In [1]:
```
# Import libraries
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

from sklearn.metrics import accuracy_score, roc_auc_score,f1_score, prec
ision_score, recall_score, classification_report, confusion_matrix
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import Normalizer
from sklearn.model_selection import cross_validate
from sklearn.model_selection import train_test_split


from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler

from sklearn.feature_extraction.text import TfidfVectorizer, TfidfTransf
ormer
from scipy import sparse
```

Task 1

In [2]:
```
# Read the data

train_data = pd.read_csv('reddit_200k_train.csv',encoding='latin1')
train_data = train_data[['body','REMOVED']]
test_data = pd.read_csv('reddit_200k_test.csv',encoding='latin1')
test_data = test_data[['body','REMOVED']]
```

In [3]:
```
train_data.head()
```

Out[3]:

	body	REMOVED
0	I've always been taught it emerged from the ea...	False
1	As an ECE, my first feeling as "HEY THAT'S NOT...	True
2	Monday: Drug companies stock dives on good new...	True
3	i learned that all hybrids are unfertile i won...	False
4	Well i was wanting to get wasted tonight. Not...	False

In [4]:
```
len(train_data)
```

Out[4]: 167529

In [5]:
```
train_data.groupby('REMOVED').count()
```

Out[5]:

	body
REMOVED	
False	102791
True	64738

The data clearly is imbalanced and we will under sample it before performing the baseline model

In [6]:
```
len(test_data)
```

Out[6]: 55843

TASK 1 Bag of Words and simple Features

1.1 Create a baseline model using a bag-of-words approach and a linear model.

In [7]:
```
def get_all_data():
    train_X = train_data[['body']]
    train_y = train_data[['REMOVED']]
    return train_X,train_y
```

Under Sampling

As the data is highly skewed, we are undersampling the data to make it balanced.

In [8]:
```
# under sample the data
train_X,train_y = get_all_data()
rus = RandomUnderSampler(replacement=False)
train_X_subsample, train_y_subsample = rus.fit_sample(
    train_X, train_y)
```

In [9]:
```
train_X_subsample.shape
```

Out[9]:
```
(129476, 1)
```

In [13]:
```
# create count vectorizer
countvect = CountVectorizer()
X = countvect.fit_transform(train_X_subsample.ravel())
```

In [14]:
```
# test the base line model
scores = cross_validate(LogisticRegression(),
                        X, train_y_subsample, cv=5,
                        scoring=('accuracy','average_precision','recall'
,'f1'))

print("-------test_accuracy------\n"+str(scores['test_accuracy'].mean
()))
print("-------test_average_precision------\n"+str(scores['test_average_p
recision'].mean()))
print("-------test_f1------\n"+str(scores['test_f1'].mean()))
print("-------recall------\n"+str(scores['test_recall'].mean()))
```

```
-------test_accuracy------
0.6890851700496398
-------test_average_precision------
0.7051205088440811
-------test_f1------
0.7112509023778023
-------recall------
0.7658715080867582
```

1.2 Try using n-grams, characters, tf-idf rescaling and possibly other ways to tune the BoW model. Be aware that you might need to adjust the (regularization of the) linear model fordifferent feature sets

1) We will remove stop word for all the cases following

1. N Gram

The first step is to find the best parameters for n gram approach. We will tune the following parameters

1. min_df
2. ngram_range
3. C

Grid search cv to get best parameters

In [16]:
```
# run grid search for parameter tuning

from sklearn.metrics import make_scorer
pipeline = Pipeline([
    ('vect', CountVectorizer(stop_words='english')),
    ('lr',LogisticRegression(penalty='l2'))

])
parameters = {
    'vect__min_df': (5,10),
    'vect__ngram_range': ((1, 2),(1,1)),
    'lr__C':(0.1,0.05)
}

grid_search = GridSearchCV(pipeline, parameters, cv=5,
                               n_jobs=-1, verbose=1)
print("Performing grid search...")
print("pipeline:", [name for name, _ in pipeline.steps])
print("parameters:")
print(parameters)
grid_search.fit(train_X_subsample.ravel(), train_y_subsample)
print("Best score: %0.3f" % grid_search.best_score_)
print("Best parameters set:")
best_parameters = grid_search.best_estimator_.get_params()
for param_name in sorted(parameters.keys()):
    print("\t%s: %r" % (param_name, best_parameters[param_name]))
```

```
Performing grid search...
pipeline: ['vect', 'lr']
parameters:
{'vect__min_df': (5, 10), 'vect__ngram_range': ((1, 2), (1, 1)), 'lr__
C': (0.1, 0.05)}
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:  2.1min finished

Best score: 0.691
Best parameters set:
        lr__C: 0.05
        vect__min_df: 5
        vect__ngram_range: (1, 2)
```

Now performing cross validation on best parameters to get model evaluations

In [58]:
```
# run cross validation to test the model

pipeline = make_pipeline(CountVectorizer(stop_words='english',ngram_rang
e=(1,2),min_df=5),LogisticRegression(C=0.05,penalty='l2'))

scores = cross_validate(pipeline,
                        train_X_subsample.ravel(), train_y_subsample, cv
=5,
                        scoring=('accuracy','average_precision','recall'
,'f1'))
print("Model Performance with best parameters")
print("-------test_accuracy------\n"+str(scores['test_accuracy'].mean
()))
print("-------test_average_precision------\n"+str(scores['test_average_p
recision'].mean()))
print("-------test_f1------\n"+str(scores['test_f1'].mean()))
print("-------recall------\n"+str(scores['test_recall'].mean()))
```

```
Model Performance with best parameters
-------test_accuracy------
0.6921592335082319
-------test_average_precision------
0.7149158896517699
-------test_f1------
0.7245863939142969
-------recall------
0.8098798065514549
```

We can see that recall increased significantly and precision and f1 score scores have increased slightly from the base line model

---

2. tf-idf

Next Approach is to introduce a tf-idf count vectorizer. We will remove the stop words from the data and also introduce a L2 penalty on the data.

We will tune parameters following parameters

1. min_df
2. C

Grid search to find best parameters for tf-idf

In [60]:
```
# run grid search to tune the parameters

pipeline = Pipeline([
    ('tfid',  TfidfVectorizer(stop_words='english')),
    ('lr',LogisticRegression(penalty='l2'))
])
parameters = {
    'tfid__min_df': (5,10),
    'lr__C':(0.1,0.2,0.05)
}
grid_search_tf_idf = GridSearchCV(pipeline, parameters, cv=5,
                               n_jobs=-1, verbose=1)
print("Performing grid search...")
print("pipeline:", [name for name, _ in pipeline.steps])
print("parameters:")
print(parameters)
grid_search_tf_idf.fit(train_X_subsample.ravel(), train_y_subsample)
print("Best score: %0.3f" % grid_search_tf_idf.best_score_)
print("Best parameters set:")
best_parameters = grid_search_tf_idf.best_estimator_.get_params()
for param_name in sorted(parameters.keys()):
    print("\t%s: %r" % (param_name, best_parameters[param_name]))
```

```
Performing grid search...
pipeline: ['tfid', 'lr']
parameters:
{'tfid__min_df': (5, 10), 'lr__C': (0.1, 0.2, 0.05)}
Fitting 5 folds for each of 6 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  1.2min finished

Best score: 0.694
Best parameters set:
        lr__C: 0.2
        tfid__min_df: 5
```

Now run cross-validation on best parameters obtained from grid search for tf-idf above

In [61]:
```
# run cross validation to test the model

pipeline = make_pipeline(TfidfVectorizer(stop_words='english',min_df=5),
LogisticRegression(C=0.2,penalty='l2'))

scores = cross_validate(pipeline,
                        train_X_subsample.ravel(), train_y_subsample, cv
=5,
                        scoring=('accuracy','average_precision','recall'
,'f1'))
print("Tf-idf model Performance with best parameters")
print("-------test_accuracy------\n"+str(scores['test_accuracy'].mean
()))
print("-------test_average_precision------\n"+str(scores['test_average_p
recision'].mean()))
print("-------test_f1------\n"+str(scores['test_f1'].mean()))
print("-------recall------\n"+str(scores['test_recall'].mean()))
```

```
Tf-idf model Performance with best parameters
-------test_accuracy------
0.6938584080068454
-------test_average_precision------
0.7286779195419226
-------test_f1------
0.7107047816199041
-------recall------
0.7520776119193578
```

Using tf-idf without ngram reduced the recall whereas precision and f1 score remained the same

We can again run grid search CV to tune a different set of paramters

In [63]:
```
pipeline = Pipeline([
    ('tfid',  TfidfVectorizer(stop_words='english')),
    ('lr',LogisticRegression())
])
parameters = {
    'tfid__min_df': (5,10),
     'lr__penalty':('l1','l2'),
    'lr__C':(0.1,0.2,0.05,0.3,0.6)
}
grid_search_tf_idf = GridSearchCV(pipeline, parameters, cv=5,
                               n_jobs=-1, verbose=1)
print("Performing grid search...")
print("pipeline:", [name for name, _ in pipeline.steps])
print("parameters:")
print(parameters)
grid_search_tf_idf.fit(train_X_subsample.ravel(), train_y_subsample)
print("Best score: %0.3f" % grid_search_tf_idf.best_score_)
print("Best parameters set:")
best_parameters = grid_search_tf_idf.best_estimator_.get_params()
for param_name in sorted(parameters.keys()):
    print("\t%s: %r" % (param_name, best_parameters[param_name]))
```

```
Performing grid search...
pipeline: ['tfid', 'lr']
parameters:
{'tfid__min_df': (5, 10), 'lr__penalty': ('l1', 'l2'), 'lr__C': (0.1,
0.2, 0.05, 0.3, 0.6)}
Fitting 5 folds for each of 20 candidates, totalling 100 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  3.4min finished

Best score: 0.696
Best parameters set:
        lr__C: 0.6
        lr__penalty: 'l2'
        tfid__min_df: 5
```

In [64]:
```
pipeline = make_pipeline(TfidfVectorizer(stop_words='english',min_df=5),
LogisticRegression(C=0.6,penalty='l2'))

scores = cross_validate(pipeline,
                        train_X_subsample.ravel(), train_y_subsample, cv
=5,
                        scoring=('accuracy','average_precision','recall'
,'f1'))
print("Tf-idf model Performance with best parameters")
print("-------test_accuracy------\n"+str(scores['test_accuracy'].mean
()))
print("-------test_average_precision------\n"+str(scores['test_average_p
recision'].mean()))
print("-------test_f1------\n"+str(scores['test_f1'].mean()))
print("-------recall------\n"+str(scores['test_recall'].mean()))
```

```
Tf-idf model Performance with best parameters
-------test_accuracy------
0.6956965541819827
-------test_average_precision------
0.7319059514580518
-------test_f1------
0.7105033377491367
-------recall------
0.7468256613981399
```

The different parameter set does not change the accuracy of the model much

The tf-idf vectorizer model perform best with the parameter set
Best parameters set:
lr**C: 0.2**
tfidmin_df: 5

---

3. Charcater n gram

The next approach would be to use character n gram instead of using word ngram or tf-idf vectorizer. We will tune the following parameters for char n gram now.
As done previousluy we will remove the stop words and introduce a L2 penalty on the data

1. ngram_range
2. C

grid serach cv for parameters

In [66]:
```
from sklearn.metrics import make_scorer
pipeline = Pipeline([
    ('vect', CountVectorizer(stop_words='english',min_df=5,analyzer="cha
r_wb")),
    ('lr',LogisticRegression(penalty='l2'))

])
parameters = {
    'vect__ngram_range': ((2, 3),(1,3)),
    'lr__C':(0.2,0.05)
}

grid_search_char = GridSearchCV(pipeline, parameters, cv=5,
                               n_jobs=-1, verbose=1)
print("Performing grid search...")
print("pipeline:", [name for name, _ in pipeline.steps])
print("parameters:")
print(parameters)
grid_search_char.fit(train_X_subsample.ravel(), train_y_subsample)
print("Best score: %0.3f" % grid_search_char.best_score_)
print("Best parameters set:")
best_parameters = grid_search_char.best_estimator_.get_params()
for param_name in sorted(parameters.keys()):
    print("\t%s: %r" % (param_name, best_parameters[param_name]))
```

```
Performing grid search...
pipeline: ['vect', 'lr']
parameters:
{'vect__ngram_range': ((2, 3), (1, 3)), 'lr__C': (0.2, 0.05)}
Fitting 5 folds for each of 4 candidates, totalling 20 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed: 26.9min finished

Best score: 0.709
Best parameters set:
        lr__C: 0.2
        vect__ngram_range: (1, 3)
```

Run cross validation on best parameters obtained above for char n gram

In [68]:
```
pipeline = make_pipeline(CountVectorizer(stop_words='english',min_df=5,a
nalyzer="char_wb",ngram_range=(1,3)),LogisticRegression(C=0.2,penalty='l
2'))

scores = cross_validate(pipeline,
                        train_X_subsample.ravel(), train_y_subsample, cv
=5,
                        scoring=('accuracy','average_precision','recall'
,'f1'))
print("Char-ngram model Performance with best parameters")
print("-------test_accuracy------\n"+str(scores['test_accuracy'].mean
()))
print("-------test_average_precision------\n"+str(scores['test_average_p
recision'].mean()))
print("-------test_f1------\n"+str(scores['test_f1'].mean()))
print("-------recall------\n"+str(scores['test_recall'].mean()))
```

```
Char-ngram model Performance with best parameters
-------test_accuracy------
0.7083629453975749
-------test_average_precision------
0.757138022180604
-------test_f1------
0.7321218569996074
-------recall------
0.7970588367933058
```

Using char n gram gives the best results by far:

1. precision and f1 score increased significantly
2. best over all accuracy achieved
3. Recall remains the high and similar to other models used above

---

Combine the above approaches

Let us try to combine the approaches char n gram and tf-idf tuned above to see how our model performs when both the methods are used

In [23]:
```
# combine char n gram and tf-idf

pipeline = make_pipeline(CountVectorizer(stop_words='english',min_df=5,a
nalyzer="char_wb",ngram_range=(1,3)),TfidfTransformer(),LogisticRegressi
on(C=0.2,penalty='l2'))

scores = cross_validate(pipeline,
                        train_X_subsample.ravel(), train_y_subsample, cv
=5,
                        scoring=('accuracy','average_precision','recall'
,'f1'))
print("Combinations of above model performance with best parameters")
print("-------test_accuracy------\n"+str(scores['test_accuracy'].mean
()))
print("-------test_average_precision------\n"+str(scores['test_average_p
recision'].mean()))
print("-------test_f1------\n"+str(scores['test_f1'].mean()))
print("-------recall------\n"+str(scores['test_recall'].mean()))
```

```
Combinations of above model performance with best parameters
-------test_accuracy------
0.7034971739898499
-------test_average_precision------
0.7513355709056339
-------test_f1------
0.7082026785256078
-------recall------
0.7196236986135749
```

We can say that the combination of char n gram and tf-idf does not perform that well, as compared to when they are used individually

---

1.3 Extract other features

Create new features:

We will create the following new features: from our data

1. Length of document
2. No of hyperlinks in a document
3. No of exclaimation mark in the document
4. No of dots in the document
5. If the document contain a HTML tag
6. No of capital letters in the document

In [10]:
```
train_X_subsample
```

Out[10]:
```
array([["I always think of it as a part of our bodies failing to adapt
to modern civilization fast enough.\r\n\r\nThousands of years ago, we'd
be exhausted from a day of hard labor, nothing on our minds but waking
up to work the next day.\r\n\r\nNow, some of us may still feel that wa
y, but in our current society we're conditioned to constantly check our
phones throughout the day and at night, an endless waterfall of content
and alerts that our brain tries to juggle. \r\n\r\nAnd when we drift of
f, alone with our thoughts, theres a seemingly endless stream of relati
onships and previous encounters popping into our head, coupled with a s
wathe of problems so far in the future we may never face them, but ca
n't help think of them."],
       ["Too bad it wasn't made from marble.  Marble is pretty exciting
right now. "],
       ["Did you and the team pick this mission or did you get assigned
to it? Was this y'alls life long passion to study the Moon?"],
       ...,
       ["What's scarier, dehumanizing objectifying materialism that is
legal and not reported as abuse or dehumanizing objectifying materialis
m that is illegal and seen as abuse. Morality doesn't depend on legalit
y. "],
       ["Eventually they're going to use crispr to make pig-human chime
ras "],
       ['So india needs to make more saffron']], dtype=object)
```

In [11]:
```
train_X_subsample_df = pd.DataFrame({'body':train_X_subsample[:,0]})
train_X_subsample_df.head()
```

Out[11]:

	body
0	I always think of it as a part of our bodies f...
1	Too bad it wasn't made from marble. Marble is...
2	Did you and the team pick this mission or did ...
3	Tell that to our four failed cycles. Only one ...
4	When I'm at home by myself with my 2 year old ...

Add new features as new columns to our under sampled dataframe

In [12]:
```
#  create new features

train_X_subsample_df['length'] = train_X_subsample_df['body'].apply(lamb
da x: (len(x)))
train_X_subsample_df['totalHyperlink'] = train_X_subsample_df['body'].st
r.count('http')
train_X_subsample_df['totalExclaimation'] = train_X_subsample_df['body']
.str.count('!')
train_X_subsample_df['totalDots'] = train_X_subsample_df['body'].str.cou
nt('.')
train_X_subsample_df['totalUpperCase'] = train_X_subsample_df['body'].ap
ply(lambda x: sum(map(str.isupper, x)))
```

In [13]:
```
train_X_subsample_df.head()
```

Out[13]:

	body	length	totalHyperlink	totalExclaimation	totalDots	totalUpperCase
0	I always think of it as a part of our bodies f...	712	0	0	706	4
1	Too bad it wasn't made from marble. Marble is...	74	0	0	74	2
2	Did you and the team pick this mission or did ...	122	0	0	122	3
3	Tell that to our four failed cycles. Only one ...	67	0	0	67	2
4	When I'm at home by myself with my 2 year old ...	343	0	0	343	8

In [14]:
```
# Count Vectorizer for word n gram
new_count_vect = CountVectorizer(stop_words='english',ngram_range=(1,2),
min_df=5)
new_tf_idf_vect = TfidfVectorizer(stop_words='english',min_df=5)

# Tf-idf Vectorizer
X_count_vect = new_count_vect.fit_transform(train_X_subsample_df['body']
.ravel())
X_tf_idf_vect = new_tf_idf_vect.fit_transform(train_X_subsample_df['bod
y'].ravel())

# char n gram vectorizer
new_charc_vect = CountVectorizer(stop_words='english',ngram_range=(1,3),
min_df=5,analyzer="char_wb")
X_char_vect = new_charc_vect.fit_transform(train_X_subsample_df['body'].
ravel())
```

1. New features from + Tf-idf features + count vectorizer word n gram features

In [15]:
```
X_count_tfidf_vect = sparse.hstack((X_count_vect,X_tf_idf_vect))
```

In [16]:
```
newFeaturesMat = train_X_subsample_df[['length','totalHyperlink','totalE
xclaimation','totalDots','totalUpperCase']].as_matrix()
```

In [17]:
```
# merge new features to our vectorized data
new_mat1 = sparse.hstack((X_count_tfidf_vect,newFeaturesMat))
new_mat1.shape
```

Out[17]:
```
(129476, 87093)
```

In [86]:
```
#  run cross validation to test the model with new features
scores = cross_validate(LogisticRegression(C=0.2,penalty='l2'),
                        new_mat1, train_y_subsample, cv=5,
                        scoring=('accuracy','average_precision','recall'
,'f1'))
print("Logistic regression model Performance with best parameters and ne
w features")
print("-------test_accuracy------\n"+str(scores['test_accuracy'].mean
()))
print("-------test_average_precision------\n"+str(scores['test_average_p
recision'].mean()))
print("-------test_f1------\n"+str(scores['test_f1'].mean()))
print("-------recall------\n"+str(scores['test_recall'].mean()))
```

```
Logistic regression model Performance with best parameters and new feat
ures
-------test_accuracy------
0.6957428921918998
-------test_average_precision------
0.7202117224627218
-------test_f1------
0.7204302274873815
-------recall------
0.7841143900780919
```

With the new features and Tf-idf features + count vectorizer word n gram features, the model does approximately the same as the previous models.

2. New Features + Char n gram features

In [80]:
```
new_mat = sparse.hstack((X_char_vect,newFeaturesMat))
```

In [81]:
```
scores = cross_validate(LogisticRegression(C=0.2,penalty='l2'),
                        new_mat, train_y_subsample, cv=5,
                        scoring=('accuracy','average_precision','recall'
,'f1'))
print("Logistic regression model Performance with best parameters and ne
w features")
print("-------test_accuracy------\n"+str(scores['test_accuracy'].mean
()))
print("-------test_average_precision------\n"+str(scores['test_average_p
recision'].mean()))
print("-------test_f1------\n"+str(scores['test_f1'].mean()))
print("-------recall------\n"+str(scores['test_recall'].mean()))
```

```
Logistic regression model Performance with best parameters and new feat
ures
-------test_accuracy------
0.70845559397729
-------test_average_precision------
0.7566988374354118
-------test_f1------
0.7322642757393277
-------recall------
0.797367859063921
```

New features + char n gram features work similar to only char n gram features and there is no improvement

3. New features + Tf-idf features + count vectorizer word n gram features + char n gram features

In [19]:
```
#  char n gram + tf-idf + word n gram
new_mat2 = sparse.hstack((new_mat1,X_char_vect))
scores = cross_validate(LogisticRegression(C=0.2,penalty='l2'),
                        new_mat2, train_y_subsample, cv=5,
                        scoring=('accuracy','average_precision','recall'
,'f1'))
print("Logistic regression model Performance with best parameters and ne
w features")
print("-------test_accuracy------\n"+str(scores['test_accuracy'].mean
()))
print("-------test_average_precision------\n"+str(scores['test_average_p
recision'].mean()))
print("-------test_f1------\n"+str(scores['test_f1'].mean()))
print("-------recall------\n"+str(scores['test_recall'].mean()))
```

```
Logistic regression model Performance with best parameters and new feat
ures
-------test_accuracy------
0.7133985788977038
-------test_average_precision------
0.7639181696232565
-------test_f1------
0.7350176974274371
-------recall------
0.7950043891067118
```

The model with new features and Tf-id, char-n gram, word n gram vectorization works by far the best and guves the highestaccuracy,precision,recall and f1 score. We can say model imporved by adding new features

Task 2 Word Vectors

Will use a pretrained word-embedding (word2vec) from genism instead of the bag-of-words

In [49]:
```
# get the pre-trained word2vec
from gensim import models

w = models.KeyedVectors.load_word2vec_format(
    '../GoogleNews-vectors-negative300.bin', binary=True)

word2vecmodel = w
```

Split our data into training set and validation set

In [53]:
```
X_train, X_val, y_train, y_val = train_test_split(
    train_X_subsample, train_y_subsample, stratify=train_y_subsample, ra
ndom_state=42)
```

In [76]:
```
#  function to remove None documents after transformation
def get_valid_docs(w2v_docs,y):
    indexList = []
    validDocs = []
    valid_y = []
    for i,doc in enumerate(w2v_docs):
        if(len(doc)==0):
            indexList.append(i)
        else:
            validDocs.append(doc)
            valid_y.append(y[i])
    return validDocs,indexList,valid_y
```

In [63]:
```
# Transform our training data to vector using pre trained word2Vec model

alldocs = X_train.ravel()
y = y_train

vect_w2v = CountVectorizer(vocabulary=word2vecmodel.index2word)
w2v_docs = vect_w2v.inverse_transform(vect_w2v.transform(alldocs))

# There are some docs which do not have mapping to word embedding and be
come None on transformation
# Removing those documents and filtering out only the valid docs
valid_w2v_docs,indexList,valid_y = get_valid_docs(w2v_docs,y)
valid_X = np.vstack([np.mean(word2vecmodel[doc], axis=0) for doc in vali
d_w2v_docs])
```

Fit our Model

In [64]:
```
lr_w2v = LogisticRegression(C=0.2).fit(valid_X, valid_y)
print('Training Score')
lr_w2v.score(valid_X, valid_y)
```

```
Training Score
```

Out[64]: 0.6729283020423305

Test on Validation set

In [66]:
```
# Transform the validation data to vector using pre trained word2Vec mod
el

alldocs = X_val.ravel()
y = y_val

vect_w2v = CountVectorizer(vocabulary=word2vecmodel.index2word)
w2v_docs = vect_w2v.inverse_transform(vect_w2v.transform(alldocs))

# There are some docs which do not have mapping to word embedding and be
come None on transformation
# Removing those documents and filtering out only the valid docs

valid_w2v_docs,indexList,valid_y_val = get_valid_docs(w2v_docs,y)
valid_X_val = np.vstack([np.mean(word2vecmodel[doc], axis=0) for doc in
valid_w2v_docs])
```

In [75]:
```
print("Using Word Embedding, word2vec Performance on validation set")
print("------validation_set_score------\n"+str(lr_w2v.score(valid_X_val,
valid_y_val)))
print("-------validation_set_roc------\n"+str(roc_auc_score(valid_y_val,
lr_w2v.predict(valid_X_val))))
print("-------validation_set_precision------\n"+str(precision_score(vali
d_y_val,lr_w2v.predict(valid_X_val))))
print("-------validation_set_recall------\n"+str(recall_score(valid_y_va
l,lr_w2v.predict(valid_X_val))))
```

```
Using Word Embedding, word2vec Performance on validation set
------validation_set_score------
0.668233402393395
-------validation_set_roc------
0.668237619554811
-------validation_set_precision------
0.66502642609805
-------validation_set_recall------
0.6773295384234624
```

The model does not seem to perform any better, It gives rather poor scores as compared to other approaches followed in Task1 But the over all accuracy is not significantly low and this can be considered as a good base line model.