In [10]:
```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import model_selection, naive_bayes
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, f1_score
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MaxAbsScaler
from sklearn.metrics import roc_auc_score
import os.path
import pickle
```

Reading test and train data from already preprocessed pickle file

In [11]:
```
X_train = pd.read_pickle('../../../Preprocessing/Data/X_train.pkl')
X_test = pd.read_pickle('../../../Preprocessing/Data/X_test.pkl')
y_train = pd.read_pickle('../../../Preprocessing/Data/y_train.pkl')
y_test = pd.read_pickle('../../../Preprocessing/Data/y_test.pkl')
```

Before we do anything, we need to get the vectors. We can download one of the pre-trained models. We downloaded the pretrained model from http://nlp.stanford.edu/data/glove.6B.zip (http://nlp.stanford.edu/data/glove.6B.zip)

In [12]:
```
import numpy as np
w2v = {}

f = open("../Word2Vec_Data/glove.6B.50d.txt", "rb")

for line in f:
    w2v[line.split()[0]] = np.array(line.split()[1:]).astype(np.float)
```

In [13]:
```
words_not_found = 0
train_doc_vectors = pd.DataFrame() # creating empty final dataframe
if os.path.isfile('../Word2Vec_Data/train_doc_vectors.pkl'):
    train_doc_vectors = pd.read_pickle('../Word2Vec_Data/train_doc_vecto
rs.pkl')
else:
    for doc in tqdm(X_train.values): # looping through each document and
cleaning it
        temp = pd.DataFrame()  # creating a temporary dataframe(store va
lue for 1st doc & for 2nd doc remove the details of 1st & proced through
2nd and so on..)
        word_vec = np.zeros(50)
        temp = temp.append(pd.Series(word_vec), ignore_index = True) # i
f word is present then append it to temporary dataframe

        for word in doc.split(" "): # looping through each word of a sin
gle document and spliting through space
            word = word.encode("utf-8")
            try:
                word_vec = w2v[word] # if word is present in embeddings
(goole provides weights associate with words(300)) then proceed
                temp = temp.append(pd.Series(word_vec), ignore_index = T
rue) # if word is present then append it to temporary dataframe
            except:
                word_vec = np.zeros(50)
                words_not_found += 1
                temp = temp.append(pd.Series(word_vec), ignore_index = T
rue) # if word is present then append it to temporary dataframe
                pass
        doc_vector = temp.mean() # take the average of each column(w0, w
1, w2,.....w300)
        train_doc_vectors = train_doc_vectors.append(doc_vector, ignore_
index = True) # append each document value to the final dataframe
    train_doc_vectors.to_pickle("../Word2Vec_Data/train_doc_vectors.pkl"
)

print(train_doc_vectors.shape)
```

```
(39912, 50)
```

In [14]:
```
words_not_found_test = 0
test_doc_vectors = pd.DataFrame() # creating empty final dataframe
if os.path.isfile('../Word2Vec_Data/test_doc_vectors.pkl'):
    test_doc_vectors = pd.read_pickle('../Word2Vec_Data/test_doc_vector
s.pkl')
else:
    for doc in tqdm(X_test.values): # looping through each document and
 cleaning it
        temp = pd.DataFrame()  # creating a temporary dataframe(store va
lue for 1st doc & for 2nd doc remove the details of 1st & proced through
2nd and so on..)
        word_vec = np.zeros(50)
        temp = temp.append(pd.Series(word_vec), ignore_index = True) # i
f word is present then append it to temporary dataframe

        for word in doc.split(" "): # looping through each word of a sin
gle document and spliting through space
            word = word.encode("utf-8")
            try:
                word_vec = w2v[word] # if word is present in embeddings
(goole provides weights associate with words(300)) then proceed
                temp = temp.append(pd.Series(word_vec), ignore_index = T
rue) # if word is present then append it to temporary dataframe
            except:
                word_vec = np.zeros(50)
                words_not_found_test += 1
                temp = temp.append(pd.Series(word_vec), ignore_index = T
rue) # if word is present then append it to temporary dataframe
                pass
        doc_vector = temp.mean() # take the average of each column(w0, w
1, w2,.....w300)
        test_doc_vectors = test_doc_vectors.append(doc_vector, ignore_in
dex = True) # append each document value to the final dataframe
    test_doc_vectors.to_pickle("../Word2Vec_Data/test_doc_vectors.pkl")

print(test_doc_vectors.shape)
```

```
(19659, 50)
```

Scaling the input data using MaxAbsScaler

In [15]:
```
train_doc_vectors.fillna(0)
test_doc_vectors.fillna(0)
scaler = MaxAbsScaler()
# using averaged word embeddings
train_term_doc = scaler.fit_transform(train_doc_vectors)
test_term_doc = scaler.fit_transform(test_doc_vectors)
```

Multilabel Classification using Binary Relevance

Following function performs Multinomial Naive Bayes for each label. In short, it uses Binary Relevance (BR) method for multi-label classification.

In [16]:
```
def perform_NB_for_label(label):
    naive_classifier = naive_bayes.MultinomialNB()
    naive_classifier.fit(train_term_doc, y_train[label])
    predictions_NB = naive_classifier.predict(test_term_doc)
    print(label + " Accuracy Score - " + str(accuracy_score(y_test[label
], predictions_NB)))
    print(label + " F1 Score - " + str(f1_score(y_test[label], predictio
ns_NB)))
    print(label + " ROC-AUC Score - " + str(roc_auc_score(y_test[label],
predictions_NB)) + '\n')
    return predictions_NB
```

Word2Vec vectors can contain -ve values but Naive Bayes don’t accept -ve values. So word2vec won't work on Naive Bayes

In [17]:
```
label_cols = ['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']
for label in label_cols:
    perform_NB_for_label(label)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-17-07bc0827b63e> in <module>
      1 label_cols = ['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']
      2 for label in label_cols:
----> 3     perform_NB_for_label(label)

<ipython-input-16-0b3ea12135b7> in perform_NB_for_label(label)
      1 def perform_NB_for_label(label):
      2     naive_classifier = naive_bayes.MultinomialNB()
----> 3     naive_classifier.fit(train_term_doc, y_train[label])
      4     predictions_NB = naive_classifier.predict(test_term_doc)
      5     print(label + " Accuracy Score - " + str(accuracy_score(y_test[label], predictions_NB)))

~/anaconda3/lib/python3.7/site-packages/sklearn/naive_bayes.py in fit(self, X, y, sample_weight)
    611         self.feature_count_ = np.zeros((n_effective_classes, n_features),
    612                                        dtype=np.float64)
--> 613         self._count(X, Y)
    614         alpha = self._check_alpha()
    615         self._update_feature_log_prob(alpha)

~/anaconda3/lib/python3.7/site-packages/sklearn/naive_bayes.py in _count(self, X, Y)
    718         """Count and smooth feature occurrences."""
    719         if np.any((X.data if issparse(X) else X) < 0):
--> 720             raise ValueError("Input X must be non-negative")
    721         self.feature_count_ += safe_sparse_dot(Y.T, X)
    722         self.class_count_ += Y.sum(axis=0)

ValueError: Input X must be non-negative
```

In [ ]:
```

```