reduce the effect of noise by giving more accurate result.
- Implementation of voting in Random forest algorithm:
- By considering the impact of decision tree it's contribution to final voting can be manipulated. Error rate of a tree is inversely proportional to weight value of the tree.
- This results in increase in decision impact of tree with lower error rate.



Implementation of Random Forest Classifier :

From sklearn.ensemble library of python we used RandomForestClassifier class. The code is as follows :

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(max_features=1000, lowercase=True, analyzer='word',stop_words= 'english',ngram_range=(1,1))
tfidf.fit(x)
traindata_tfid = tfidf.transform(x)
testdata_tfid = tfidf.transform(testing_data['body'])


#Random Forest
from sklearn.ensemble import RandomForestClassifier

RF_trained_model = RandomForestClassifier(n_estimators=100)
RF_trained_model.fit(traindata_tfid,y)
print(RF_trained_model)

RF_predictions = RF_trained_model.predict(testdata_tfid)

from sklearn import metrics

print("Random Forest Accuracy:",metrics.accuracy_score(testing_data['subreddit'], RF_predictions))
print("Random Forest Confusion Matrix:",metrics.confusion_matrix(testing_data['subreddit'], RF_predictions))
print("Random Forest Classification Report:",metrics.classification_report(testing_data['subreddit'], RF_predictions))
```

Results :

Training -

```
Splitting the data in training 50%, development 25%, testing 25%
Implementing Random Forest using training and testing data
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
Random Forest Accuracy: 0.7733333333333333
Random Forest Confusion Matrix: [[ 84  14  22]
 [ 13  95  21]
 [ 15   0 111]]
Random Forest Classification Report:              precision    recall  f1-score   support

           0       0.75      0.70      0.72       120
           1       0.87      0.74      0.80       129
           2       0.72      0.88      0.79       126

   micro avg       0.77      0.77      0.77       375
   macro avg       0.78      0.77      0.77       375
weighted avg       0.78      0.77      0.77       375
```

Development -

```
Performing Feature Extraction on Development Data:
Implementing Random Forest using development and testing data
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
Random Forest Accuracy: 0.7653333333333333
Random Forest Confusion Matrix: [[ 82  16  22]
 [ 15  96  18]
 [ 12   5 109]]
Random Forest Classification Report:               precision    recall  f1-score   support

           0       0.75      0.68      0.72       120
           1       0.82      0.74      0.78       129
           2       0.73      0.87      0.79       126

   micro avg       0.77      0.77      0.77       375
   macro avg       0.77      0.76      0.76       375
weighted avg       0.77      0.77      0.76       375
```

Testing -

```
Performing Feature Extraction on training data set and then testing:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
Random Forest Accuracy: 0.7813333333333333
Random Forest Confusion Matrix: [[ 85  16  19]
 [ 14  96  19]
 [ 13   1 112]]
Random Forest Classification Report:               precision    recall  f1-score   support

           0       0.76      0.71      0.73       120
           1       0.85      0.74      0.79       129
           2       0.75      0.89      0.81       126

   micro avg       0.78      0.78      0.78       375
   macro avg       0.79      0.78      0.78       375
weighted avg       0.79      0.78      0.78       375
```

II. Support Vector Machine (SVM) :

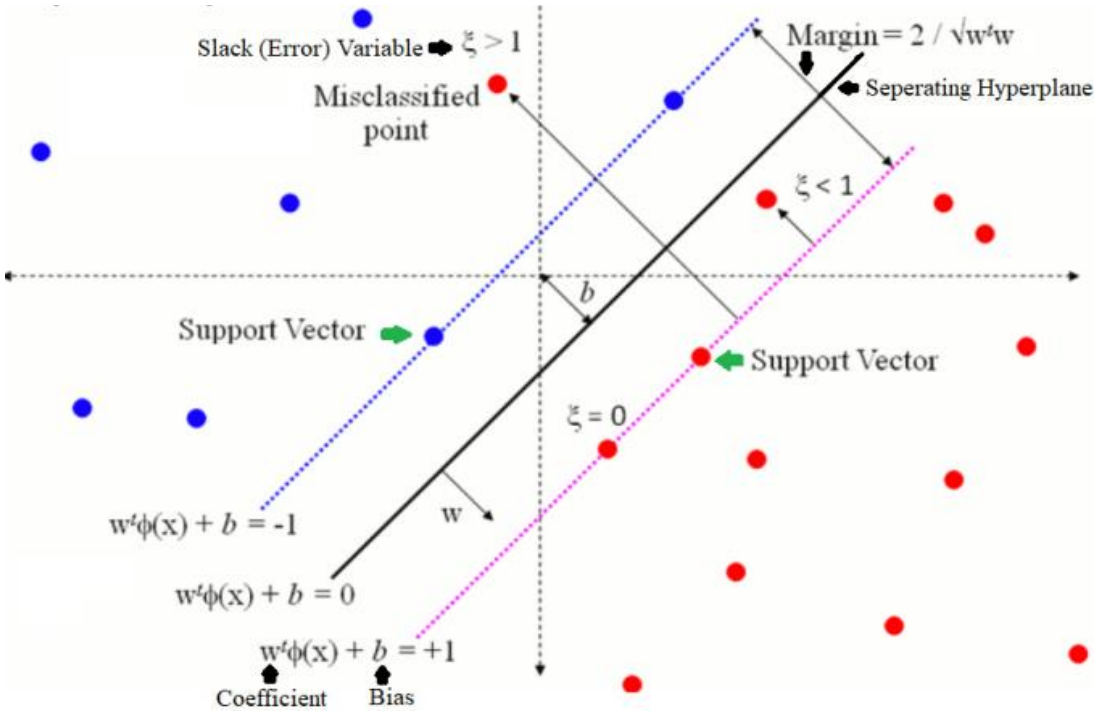
- Support vector machine is a is a discriminative classifier. Key feature in SVM is it's separating hyperplane. It is a supervised learning algorithm meaning that given a labelled training data the algorithm returns optimal hyperplane which classifies testing data samples in respective classes.
- In two dimensional co-ordinate system, SVM can this hyperplane is a line. Line partitions data points of different subreddits.

Tuning parameters for SVM :

- Regularization parameter: This is also known as C parameter in sklearn library. This is the estimation of the measure of misclassification of each training example. For large c value, the optimization will choose smaller margin hyperplane.
- Gamma: This is the estimation of how far the influence of single parameter reaches. Higher value of Gamma means close i.e. high gamma means the points close to plausible line are considered in calculation. Lower values means points away from plausible separation line.
- Kernel: For new input (x) , for each support vector(xi) is calculated like:

Linear kernel: $F(x) = B(0) + \text{sum}(a_i * (x, x_i))$

Polynomial kernel: $K(x, x_i) = \exp(-\text{gamma} * \text{sum}((x - x_i^2)))$



Implementation of Support Vector Machine (SVM) :

From sklearn library of python we used SVC class. The code is as follows :

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(max_features=1000, lowercase=True, analyzer='word',stop_words= 'english',ngram_range=(1,1))
tfidf.fit(x)
traindata_tfid = tfidf.transform(x)
testdata_tfid = tfidf.transform(testing_data['body'])


#SVM
from sklearn import svm

SVM_trained_model = svm.SVC(kernel='linear')
SVM_trained_model.fit(traindata_tfid,y)
print(SVM_trained_model)

SVM_predictions = SVM_trained_model.predict(testdata_tfid)

from sklearn import metrics

print("SVM Accuracy:",metrics.accuracy_score(testing_data['subreddit'], SVM_predictions))
print("SVM Confusion Matrix:",metrics.confusion_matrix(testing_data['subreddit'], SVM_predictions))
print("SVM Classification Report:",metrics.classification_report(testing_data['subreddit'], SVM_predictions))
```

Training -

```
Splitting the data in training 50%, development 25%, testing 25%
Implementing SVM using training and testing data
SVC(C=1.0, 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=None,
  shrinking=True, tol=0.001, verbose=False)
SVM Accuracy: 0.7893333333333333
SVM Confusion Matrix: [[ 87   6  23]
 [ 20  81  18]
 [  7   5 128]]
SVM Classification Report:              precision    recall  f1-score   support

           0       0.76      0.75      0.76       116
           1       0.88      0.68      0.77       119
           2       0.76      0.91      0.83       140

   micro avg       0.79      0.79      0.79       375
   macro avg       0.80      0.78      0.78       375
weighted avg       0.80      0.79      0.79       375
```

Development -

```
Performing Feature Extraction on Development Data:
Implementing SVM using development and testing data
SVC(C=1.0, 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=None,
  shrinking=True, tol=0.001, verbose=False)
SVM Accuracy: 0.792
SVM Confusion Matrix: [[ 82  12  22]
 [ 13  88  18]
 [  9   4 127]]
SVM Classification Report:              precision    recall  f1-score   support

           0       0.79      0.71      0.75       116
           1       0.85      0.74      0.79       119
           2       0.76      0.91      0.83       140

   micro avg       0.79      0.79      0.79       375
   macro avg       0.80      0.78      0.79       375
weighted avg       0.80      0.79      0.79       375
```

Testing -

```
Performing Feature Extraction on training data set and then testing:
SVC(C=1.0, 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=None,
  shrinking=True, tol=0.001, verbose=False)
SVM Accuracy: 0.8133333333333334
SVM Confusion Matrix: [[ 85   7  24]
 [ 17  85  17]
 [  3   2 135]]
SVM Classification Report:              precision    recall  f1-score   support

           0       0.81      0.73      0.77       116
           1       0.90      0.71      0.80       119
           2       0.77      0.96      0.85       140

   micro avg       0.81      0.81      0.81       375
   macro avg       0.83      0.80      0.81       375
weighted avg       0.82      0.81      0.81       375
```

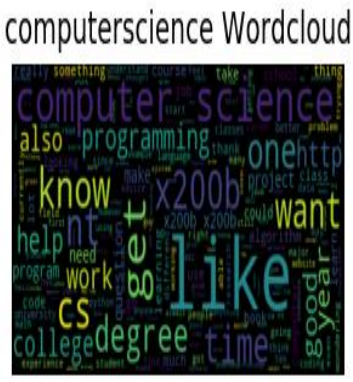
DATA VISUALIZATION

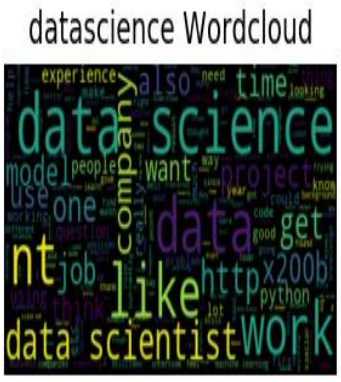
I. Word Cloud

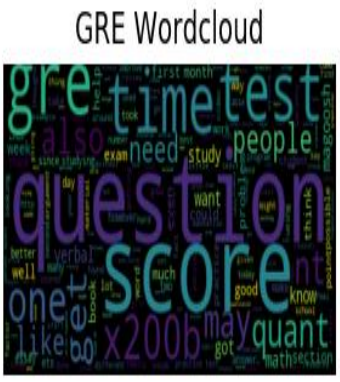
- When raw data is text based, the best visualization format to highlight important textual data points is a word cloud. Word clouds are used frequently. It is the best way to immediately convey crucial information. It underscores the keywords that we should focus on.

Working of word cloud:

- Word cloud has simple underlying working technique which is more a specific word appears in raw data or the data under consideration, bigger and bolder the word appears in the word cloud.







PERFORMANCE PARAMETERS

I. Receiver Operator Curve (ROC) :

When performance measurement is required for any machine learning task AUC (Area under the Curve) – Receiver Operating Characteristics are one of the most efficient method.

- ROC measures performance at various threshold settings.
- ROC: Probability Curve
- AUC: Degree of measure of separability

It can be determined from ROC curve that how much model is capable of distinguishing between classes.

In order to have plot ROC Curve it is necessary to have understanding of confusion matrix.

Confusion Matrix is another method of performance measurement machine learning classification problem. It can be represented as:

		Predicted class	
		P	*N*
Actual Class	*P*	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

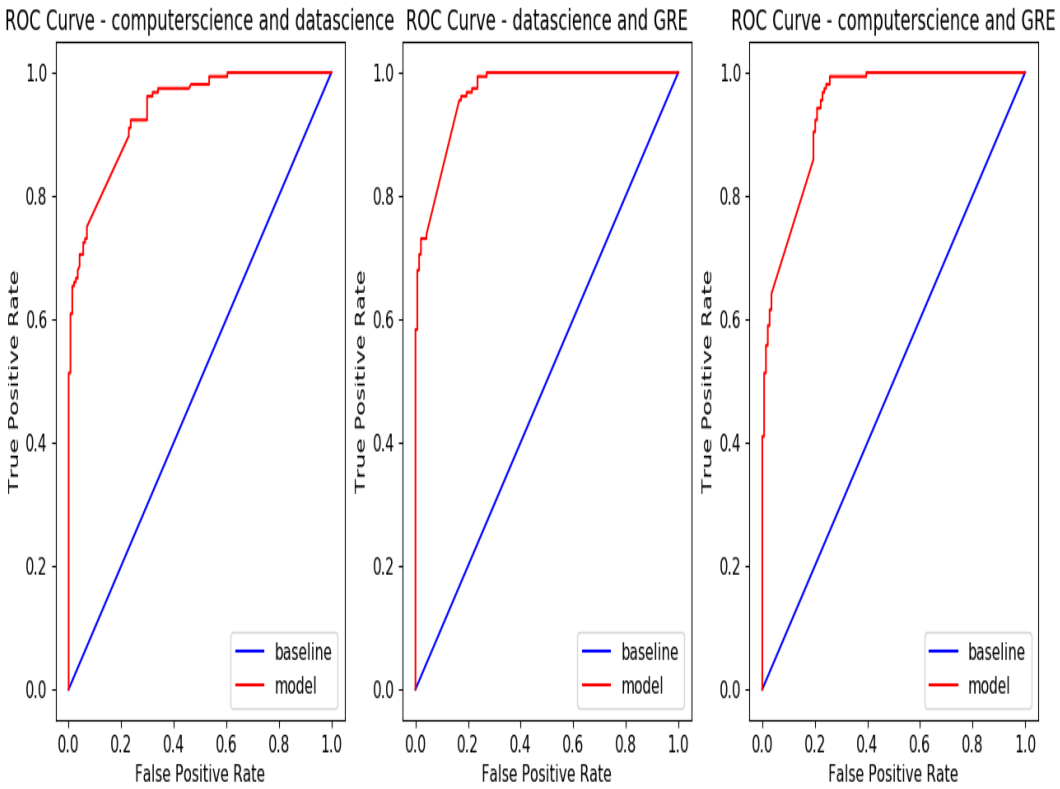
ROC is a plot of FPR against TPR where,

TPR (True Positive Rate) $= \frac{TP}{(TP+FN)}$

FPR (False Positive Rate) $= \frac{FP}{(TN+FP)}$

Performance measurement: Best AUC, near to 1

The ROC curve for the given subreddits are as follows :



II. Precision and Recall:

- Precision is a measure of what proportion of positive identification is actually correct.

  Precision = $\frac{TP}{(TP+FP)}$

  TP: True Positive

  FP: False Positive

- Recall : Recall measures what proportion of actual positives are identified correctly.

  Recall = $\frac{TP}{(TP+FN)}$

SVM Classification Report:	precision	recall	f1-score	support
0	0.81	0.73	0.77	116
1	0.90	0.71	0.80	119
2	0.77	0.96	0.85	140

III. Classification Report:

Consists of precision , recall , f1 score , support.

SVM Classification Report:	precision	recall	f1-score	support
0	0.81	0.73	0.77	116
1	0.90	0.71	0.80	119
2	0.77	0.96	0.85	140
micro avg	0.81	0.81	0.81	375
macro avg	0.83	0.80	0.81	375
weighted avg	0.82	0.81	0.81	375

CONCLUSION

The goal was to classify subreddits based on their body using classifiers such as Support Vector Machine (SVM) and Random Forest Classifier along with tasks such as stemming and lemming over three subreddits for more than 500 subreddit posts. The task was performed successfully as it can be observed by various performance parameters like ROC curve , Classification Report, Confusion Matrix and Accuracy Score.