In [24]:
```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer, TfidfVector
izer
from sklearn import model_selection, naive_bayes, svm
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MaxAbsScaler
import os.path
import pickle
```

In [25]:
```
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 [26]:
```
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 [27]:
```
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 [28]:
```
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)
```

In [29]:
```
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)
```

In [30]:
```
if os.path.isfile('Models/svm_poly_w2v.sav'):
    clf = pickle.load(open('Models/svm_poly_w2v.sav', 'rb'))
else:
    clf = svm.SVC(kernel = 'poly', gamma='scale', degree=1, verbose=1)
    clf.fit(train_term_doc, y_train['toxic'])
    pickle.dump(clf, open('Models/svm_poly_w2v.sav', 'wb'))
```

In [11]:
```
svm_pred = clf.predict(test_term_doc)
```

In [12]:
```
print('Accuracy SVM Linear Kernel:', accuracy_score(y_test['toxic'], svm
_pred))
print('F1 Score SVM Linear Kernel:', f1_score(y_test['toxic'], svm_pred
))
print('ROC-AUC Score SVM Linear Kernel:', roc_auc_score(y_test['toxic'],
svm_pred))
```

```
Accuracy SVM Linear Kernel: 0.8594536853349611
F1 Score SVM Linear Kernel: 0.6825962090752441
ROC-AUC Score SVM Linear Kernel: 0.7721996341677192
```

In [13]:
```
if os.path.isfile('Models/svm_rbf_w2v.sav'):
    clf_rbf = pickle.load(open('Models/svm_rbf_w2v.sav', 'rb'))
else:
    clf_rbf = svm.SVC(kernel = 'rbf', gamma='scale', verbose=1)
    clf_rbf.fit(train_term_doc, y_train['toxic'])
    pickle.dump(clf, open('Models/svm_rbf_w2v.sav', 'wb'))
```

```
[LibSVM]
```

In [14]:
```
svm_rbf_pred = clf_rbf.predict(test_term_doc)
```

In [15]:
```
print('Accuracy SVM RBF Kernel:', accuracy_score(y_test['toxic'], svm_rb
f_pred))
print('F1 Score SVM RBG Kernel:', f1_score(y_test['toxic'], svm_rbf_pred
))
print('ROC-AUC Score SVM RBF Kernel:', roc_auc_score(y_test['toxic'], sv
m_rbf_pred))
```

```
Accuracy SVM RBF Kernel: 0.8762907574139072
F1 Score SVM RBG Kernel: 0.7174064606088775
ROC-AUC Score SVM RBF Kernel: 0.7911421241208475
```

## Cross Validation over RBF Kernel

In [18]:
```
n_folds = 5
c_vals = np.power(float(10), range(-3, 3 + 1))
param_grid = {'C': c_vals}
grid_search = GridSearchCV(svm.SVC(kernel='rbf', gamma='scale'), param_g
rid, cv=n_folds, iid=False, n_jobs = -1, verbose=10)
```

In [19]:
```
if os.path.isfile('Models/svm_rbf_cv_w2v.sav'):
    grid_search = pickle.load(open('Models/svm_rbf_cv_w2v.sav', 'rb'))
else:
    grid_search.fit(train_term_doc, y_train['toxic'])
    pickle.dump(grid_search, open('Models/svm_rbf_cv_w2v.sav', 'wb'))
```

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent wor
kers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done   8 tasks      | elapsed:  3.1min
[Parallel(n_jobs=-1)]: Done  16 out of  35 | elapsed:  4.6min remainin
g:  5.4min
[Parallel(n_jobs=-1)]: Done  20 out of  35 | elapsed:  5.0min remainin
g:  3.8min
[Parallel(n_jobs=-1)]: Done  24 out of  35 | elapsed:  7.8min remainin
g:  3.6min
[Parallel(n_jobs=-1)]: Done  28 out of  35 | elapsed: 20.1min remainin
g:  5.0min
[Parallel(n_jobs=-1)]: Done  32 out of  35 | elapsed: 40.3min remainin
g:  3.8min
[Parallel(n_jobs=-1)]: Done  35 out of  35 | elapsed: 42.0min finished
```

In [22]:
```
svm_rbf_grid_search_predict = grid_search.predict(test_term_doc)
```

In [23]:
```
print('Accuracy SVM RBF Kernel:', accuracy_score(y_test['toxic'], svm_rb
f_grid_search_predict))
print('F1 Score SVM RBG Kernel:', f1_score(y_test['toxic'], svm_rbf_grid
_search_predict))
print('ROC-AUC Score SVM RBF Kernel:', roc_auc_score(y_test['toxic'], sv
m_rbf_grid_search_predict))
```

```
Accuracy SVM RBF Kernel: 0.8764942265628974
F1 Score SVM RBG Kernel: 0.7223873770866682
ROC-AUC Score SVM RBF Kernel: 0.7960307279456216
```

In [21]:
```
grid_search.best_estimator_
```

Out[21]:
```
SVC(C=10.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rb
f',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [ ]:
```

```