SVM on Amazon Food Reviews

Dataset

- This dataset is a pre processed dataset which contains 351k rows with two columns
- Column 1 "Clean" has the text reviews which is preprocessed removing all special characters
- Column 2 "Score" has the corresponding target variables 1 - For rating > 4 and 0 for < 2 rating
- This data is already uploaded in Kaggle and we are fetching from it

In [0]:

```
!pip install kaggle
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!kaggle datasets download -d sanjeev5/affcleaned
!unzip affcleaned
```

Preparation

Loading the data file

In [0]:

```
import pandas as pd
data = pd.read_csv('aff_cleaned.csv',sep='\t')
data = data[data.Cleaned.notnull()]
```

Splitting Data for Train & Test

In [0]:

```
from sklearn.model_selection import train_test_split,GridSearchCV
x_train,x_test,y_train,y_test =
train_test_split(data.Cleaned,data.Score,shuffle=False,test_size=0.2)
```

Importing essential packages

In [0]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.model_selection import GridSearchCV,TimeSeriesSplit,RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score,confusion_matrix
import numpy as np
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
!pip install wordcloud
from wordcloud import WordCloud
import seaborn as sns
from sklearn.preprocessing import StandardScaler
import decimal
from sklearn.svm import SVC
%env JOBLIB_TEMP_FOLDER=/tmp
```

```
Requirement already satisfied: wordcloud in /usr/local/lib/python3.6/dist-packages (1.5.0)
Requirement already satisfied: pillow in /usr/local/lib/python3.6/dist-packages (from wordcloud)
(4.0.0)
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.6/dist-packages (from
wordcloud) (1.14.6)
```

```
Requirement already satisfied: olefile in /usr/local/lib/python3.6/dist-packages (from pillow->wordcloud) (0.46)
env: JOBLIB_TEMP_FOLDER=/tmp
```

Method Declarations

In [0]:

```
ctx = decimal.Context()
"""Method to prevent python from converting numbers to scientific notation"""
ctx.prec = 20
def float_to_str(f):
    d1 = ctx.create_decimal(repr(f))
    return format(d1, 'f')


# Time series split For Linear SVM
tss = TimeSeriesSplit(n_splits=10)
# Scores list for the scores
scores = []
# Parameters for Linear SVM
linear_params = {'alpha':[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4],'penalty':['l1','elasticnet']}
# Parameters for RBF SVM
rbf_params = {'C':[2**-5,2**-3,2**-1,2**1,2**3,2**5],'gamma':[2**-5,2**-3,2**-1,2**1,2**3,2**5]}
# Time series split For RBF SVM
tss1 = TimeSeriesSplit()

# Validation Curve
def validationCurve(alpha_params,train_data,train_label,penalty,cv_scores,title):
  """Method to decide which hyper params to be chosen to avoid overfit and underfit"""
  train_scores = dict()
  for i in alpha_params:
    X_sgd = SGDClassifier(loss="hinge",alpha = i, max_iter = 2000, penalty = penalty, tol = 1e-3, n_jobs = -1,random_state = 0,shuffle = True,learning_rate = 'optimal',class_weight = 'balanced')
    X_sgd.fit(train_data,train_label)
    cccv = CalibratedClassifierCV(X_sgd)
    cccv.fit(train_data,train_label)
    train_scores[float_to_str(i)] = roc_auc_score(train_label,cccv.predict_proba(train_data)[:,1])
  print(cv_scores)
  x = list(train_scores.keys())
  y = list(train_scores.values())
  print(y)
  x1 = list(train_scores.keys())
  y1 = cv_scores
  plt.figure(figsize=(20, 5))
  plt.plot(x,y, 'or-',label='Train ROC AUC')
  plt.plot(x1, y1, 'xb-',label='CV ROC AUC')
  plt.legend()
  plt.xlabel('Alpha-Params')
  plt.ylabel('ROC AUC')
  plt.title(title)
  for xy in zip(x, np.round(y,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
  for xy in zip(x1, np.round(y1,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
  plt.show()



def linearSVM_gridSearch(train_data,train_label,linear_params,tss,title):
  """Method to plot the heatmap of the grid search CV and is only for Linear SVM"""
  Xcv = []
  sgd = SGDClassifier(loss="hinge" , max_iter = 2000, tol = 1e-3, n_jobs = -1,random_state = 0,shuffle = True,learning_rate = 'optimal',class_weight = 'balanced')
  gscv = GridSearchCV(sgd, linear_params, scoring = 'roc_auc', cv=tss, n_jobs = -1,verbose = 1)
  gscv.fit(train_data, train_label)
  scores = gscv.cv_results_['mean_test_score'].reshape(9,2).T.reshape(2,9)
  df_cm = pd.DataFrame(scores,index=['L1','Elastic_Net'], columns=[10**-4,10**-3, 10**-2, 10**-1,10**0,10**1, 10**2, 10**3, 10**4])
  plt.figure(figsize=(20, 5))
  heatmap = sns.heatmap(df_cm, annot=True, fmt="f")
  heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right')
```

```
  heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right')
  plt.ylabel('Penalty')
  plt.xlabel('Alpha Params')
  plt.title('Grid Results Heat Map')
  plt.show()
  if scores[0].mean() > scores[1].mean():
    Xcv = scores[0]
    print('L1 scores are better than Elastic Net')
  else:
    Xcv = scores[1]
    print('Elastic Net scores are better than L1')
  return Xcv



def Wordcl(title,val):
  """ Wordcloud for features"""
  wordcloud = WordCloud(
                          background_color='white',
                          max_words=200,
                          max_font_size=40,
                          random_state=42
                         ).generate(str(val))
  fig = plt.figure(1)
  plt.figure(figsize=(10, 10))
  plt.imshow(wordcloud)
  plt.axis('off')
  plt.title(title)
  plt.show()

def confusion_matrix_display(conf_mtrx,Title):
  """ Confusion Matrices"""
  class_names = [0,1]
  df_cm = pd.DataFrame(conf_mtrx, index=class_names, columns=class_names)
  TN, FP, FN, TP = conf_mtrx.ravel()
  heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
  heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right')
  heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right')
  plt.ylabel('True label')
  plt.xlabel('Predicted label')
  plt.title(Title + ' Confusion Matrix')
  plt.show()
  print('\nThe TPR is : ',TP/(TP+FN))
  print('The TNR is : ',TN/(TN+FP))
  print('The FPR is : ',FP/(FP+TN))
  print('The FNR is : ',FN/(TP+FN),'\n')

def linear_test(train_data,train_label,test_data,test_label,alpha,vect_dict):
  """Method for applying Linear SVM on test data"""
  SGD = SGDClassifier(loss="hinge",alpha = alpha, max_iter = 2000 , penalty = 'elasticnet', tol = 1
e-3, n_jobs = -1,random_state = 0,shuffle = True,learning_rate = 'optimal',class_weight = 'balanced
')
  SGD.fit(train_data,train_label)
  cccv = CalibratedClassifierCV(SGD)
  cccv.fit(train_data,train_label)
  print('The ROC AUC score on test data is :
',roc_auc_score(test_label,cccv.predict_proba(test_data)[:,1]))
  confusion_matrix_display(confusion_matrix(train_label,SGD.predict(train_data)),'Train Confusion
Matrix')
  confusion_matrix_display(confusion_matrix(test_label,SGD.predict(test_data)),'Test Confusion
Matrix')
  if vect_dict != 'na':
    vect = vect_dict.get_feature_names()
    sorted_index = np.argsort(SGD.coef_)[::-1]
    top_200_negative = sorted_index[0][0:201].tolist()
    top_200_positive = sorted_index[0][-200:].tolist()
    neg = [vect[i] for i in top_200_negative]
    pos = [vect[i] for i in top_200_positive]
    return neg,pos



def rbf_search(train_data,train_label,params,tss):
  """ To choose the best hyperparamters for the RBF Kernel SVM"""
  svc = SVC(class_weight='balanced',max_iter=2000,tol=1e-3,cache_size = 20000,probability = True,ra
ndom_state = 0)
  rscv = RandomizedSearchCV(svc,rbf_params, scoring = 'roc_auc', cv=tss, n_jobs = -1,verbose = 1)
  rscv.fit(train_data,train_label)
```

```
  rscv.fit(train_data, train_label)
  params = rscv.cv_results_['params']
  train_scores = rscv.cv_results_['mean_train_score']
  cv_scores = rscv.cv_results_['mean_test_score']
  pr = []
  for i in params:
    c_val = str(i['C'])
    gamma_val = str(i['gamma'])
    pr.append(c_val+','+gamma_val)
  df_cm = pd.DataFrame(data = [train_scores,cv_scores],index=['Train','CV'], columns=pr)
  plt.figure(figsize=(20, 5))
  heatmap = sns.heatmap(df_cm, annot=True, fmt="f")
  heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right')
  heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right')
  plt.ylabel('Type')
  plt.xlabel('(C,GAMMA) Params')
  plt.title('Grid Results Heat Map')
  plt.show()

def rbf_test(train_data,train_label,test_data,test_label,c,gam):
  """ RBF SVM on test data with best hyper params using CV"""
    svc = SVC(class_weight='balanced',C = c, gamma = gam, max_iter=2000,tol=1e-3,cache_size = 20000
,probability = True,random_state = 0)
    svc.fit(train_data,train_label)
    print('The ROC AUC score on test data is :
',roc_auc_score(test_label,svc.predict_proba(test_data)[:,1]))
    confusion_matrix_display(confusion_matrix(train_label,svc.predict(train_data)),'Train Confusion
Matrix')
    confusion_matrix_display(confusion_matrix(test_label,svc.predict(test_data)),'Test Confusion
Matrix')
```

Bag of Words

In [0]:

```
from sklearn.feature_extraction.text import CountVectorizer
BoW_dict_bigram = CountVectorizer(ngram_range = (1,2)).fit(x_train) #bi-gram
BoW_train = BoW_dict_bigram.transform(x_train)
BoW_test = BoW_dict_bigram.transform(x_test)

standardised = StandardScaler(with_mean=False).fit(BoW_train)
train_BoW_standardised = standardised.transform(BoW_train)
test_BoW_standardised = standardised.transform(BoW_test)
```

Linear SVM Grid Search

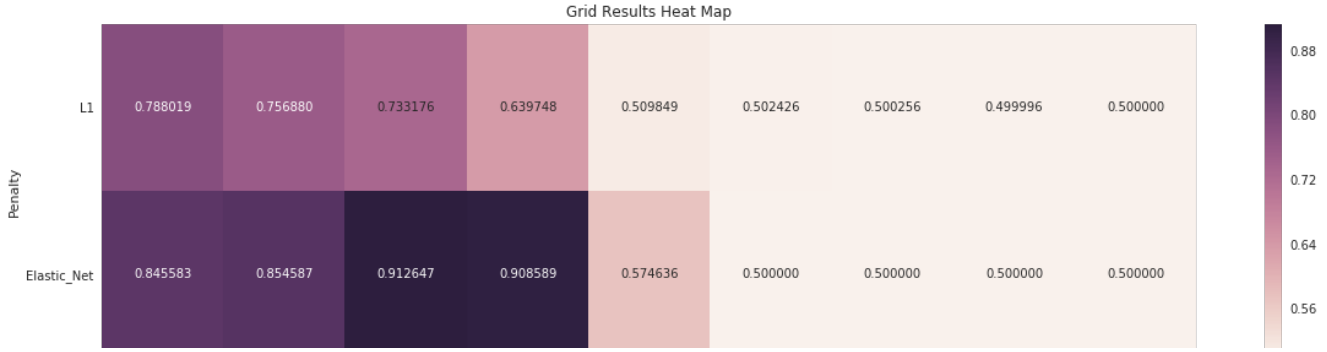
In [0]:

```
Xcv = linearSVM_gridSearch(train_BoW_standardised,y_train,linear_params,tss,'BoW Linear SVM
Validation Curve')
```

```
Fitting 10 folds for each of 18 candidates, totalling 180 fits
```

```
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 40.8min
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed: 122.0min finished
```

Out[0]:



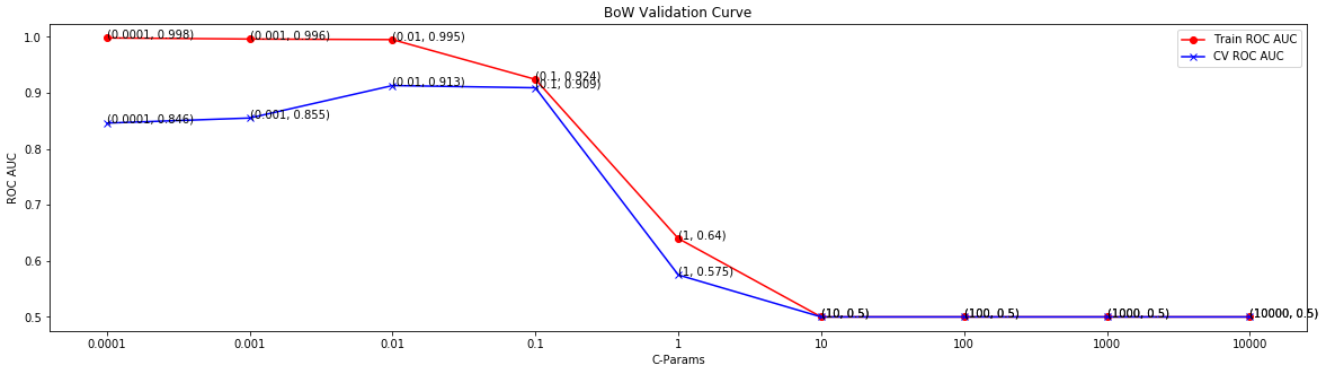

```
Elastic Net scores are better than L1
```

Linear SVM Validation Curve

In [0]:

```
validationCurve(linear_params['alpha'],train_BoW_standardised,y_train,'elasticnet',Xcv,'BoW Validation Curve')
```

Out[0]:



Linear SVM On Test Data

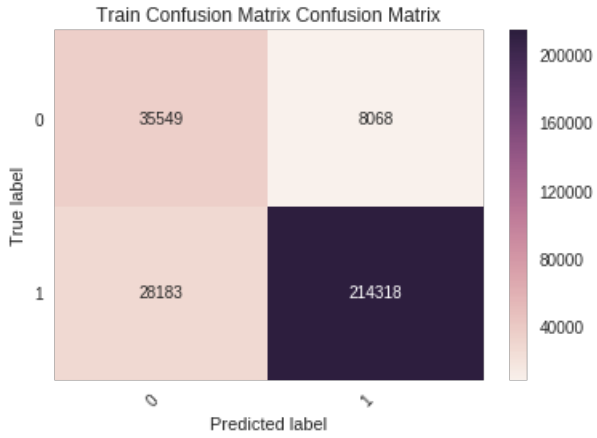
The hyper params are penalty = elsaticnet and alpha = 0.1

In [0]:

```
neg,pos = linear_test(train_BoW_standardised,y_train,test_BoW_standardised,y_test,0.1,BoW_dict_bigram)
```

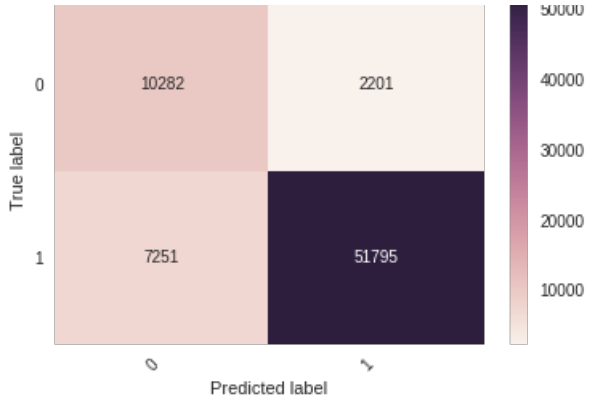
```
The ROC AUC score on test data is :  0.9254047157760541
```

Out[0]:




```
The TPR is :  0.883781922548773
The TNR is :  0.8150262512323176
The FPR is :  0.18497374876768233
The FNR is :  0.116218077451227
```

Out[0]:




```
The TPR is :  0.8771974392846256
The TNR is :  0.8236802050789073
The FPR is :  0.17631979492109268
The FNR is :  0.12280256071537446
```

Top 200 Negative Features

In [0]:

```
Wordcl('Top 200 Negative Features',neg)
```

Out[0]:

```
<matplotlib.figure.Figure at 0x7f9ed0fe8828>
```

Out[0]:

Top 200 Positive Features

In [0]:

```
Wordcl('Top 200 Positive Features',pos)
```

Out[0]:

```
<matplotlib.figure.Figure at 0x7f9ed0376f28>
```

Out[0]:

RBF Kernel

In [0]:

```
from sklearn.feature_extraction.text import CountVectorizer
BoW_dict_bigram = CountVectorizer(ngram_range = (1,2),min_df = 10,max_features = 1000).fit(x_train)
#bi-gram
BoW_train = BoW_dict_bigram.transform(x_train)
BoW_test = BoW_dict_bigram.transform(x_test)

standardised = StandardScaler(with_mean=False).fit(BoW_train)
train_BoW_standardised = standardised.transform(BoW_train)
test_BoW_standardised = standardised.transform(BoW_test)
```

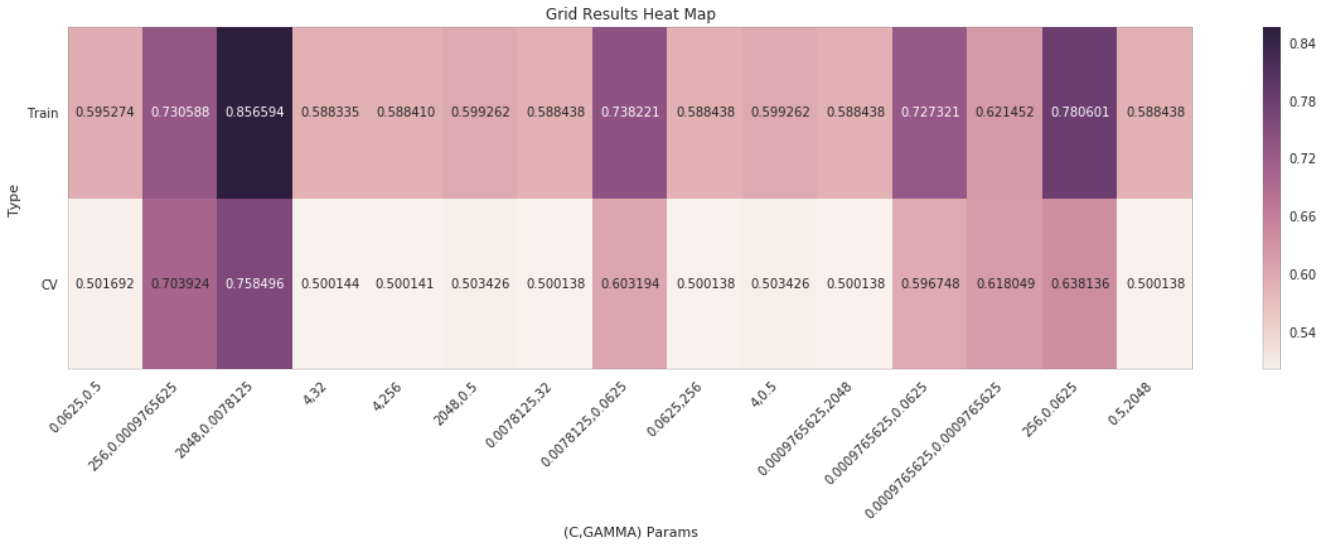
CV

In [0]:

```
rbf_search(train_BoW_standardised,y_train,rbf_params,tss1)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed: 151.7min finished
```

Out[0]:



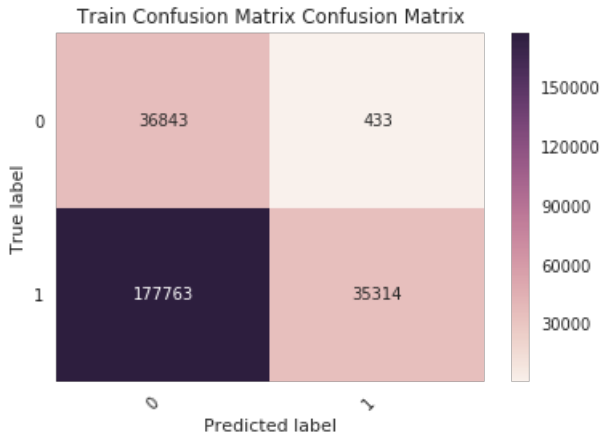
On Test Data

In [0]:

```
rbf_test(train_BoW_standardised,y_train,test_BoW_standardised,y_test,2048,0.0078125)
```

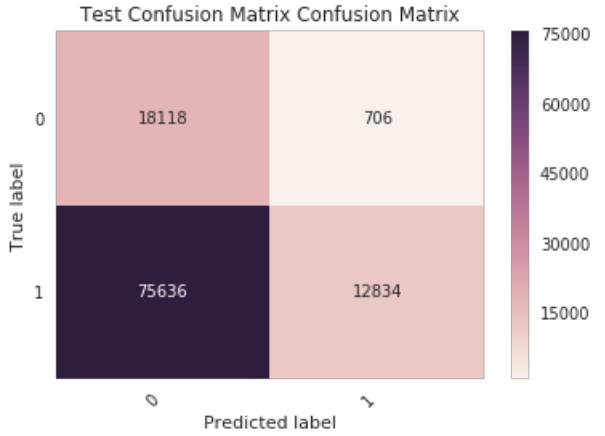
```
The ROC AUC score on test data is : 0.7269210064989702
```

Out[0]:




```
The TPR is : 0.16573351417562665
The TNR is :  0.9883839467754051
The FPR is :  0.011616053224594913
The FNR is :  0.8342664858243733
```

Out[0]:




```
The TPR is :  0.14506612410986774
The TNR is :  0.9624946876328092
The FPR is :  0.03750531236719082
The FNR is :  0.8549338758901323
```

TFIDF

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
TFIDF_dict_bigram = TfidfVectorizer(ngram_range = (1,2)).fit(x_train) #bi-gram
TFIDF_train = TFIDF_dict_bigram.transform(x_train)
TFIDF_test = TFIDF_dict_bigram.transform(x_test)

standardised = StandardScaler(with_mean=False).fit(TFIDF_train)
train_TFIDF_standardised = standardised.transform(TFIDF_train)
test_TFIDF_standardised = standardised.transform(TFIDF_test)
```

Linear SVM Grid Search

In [0]:

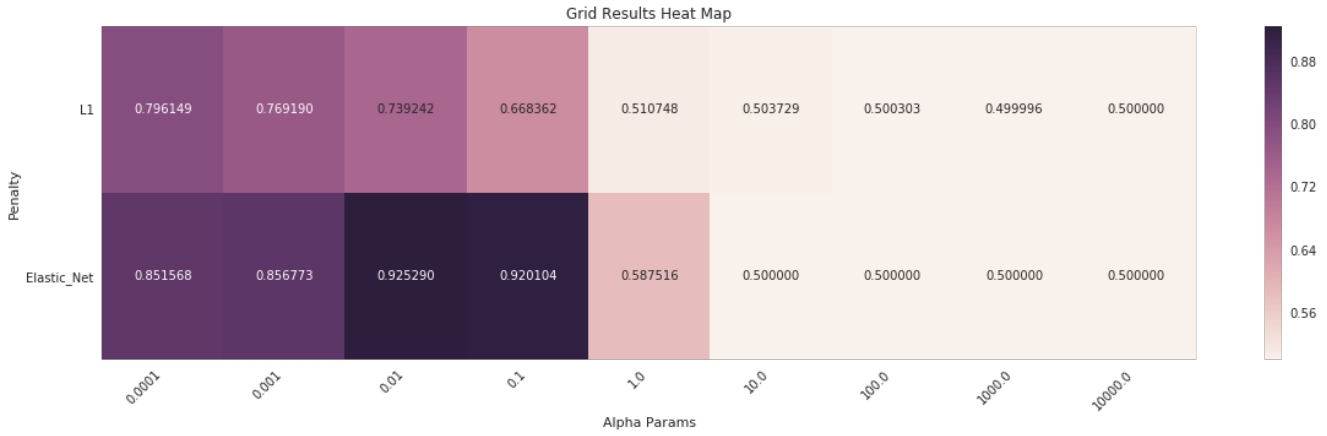
```
Xcv = linearSVM_gridSearch(train_TFIDF_standardised,y_train,linear_params,tss,'Grid Results Heat Map')
```

```
Fitting 10 folds for each of 18 candidates, totalling 180 fits
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits

[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 38.8min
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed: 118.4min finished
```

Out[0]:



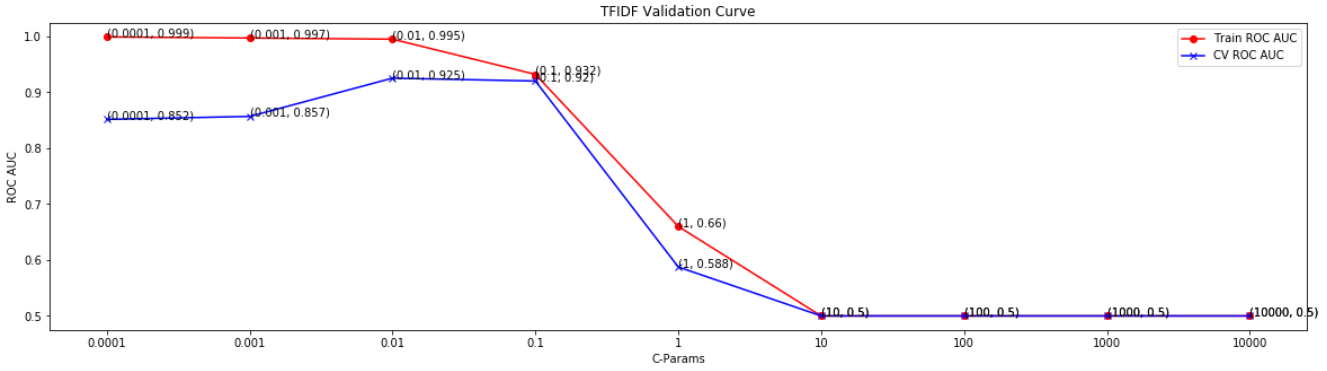
Elastic Net scores are better than L1

Linear SVM Validation Curve

In [0]:

```
validationCurve(linear_params['alpha'],train_TFIDF_standardised,y_train,'elasticnet',Xcv,'TFIDF Validation Curve')
```

Out[0]:



Linear SVM On Test Data

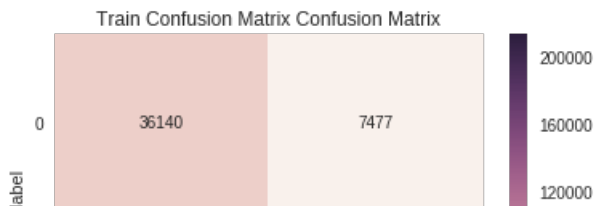
The hyperparams are penalty = 'elsatic net' and alpha = 0.1

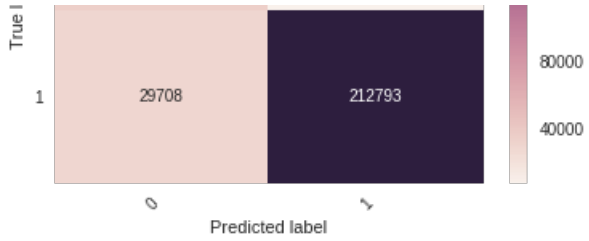
In [0]:

```
neg,pos = linear_test(train_TFIDF_standardised,y_train,test_TFIDF_standardised,y_test,0.1,TFIDF_dict_bigram)
```

```
The ROC AUC score on test data is :  0.9342486074391796
```

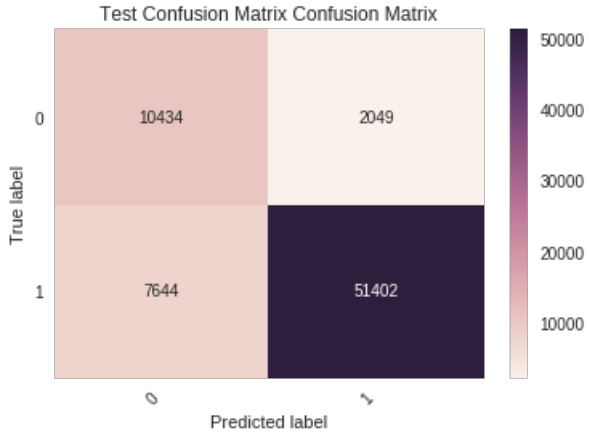
Out[0]:






```
The TPR is :  0.8774932886874693
The TNR is :  0.828576013939519
The FPR is :  0.171423986060481
The FNR is :  0.12250671131253067
```

Out[0]:




```
The TPR is :  0.8705416116248349
The TNR is :  0.8358567652006729
The FPR is :  0.16414323479932708
The FNR is :  0.12945838837516513
```

Top 200 Negative Features

In [0]:

```
Wordcl('Top 200 Negative Features',neg)
```

Out[0]:

```
<matplotlib.figure.Figure at 0x7f9eda16b978>
```

Out[0]:

Top 200 Positive Features

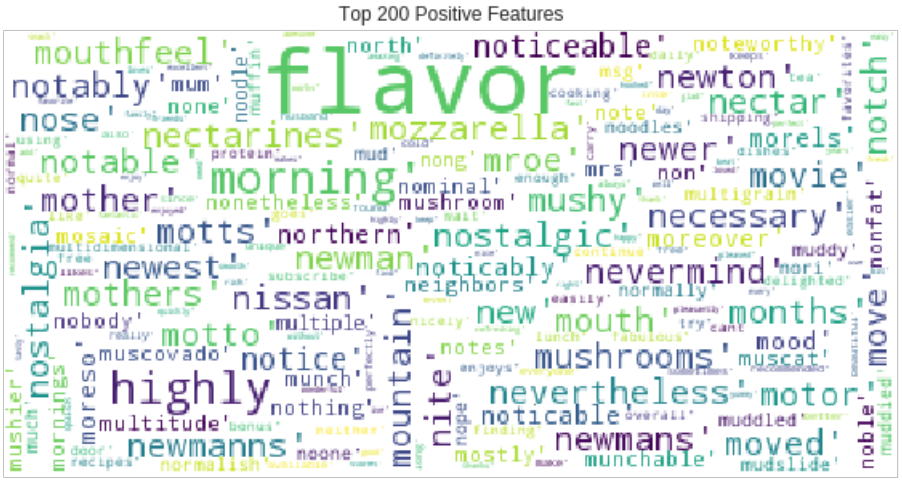
In [0]:

```
Wordcl('Top 200 Positive Features',pos)
```

Out[0]:

```
<matplotlib.figure.Figure at 0x7f9ed0374fd0>
```

Out[0]:



RBF Kernel

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
TFIDF_dict_bigram = TfidfVectorizer(ngram_range = (1,2),min_df = 10,max_features = 1000).fit(x_trai
n) #bi-gram
TFIDF_train = TFIDF_dict_bigram.transform(x_train)
TFIDF_test = TFIDF_dict_bigram.transform(x_test)

standardised = StandardScaler(with_mean=False).fit(TFIDF_train)
train_TFIDF_standardised = standardised.transform(TFIDF_train)
test_TFIDF_standardised = standardised.transform(TFIDF_test)
```

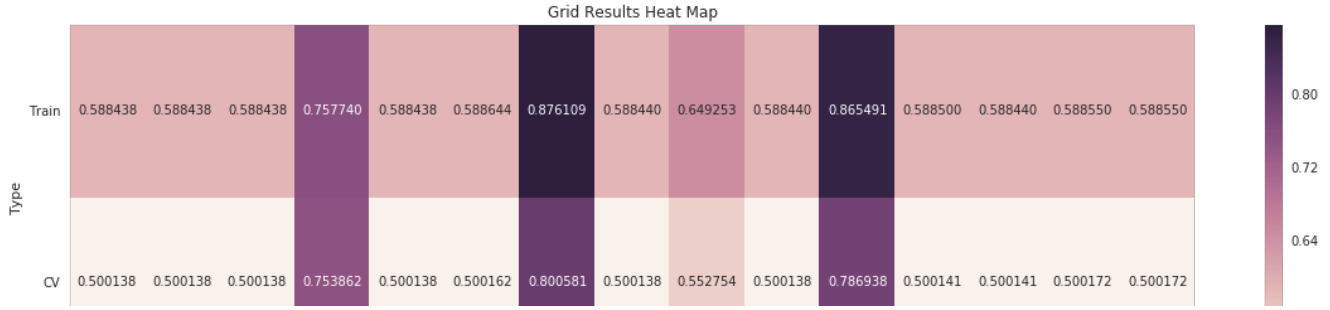
CV

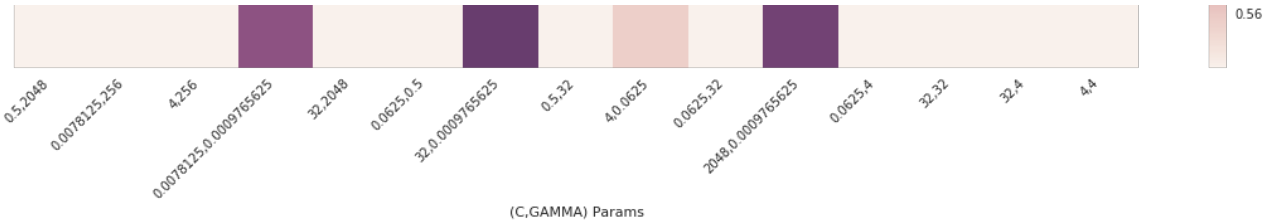
In [0]:

```
rbf_search(train_TFIDF_standardised,y_train,rbf_params,tss1)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed: 163.9min finished
```

Out[0]:





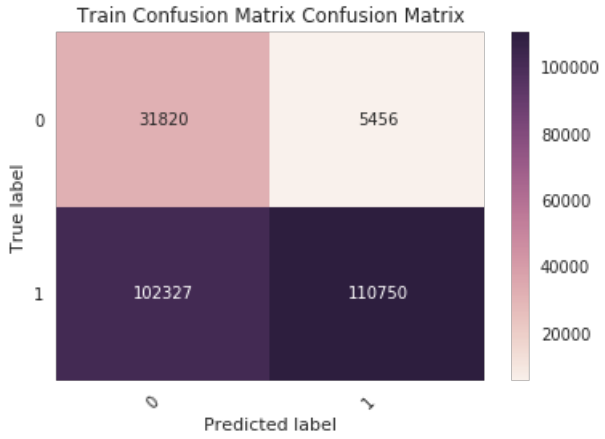
Test Data

In [0]:

```
rbf_test(train_TFIDF_standardised,y_train,test_TFIDF_standardised,y_test,32,0.0009765625)
```

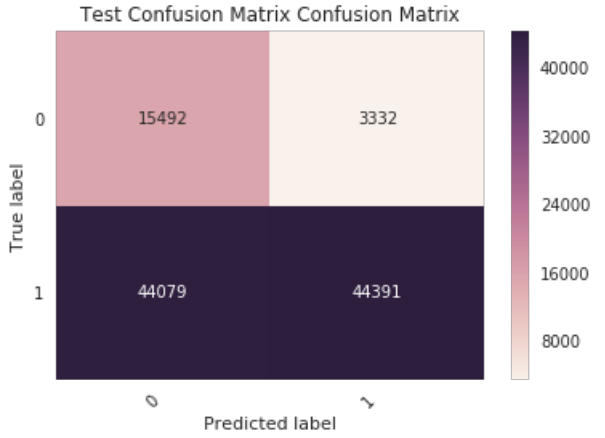
```
The ROC AUC score on test data is :  0.7371016394132082
```

Out[0]:




```
The TPR is :  0.5197651553194386
The TNR is :  0.8536323639875523
The FPR is :  0.14636763601244768
The FNR is :  0.4802348446805615
```

Out[0]:




```
The TPR is :  0.5017633095964734
The TNR is :  0.8229919252018699
The FPR is :  0.17700807479813005
The FNR is :  0.49823669040352664
```

Average W2V

In [0]:

```
import re
def cleanhtml(sentence):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence):
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned


i=0
list_of_sent=[]
for sent in x_train.values:
    filtered_sentence=[]
    sent=cleanhtml(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_of_sent.append(filtered_sentence)
```

In [0]:

```
!pip install gensim
import gensim
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

w2v_model=gensim.models.Word2Vec(list_of_sent,min_count=5,size=50, workers=8)
```

In [0]:

```
sent_vectors_train = []
for sent in x_train.values:
    sent_vec = np.zeros(50)
    cnt_words =0
    for word in sent.split():
        try:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass
    sent_vec /= cnt_words
    sent_vectors_train.append(sent_vec)

sent_vectors_train = np.nan_to_num(sent_vectors_train)
```

Linear SVM Grid Search

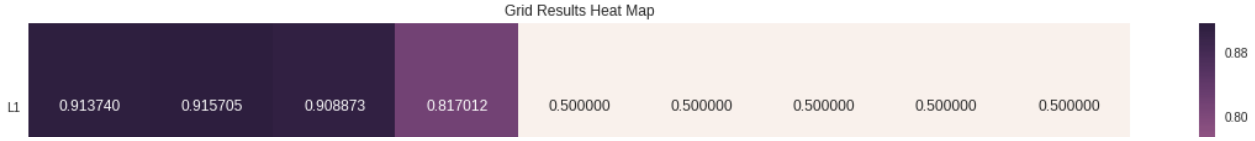
In [0]:

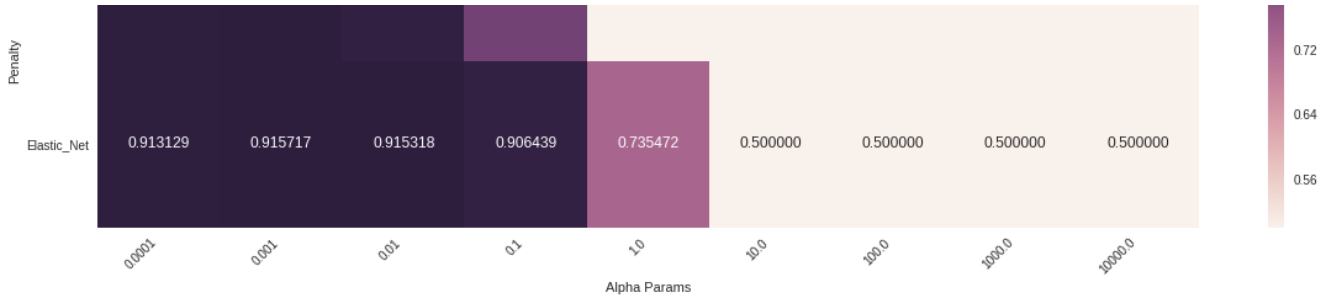
```
Xcv = linearSVM_gridSearch(sent_vectors_train,y_train,linear_params,tss,'Grid Results Heat Map')
```

```
Fitting 10 folds for each of 18 candidates, totalling 180 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed:  3.4min finished
```

Out[0]:





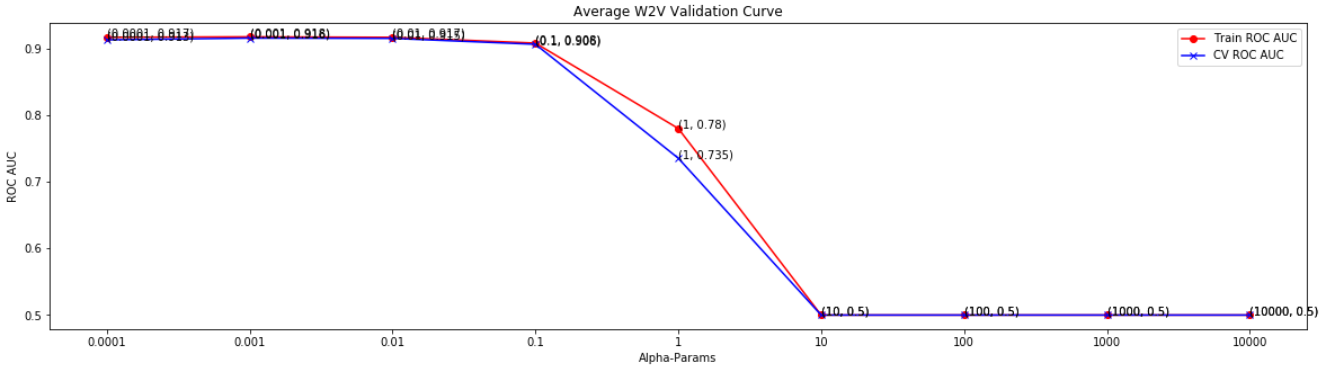
Elastic Net scores are better than L1

Linear SVM Validation Curve

In [0]:

```
validationCurve(linear_params['alpha'],sent_vectors_train,y_train,'elasticnet',Xcv,'Average W2V Validation Curve')
```

Out[0]:



Linear SVM On Test Data

In [0]:

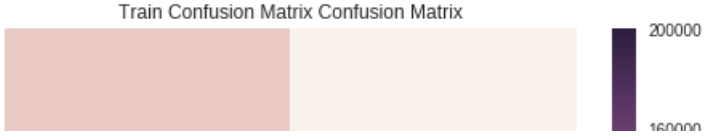
```
sent_vectors_test = []
for sent in x_test.values:
    sent_vec = np.zeros(50)
    cnt_words =0
    for word in sent.split():
        try:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass
    sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
sent_vectors_test = np.nan_to_num(sent_vectors_test)
```

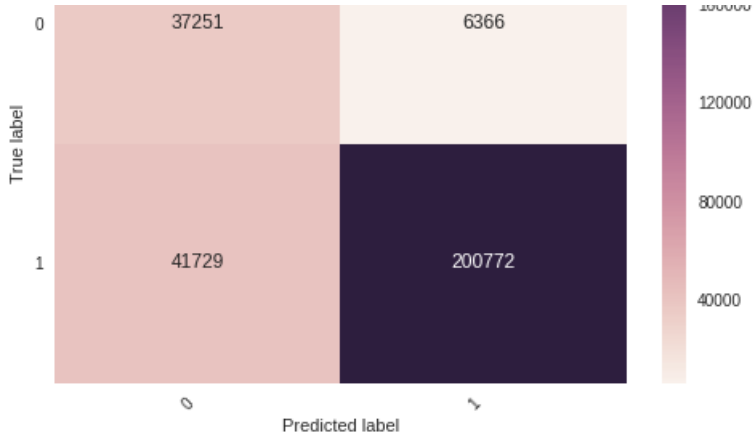
In [0]:

```
linear_test(sent_vectors_train,y_train,sent_vectors_test,y_test,0.001,'na')
```

The ROC AUC score on test data is : 0.9161320243005338

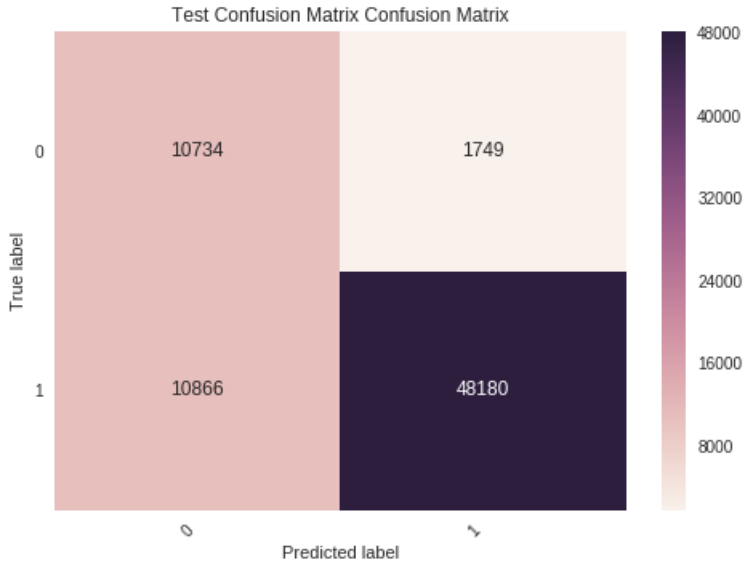
Out[0]:






```
The TPR is :  0.8279223590830553
The TNR is :  0.8540477336818213
The FPR is :  0.1459522663181787
The FNR is :  0.17207764091694466
```

Out[0]:




```
The TPR is :  0.8159739863834976
The TNR is :  0.859889449651526
The FPR is :  0.14011055034847392
The FNR is :  0.1840260136165024
```

RBF Kernel

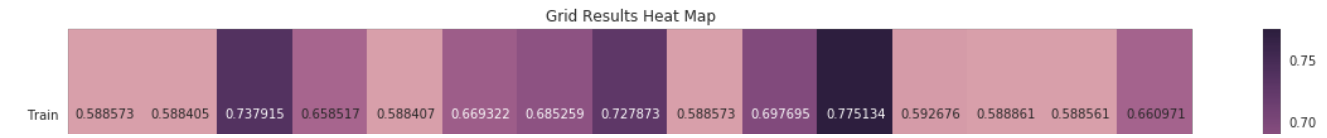
CV

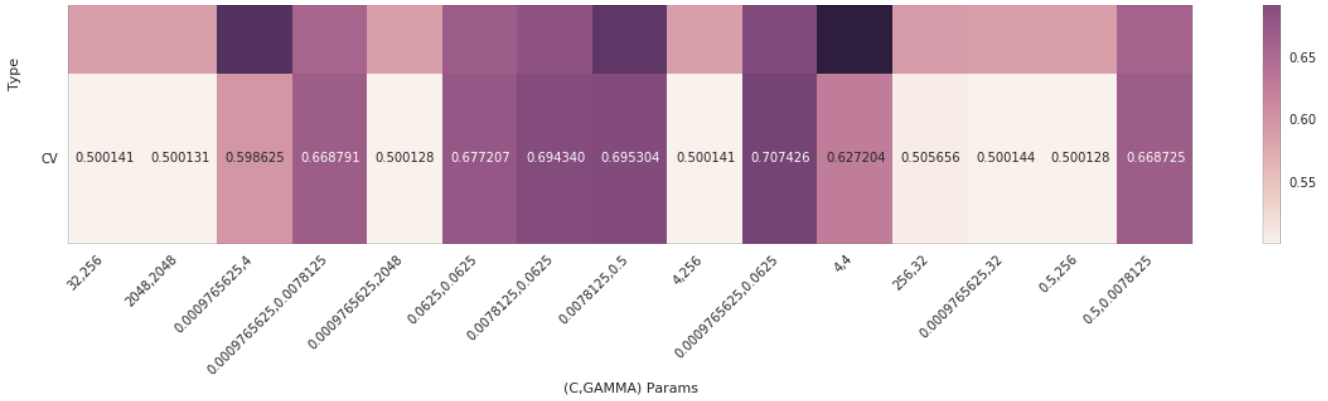
In [0]:

```
rbf_search(sent_vectors_train,y_train,rbf_params,tss1)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed: 92.3min finished
```

Out[0]:





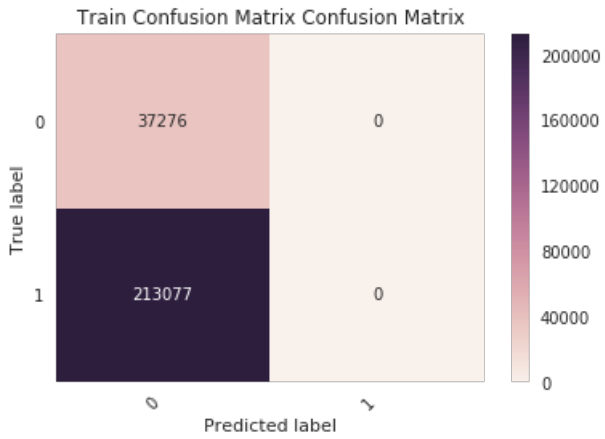
On Test Data

In [0]:

```
rbf_test(sent_vectors_train,y_train,sent_vectors_test,y_test,0.0078125,0.5)
# Performs poort even for the next few high parameter combination
```

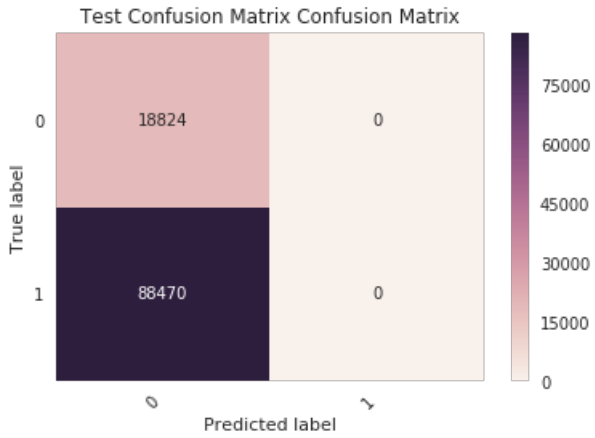
```
The ROC AUC score on test data is :  0.5
```

Out[0]:




```
The TPR is :  0.0
The TNR is :  1.0
The FPR is :  0.0
The FNR is :  1.0
```

Out[0]:




```
The TPR is :  0.0
The TNR is :  1.0
The FPR is :  0.0
The FNR is :  1.0
```

TFIDF W2V

In [0]:

```
!kaggle datasets download -d sanjeev5/w2vtfidf-train
!unzip w2vtfidf-train.zip
!kaggle datasets download -d sanjeev5/w2vtfidftest
!unzip w2vtfidftest.zip

import pickle

infile = open('test_AgTFIDF.txt','rb')
tfidf_sent_vectors_test = pickle.load(infile)
infile.close()

infile = open('AgTFIDF_Train.txt','rb')
tfidf_sent_vectors_train = pickle.load(infile)
infile.close()
```

Linear SVM Grid Search

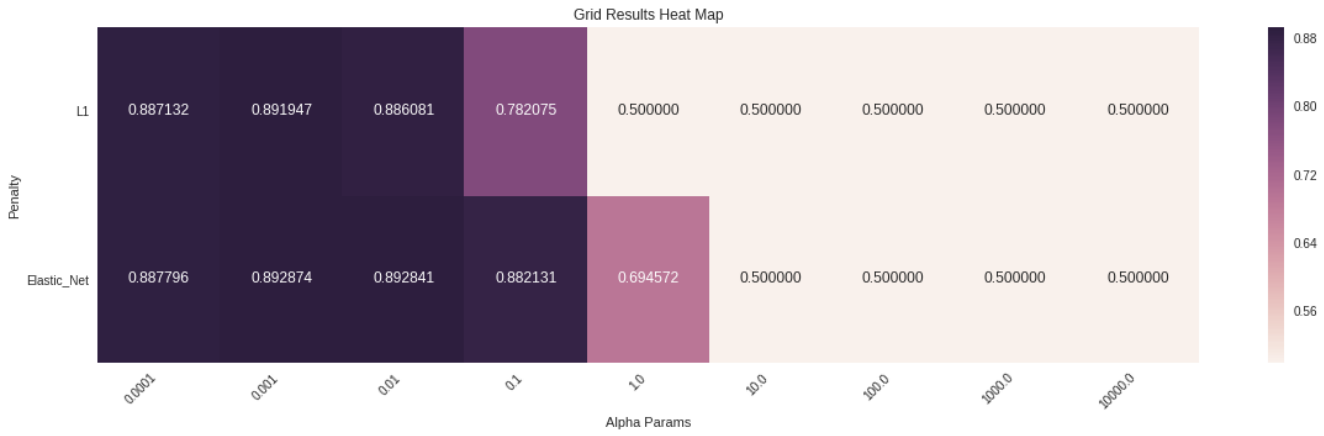
In [0]:

```
Xcv = linearSVM_gridSearch(tfidf_sent_vectors_train,y_train,linear_params,tss,'TFIDF W2V Linear SVM Validation Curve')
```

```
Fitting 10 folds for each of 18 candidates, totalling 180 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed:  3.6min finished
```

Out[0]:



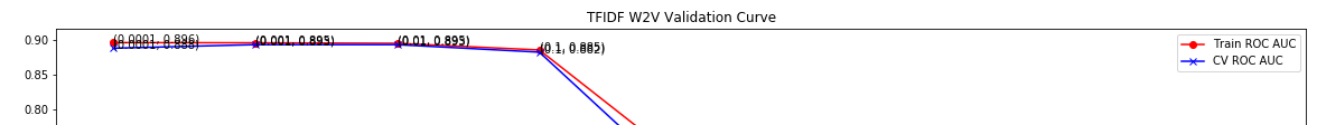

```
Elastic Net scores are better than L1
```

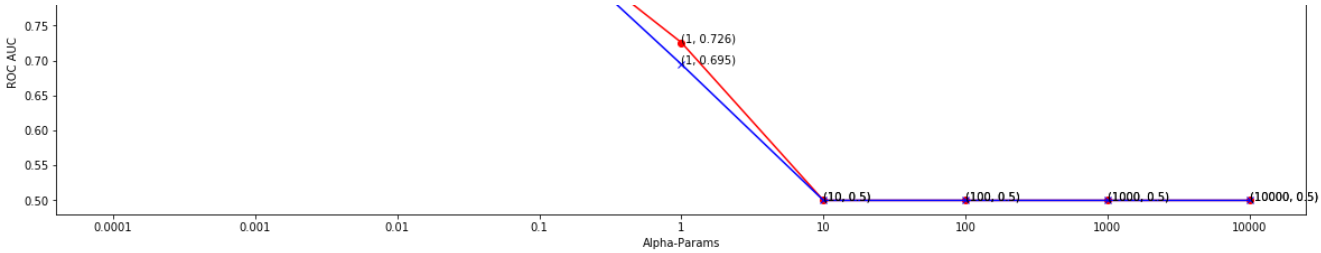
Linear SVM Validation Curve

In [0]:

```
validationCurve(linear_params['alpha'],tfidf_sent_vectors_train,y_train,'elasticnet',Xcv,'TFIDF W2V Validation Curve')
```

Out[0]:





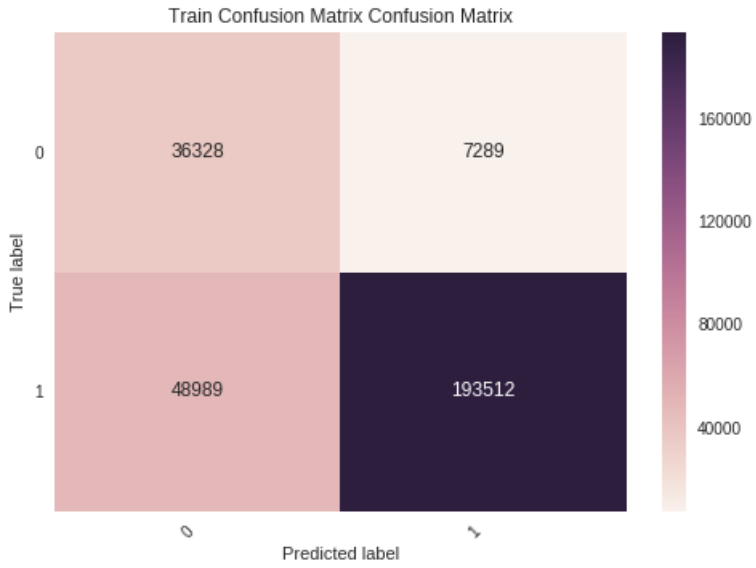
Linear SVM On Test Data

In [0]:

```
linear_test(tfidf_sent_vectors_train,y_train,tfidf_sent_vectors_test,y_test,0.01,'na')
```

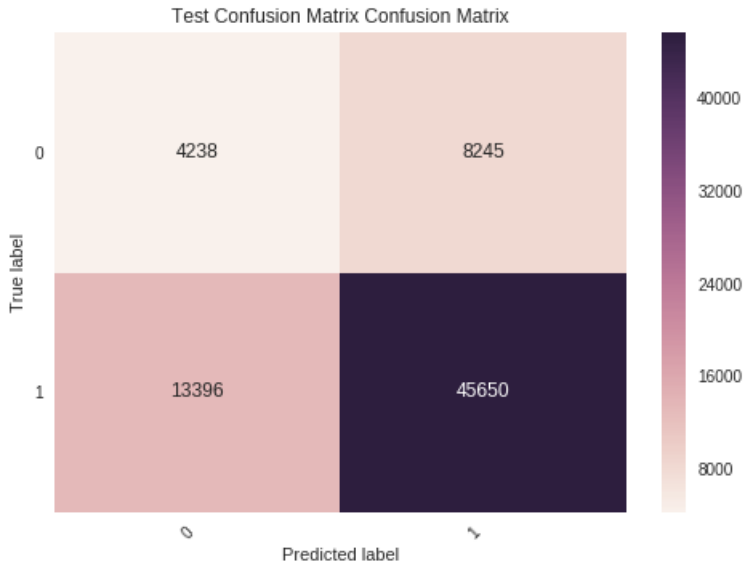
```
The ROC AUC score on test data is :  0.6033216345180907
```

Out[0]:




```
The TPR is :  0.7979843382089146
The TNR is :  0.832886259944517
The FPR is :  0.16711374005548296
The FNR is :  0.2020156617910854
```

Out[0]:



```
The TPR is :  0.7731260373268299
The TNR is :  0.3395017223423856
The FPR is :  0.6604982776576144
The FNR is :  0.22687396267317006
```

RBF Kernel

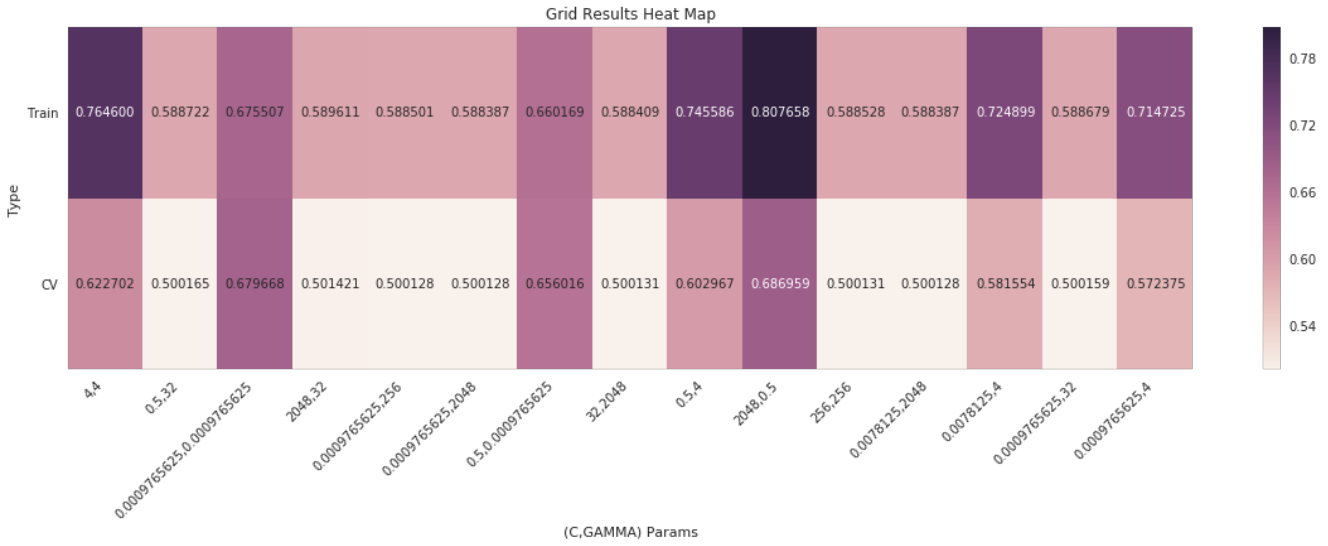
CV

In [0]:

```
rbf_search(tfidf_sent_vectors_train,y_train,rbf_params,tss1)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed: 103.4min finished
```

Out[0]:



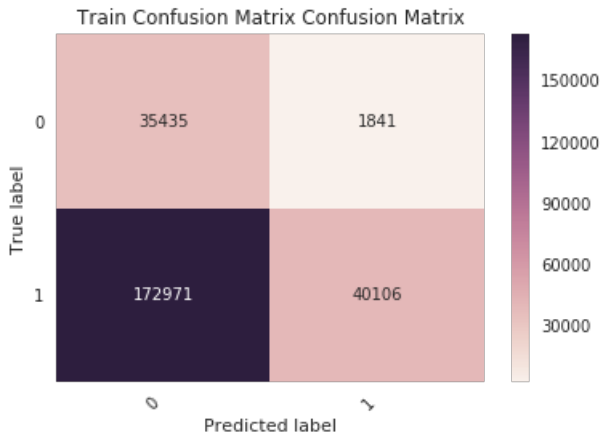
On Test Data

In [0]:

```
rbf_test(tfidf_sent_vectors_train,y_train,tfidf_sent_vectors_test,y_test,2048,0.5)
```

```
The ROC AUC score on test data is : 0.5064450927369858
```

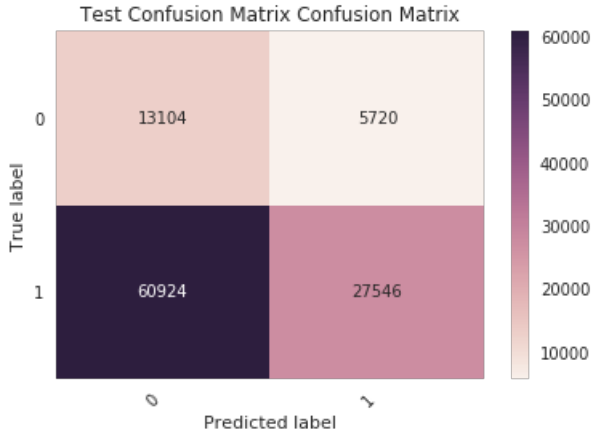
Out[0]:




```
The TPR is :  0.188223036742586
```

```
The TNR is :  0.9506116536109025
The FPR is :  0.049388346389097545
The FNR is :  0.811776963257414
```

Out[0]:




```
The TPR is :  0.31135978297728045
The TNR is :  0.6961325966850829
The FPR is :  0.30386740331491713
The FNR is :  0.6886402170227196
```

Conclusion

Model	Reg - Alpha	Train FPR	Test FPR	Train FNR	Test FNR
Bag Of Words	Elastic Net - 0.1	0.185	0.116	0.176	0.123
TFIDF	Elastic Net - 0.1	0.171	0.122	0.16	0.129
Avg W2V	Elastic Net - 0.1	0.145	0.136	0.175	0.190
TF-IDF W2V	Elastic Net - 0.1	0.173	0.257	0.193	0.601

The best is : BoW with Elastic Net regulariser with Alpha 0.1