Logistic Regression on Amazon Fine Food Review

Problem defintion

Problem description:

Our objective is to apply the Logistic Regression algorithm on the reviews and to create a model that can predict a review is a positive or negative review on unseen data.

About Input Data:

1. Amazon Fine food reviews dataset
2. Removing neutral reviews that is the Score field = 3
3. Score of 1 and 2 is considered as negative while positive reviews are of score 4 and 5
4. 0 represents negative review and 1 represents positive review

Overview:

1. We will have a train:test split up as 70:30
2. We will apply a time series split CV and use ROC-AUC as scoring
3. We will apply different text processing techniques like BoW, TF-IDF, Average W2V and TFIDF W2V.

Assumptions:

1. The distribution of test and the train data are not very different
2. The model with the lowest False Positive Rate is chosen as the best model since the positive class is dominating while the FNR is also considered to avoid the biased predictions

Running instance:

8 Core - Processor with 52 GB RAM on Google Cloud.

Dataset Pre-Processing

Downloading Dataset

In [0]:

```
!pip install -q kaggle
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!kaggle datasets download -d snap/amazon-fine-food-reviews
!unzip amazon-fine-food-reviews.zip
```

```
Warning: Your Kaggle API key is readable by other users on this system! To
fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Downloading amazon-fine-food-reviews.zip to /content
 94% 236M/251M [00:02<00:00, 107MB/s]
100% 251M/251M [00:02<00:00, 123MB/s]
Archive:  amazon-fine-food-reviews.zip
  inflating: Reviews.csv
  inflating: database.sqlite
  inflating: hashes.txt
```

Importing Dataset

In [0]:

```
import pandas as pd
input_data = pd.read_csv('Reviews.csv')
print("Shape of data is ", input_data.shape)
```

```
Shape of data is  (568454, 10)
```

Cleaning Dataset

In [0]:

```
# Removing all the neutral reviews
input_data = input_data[input_data.Score != 3]
sorted_data=input_data.sort_values('ProductId', axis=0, ascending=True, inplace=False,
kind='quicksort', na_position='last')
input_data = sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"},
 keep='first', inplace=False)
input_data = input_data[input_data.HelpfulnessNumerator<=input_data.HelpfulnessDenomina
tor]
```

Pre-Processing

In [0]:

```
sorted_data = input_data.iloc[input_data.Time.argsort()]
sorted_data.to_csv('am.csv',sep = '\t')
ii = pd.read_csv('am.csv',sep = '\t')
```

In [0]:

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
import string

sno = SnowballStemmer('english')
stop = set(stopwords.words('english'))

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
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

In [0]:

```
#Stemming
import re
i=0
str1=' '
final_string=[]
s=''
for sent in ii['Text'].values:
    filtered_sentence=[]
    sent=cleanhtml(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                else:
                    continue
            else:
                continue
    str1 = b" ".join(filtered_sentence)
    final_string.append(str1)
    i+=1


ii['Cleaned_stemmed']=final_string
ii['Cleaned_stemmed']=ii['Cleaned_stemmed'].str.decode("utf-8")
```

In [0]:

```
#Cleaning words
i=0
str1=' '
final_string=[]
s=''
for sent in ii['Text'].values:
    filtered_sentence=[]
    sent=cleanhtml(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    filtered_sentence.append(cleaned_words.lower())
                else:
                    continue
            else:
                continue
    str1 = ' '.join(filtered_sentence)
    final_string.append(str1)
    i+=1


ii['Cleaned']=final_string
```

In [0]:

```
#0 for negative reviews and 1 for positive reviews
ii.Score = ii.Score.map(lambda x : 1 if (x > 3) else 0)

pick = ii[['Cleaned','Score']] # data
pick = pick[pick.Cleaned.notnull()]
x_vect = pick.Cleaned

y = pick.Score # label

from sklearn.model_selection import train_test_split
#Splitting into train and test
X_train, X_test, Y_train, Y_test = train_test_split(x_vect, y, test_size=0.30, random_s
tate=42,shuffle=False)
#Splitting test into CV and data
```

Train Class Distribution

In [0]:

```
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
def plot_bar_val(label,ct,title):
    index = np.arange(len(label))
    plt.figure(figsize=(10,5))
    plt.bar(index, ct)
    plt.xlabel('Review Type', fontsize=15)
    plt.ylabel('Total Reviews', fontsize=15)
    plt.xticks(index, label, fontsize=15)
    plt.title(title)
    plt.show()

vc = Y_train.value_counts().to_frame()
label = ['Positive','Negative']
ct = vc.Score.values
print(vc)
#plot_bar_val(label,ct,'Train Classes')
```

```
    Score
1  216890
0   38029
```

Test Class Distribution

In [0]:

```
cvv = Y_test.value_counts().to_frame()
cv = cvv.Score.values
print(cvv)
#plot_bar_val(label,cv,'Test Classes')
```

```
   Score
1  90171
0  19081
```

Importing Essential Packages

In [0]:

```
!pip install wordcloud
#%matplotlib inline
import pandas as pd
from sklearn.linear_model import LogisticRegression
# (penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=
1,
#class_weight=None, random_state=None, solver='warn', max_iter=100,
#multi_class='warn', verbose=0, warm_start=False, n_jobs=None)[source]
from sklearn.metrics import confusion_matrix,classification_report,roc_auc_score
from sklearn.model_selection import RandomizedSearchCV,TimeSeriesSplit,GridSearchCV
import seaborn as sns
import numpy as np
from wordcloud import WordCloud
tss = TimeSeriesSplit(n_splits=10)
params = {'penalty':['l1','l2'],'C': [10**-4,10**-3, 10**-2, 10**-1,10**0,10**1, 10**2,
 10**3, 10**4]}
C_params = [10**-4,10**-3, 10**-2, 10**-1,10**0,10**1, 10**2, 10**3, 10**4]
```

```
Collecting wordcloud
  Downloading https://files.pythonhosted.org/packages/ae/af/849edf14d573eb
a9c8082db898ff0d090428d9485371cc4fe21a66717ad2/wordcloud-1.5.0-cp36-cp36m-
manylinux1_x86_64.whl (361kB)
    100% |████████████████████████████████| 368kB 25.5MB/s
Requirement already satisfied: pillow in /usr/local/lib/python3.6/dist-pac
kages (from wordcloud) (4.0.0)
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.6/di
st-packages (from wordcloud) (1.14.6)
Requirement already satisfied: olefile in /usr/local/lib/python3.6/dist-pa
ckages (from pillow->wordcloud) (0.46)
Installing collected packages: wordcloud
Successfully installed wordcloud-1.5.0
```

Method Declrations

In [0]:

```
def Wordcl(title,val):
  wordcloud = WordCloud(
                          background_color='white',
                          max_words=200,
                          max_font_size=40,
                          random_state=42
                         ).generate(str(val))
  fig = plt.figure(1)
  plt.imshow(wordcloud)
  plt.axis('off')
  plt.title(title)
  plt.show()
# Confusion Matrix
def confusion_matrix_display(conf_mtrx,tst_labels,Title):
  class_names = [0,1]
  df_cm = pd.DataFrame(conf_mtrx, index=class_names, columns=class_names)
  ts = tst_labels.value_counts().to_frame()
  TN, FP, FN, TP = conf_mtrx.ravel()
  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')
  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()


import decimal
ctx = decimal.Context()
ctx.prec = 20
def float_to_str(f):
    d1 = ctx.create_decimal(repr(f))
    return format(d1, 'f')
```

In [0]:

```
def searchLR(search_type,params,input_data,input_label,tss):
  LR = LogisticRegression(class_weight='balanced',random_state=42)
  if search_type == 'GS':
    Gscv = GridSearchCV(LR, params, scoring = 'roc_auc', cv=tss,n_jobs = -1,verbose = 1
)
    Gscv.fit(input_data, input_label)
    print('The parameters that would gives best roc_auc is : ', Gscv.best_params_)
    print('The best roc_auc achieved after parameter tuning via grid search is : ', Gsc
v.best_score_)
    scores = Gscv.cv_results_['mean_test_score'].reshape(9,2).T.reshape(2,9)
    df_cm = pd.DataFrame(scores,index=['l1','l2'], 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='righ
t')
    plt.ylabel('Penalty')
    plt.xlabel('C Params')
    plt.title('Grid Results Heat Map')
    plt.show()
    return scores

Scores_err = dict()
non_zero_dict = dict()


def l1(params,train_data,train_label):
  for i in params:
    clf = LogisticRegression(C=i, penalty='l1',n_jobs = -1,class_weight = 'balanced')
    clf.fit(train_data, train_label)
    Scores_err[float_to_str(i)] = roc_auc_score(train_label,clf.predict_proba(train_dat
a)[:,1])
    non_zero_dict[float_to_str(i)] = clf.coef_.shape[1] - np.count_nonzero(clf.coef_)
  best(Scores_err,'ROC AUC Plot','ROC AUC')
  best(non_zero_dict,'Sparsity Plot','Sparsity')
```

In [0]:

```
import matplotlib.pyplot as plt
#Start of method to choose the best 'K'
def best(error,title,find):
  x = list(error.keys())
  y = list(error.values())
  error_d = dict(zip(x, y))
  plt.figure(figsize=(20, 5))
  plt.scatter(x,y)
  plt.xlabel('C Hyperparam')
  plt.ylabel(find)
  plt.title(title)
  for xy in zip(x, np.round(y,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
  plt.show()
#End of method to choose the best 'K'

def best_k(error_dict):
  return max(error_dict, key=lambda k: error_dict[k])

def best_k_value(error):
  lowest = min(error.values())
  keys = [k for k, v in error.items() if v == lowest]
  print('The optimal \'K\' value is : ',sorted(keys)[len(keys) - 1], 'with error\
        : ',lowest)
```

In [0]:

```
def validation_curve(xticks,plot_y,plot_y1,title):
  x = xticks
  y = plot_y
  x1 = xticks
  y1 = plot_y1
  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('C-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()
```

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)
```

In [0]:

```
from sklearn.preprocessing import StandardScaler
standardised = StandardScaler(with_mean=False).fit(BoW_train)
train_BoW_standardised = standardised.transform(BoW_train)
test_BoW_standardised = standardised.transform(BoW_test)
```

In [0]:

```
%env JOBLIB_TEMP_FOLDER=/tmp
```

```
env: JOBLIB_TEMP_FOLDER=/tmp
```

GridSearchCV

In [0]:

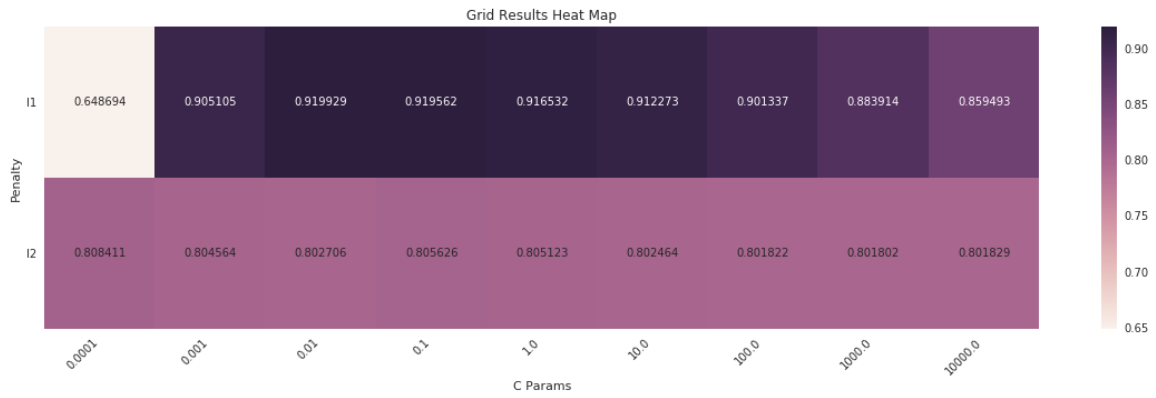
```
scores = searchLR('GS',params,train_BoW_standardised,Y_train,tss)
```

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

[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  7.8min
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed: 83.9min finished

The parameters that would gives best roc_auc is :  {'C': 0.01, 'penalty':
'l1'}
The best roc_auc achieved after parameter tuning via grid search is :  0.9
199286263640268
```

Out[0]:



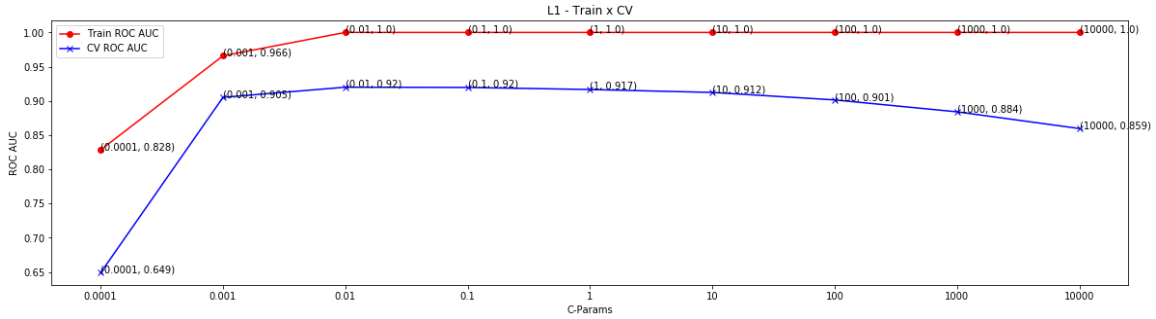
Validation Curve

In [0]:

```
from sklearn.metrics import roc_auc_score
l1_cv_scores = scores[0]
l1_train_scores = []
for i in C_params:
  LR = LogisticRegression(penalty = 'l1',C = i, class_weight='balanced',n_jobs = -1,ran
dom_state = 42)
  LR.fit(train_BoW_standardised,Y_train)
  l1_train_scores.append(roc_auc_score(Y_train,LR.predict_proba(train_BoW_standardised)
[:,1]))

x = [float_to_str(i) for i in C_params]
validation_curve(x,l1_train_scores,l1_cv_scores,'L1 - Train x CV')
```

Out[0]:

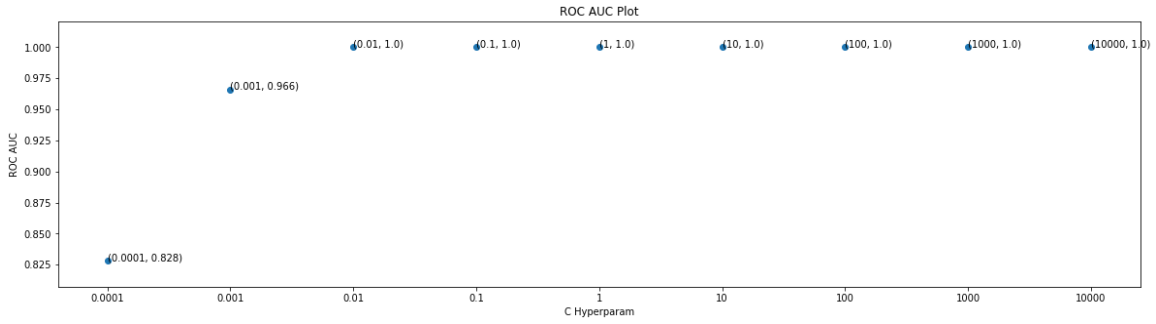


L1 Sparsity & Error

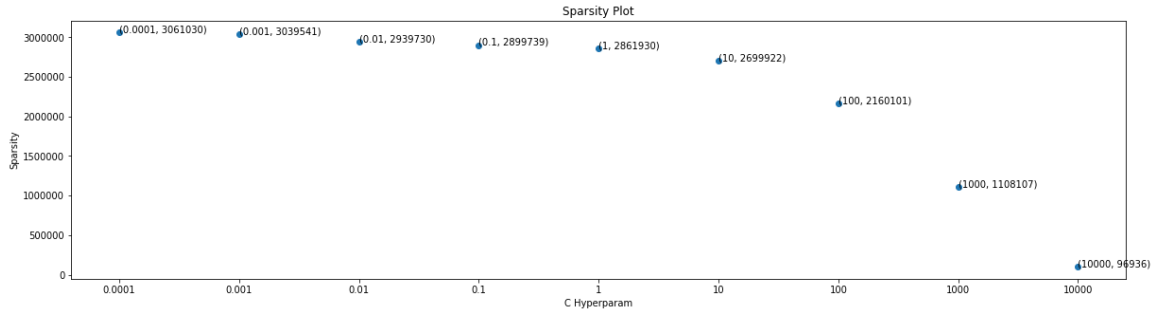
In [0]:

```
l1(C_params,train_BoW_standardised,Y_train)
```

Out[0]:



Out[0]:



On test data

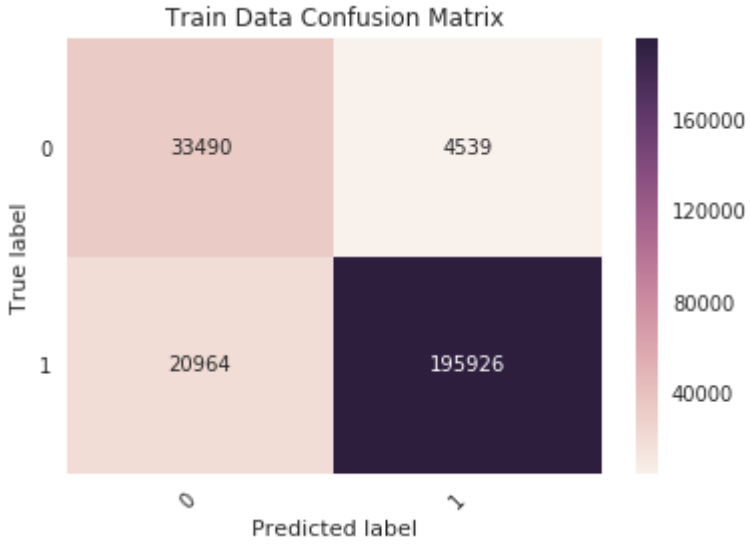
In [0]:

```
LR = LogisticRegression(penalty = 'l1',C = 0.001, class_weight='balanced',n_jobs = -1,r
andom_state = 42)
LR.fit(train_BoW_standardised,Y_train)
print('****************Train****************')
confusion_matrix_display(confusion_matrix(Y_train,LR.predict(train_BoW_standardised)),Y
_train,'Train Data')
print('****************Test****************')
confusion_matrix_display(confusion_matrix(Y_test,LR.predict(test_BoW_standardised)),Y_t
est,'Test Data')
```

```
**********Train**********

The TPR is :  0.9033427082853059
The TNR is :  0.8806437192668752
The FPR is :  0.11935628073312472
The FNR is :  0.09665729171469409
```

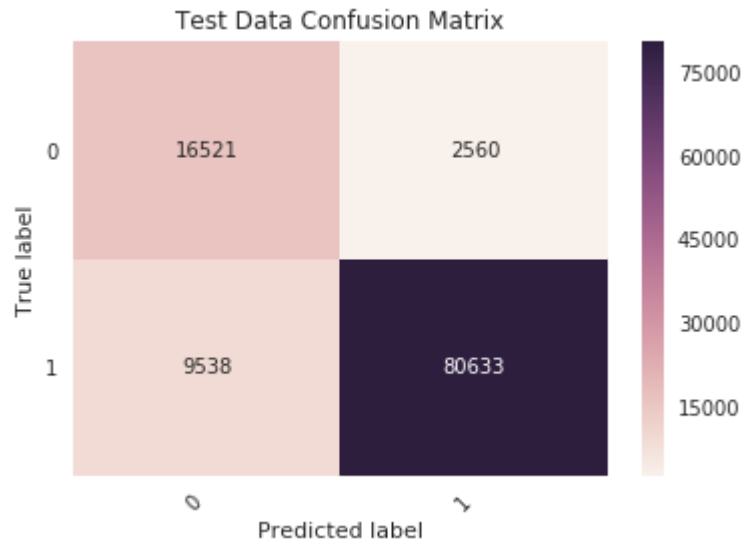
Out[0]:




```
**********Test**********

The TPR is :  0.8942231981457454
The TNR is :  0.8658351239452858
The FPR is :  0.1341648760547141
The FNR is :  0.1057768018542547
```

Out[0]:



Perturbation Test

Before Perturbation

In [0]:

```
LR = LogisticRegression(penalty = 'l1',C = 0.001, class_weight='balanced',n_jobs = -1,r
andom_state = 42)
LR.fit(train_BoW_standardised,Y_train)
before_pert = LR.coef_
```

After Perturbation

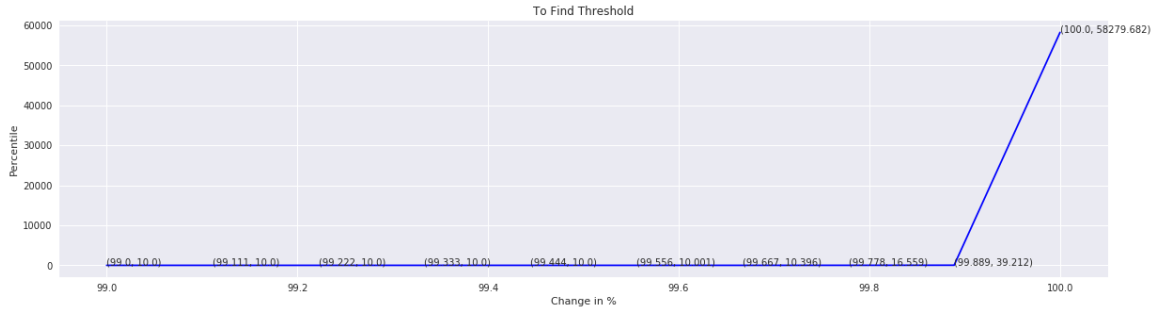
In [0]:

```
epsilon = np.random.normal(loc=0.0, scale=0.001)
train_BoW_standardised.data = train_BoW_standardised.data + epsilon
LR.fit(train_BoW_standardised,Y_train)
after_pert = LR.coef_
```

In [0]:

```
result = ( np.absolute(((before_pert - after_pert)+0.0001) / (before_pert+0.001))*100)
l = np.linspace(99, 100, num=10)
vals = []
for i in l:
  vals.append(np.percentile(result,i))
plt.figure(figsize=(20, 5))
plt.plot(l, vals, c='blue')
plt.xlabel('Change in %')
plt.ylabel('Percentile')
for xy in zip(np.round(l,3), np.round(vals,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
plt.title('To Find Threshold')
plt.show()
```

Out[0]:



In [0]:

```
print('The number of features greater than the threshold value of 16.559 is : ', len(np
.where(result > 16.559)[0]))
```

```
The number of features greater than the threshold value of 16.559 is :  58
34
```

WordCloud Of Features

WordCloud Of Multi Collinear Features

In [0]:

```
indices = np.where(result > 16.559)[1].tolist()
dict_boW = BoW_dict_bigram.get_feature_names()
multicolinear_features = [dict_boW[i] for i in indices]
Wordcl('BoW Multi-Collinear Features',multicolinear_features)
```

Out[0]:

WordCloud Of Top 200 Negative Features

In [0]:

```
sorted_index = np.argsort(before_pert)[::-1]
top_20_negative = sorted_index[0][0:201].tolist()
top_20_positive = sorted_index[0][-200:].tolist()
neg = [dict_boW[i] for i in top_20_negative]
pos = [dict_boW[i] for i in top_20_positive]
Wordcl('Top 200 Negative Features',neg)
```

Out[0]:

WordCloud Of Top 200 Positive Features

In [0]:

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

Out[0]:

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)
```

In [0]:

```
from sklearn.preprocessing import StandardScaler
standardised = StandardScaler(with_mean=False).fit(TFIDF_train)
train_TFIDF_standardised = standardised.transform(TFIDF_train)
test_TFIDF_standardised = standardised.transform(TFIDF_test)
```

In [0]:

```
%env JOBLIB_TEMP_FOLDER=/tmp
```

```
env: JOBLIB_TEMP_FOLDER=/tmp
```

GridSearchCV

In [0]:

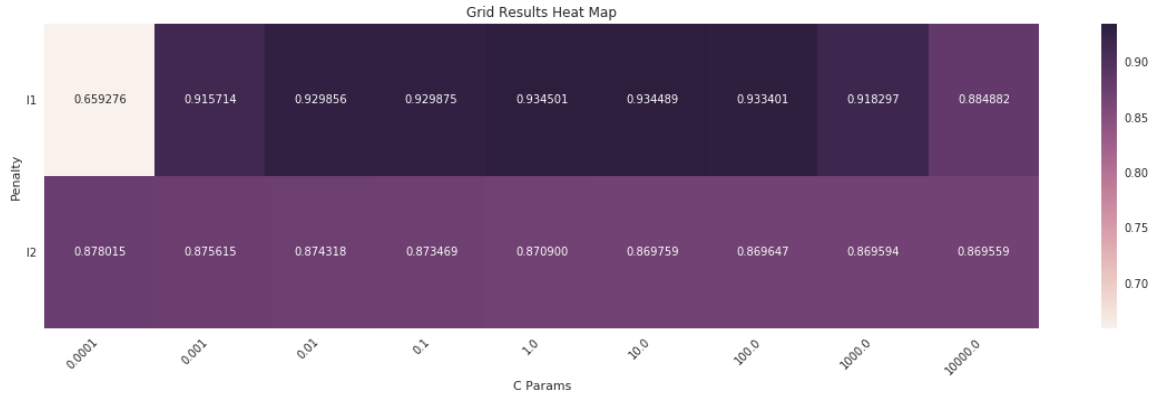
```
scores = searchLR('GS',params,train_TFIDF_standardised,Y_train,tss)
```

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

[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  5.4min
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed: 52.9min finished

The parameters that would gives best roc_auc is :  {'C': 1, 'penalty': 'l
1'}
The best roc_auc achieved after parameter tuning via grid search is :  0.9
345008590007411
```

Out[0]:



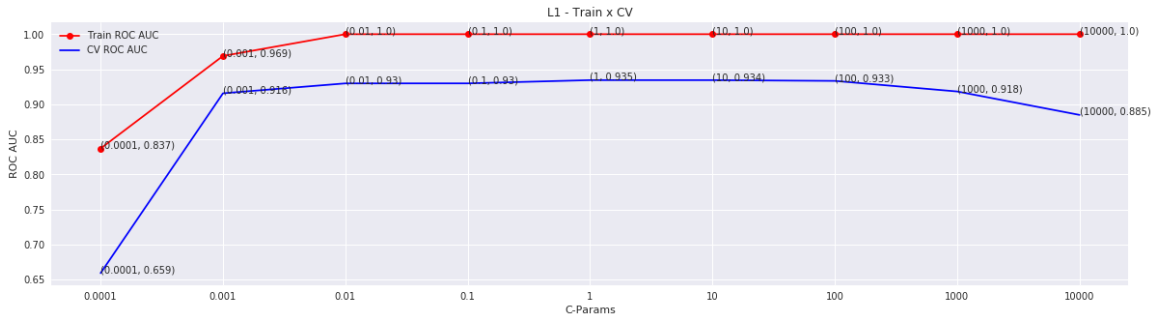
Validation Curve

In [0]:

```
from sklearn.metrics import roc_auc_score
l1_cv_scores = scores[0]
l1_train_scores = []
for i in C_params:
  LR = LogisticRegression(penalty = 'l1',C = i, class_weight='balanced',n_jobs = -1,ran
dom_state = 42)
  LR.fit(train_TFIDF_standardised,Y_train)
  l1_train_scores.append(roc_auc_score(Y_train,LR.predict_proba(train_TFIDF_standardise
d)[:,1]))

x = [float_to_str(i) for i in C_params]
validation_curve(x,l1_train_scores,l1_cv_scores,'L1 - Train x CV')
```

Out[0]:

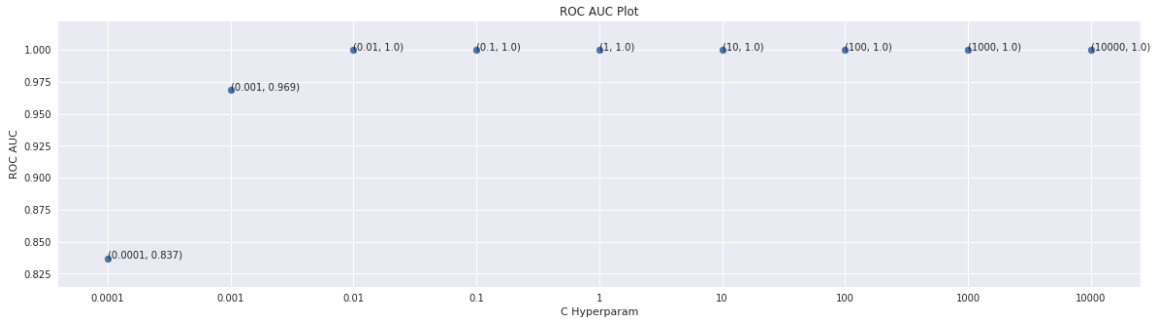


L1 Sparsity & Error

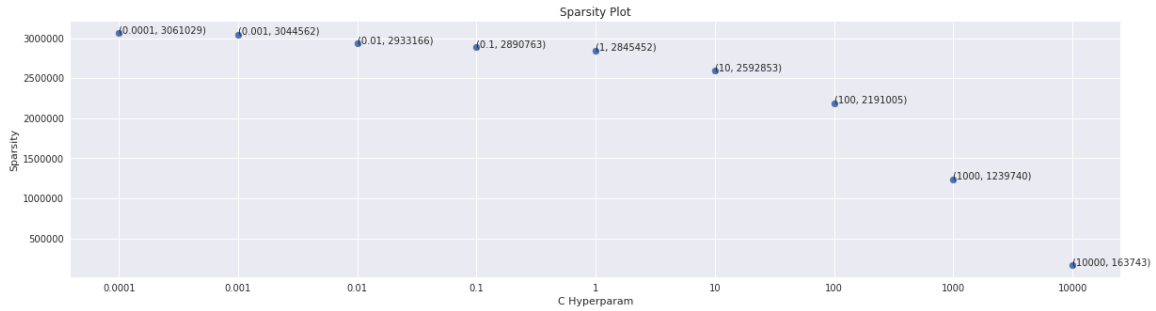
In [0]:

```
l1(C_params,train_TFIDF_standardised,Y_train)
```

Out[0]:



Out[0]:



On Test Data

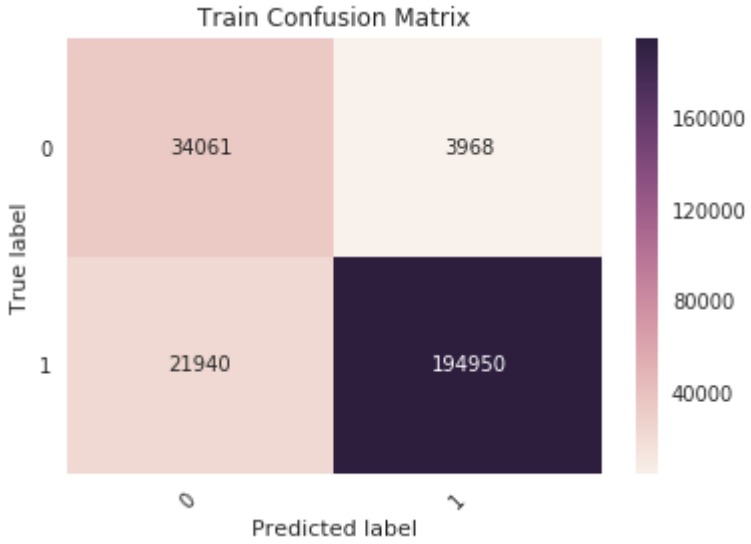
In [0]:

```
LR = LogisticRegression(penalty = 'l1',C = 0.001, class_weight='balanced',n_jobs = -1,r
andom_state = 42)
LR.fit(train_TFIDF_standardised,Y_train)
print('****************Train****************')
confusion_matrix_display(confusion_matrix(Y_train,LR.predict(train_TFIDF_standardised
)),Y_train,'Train')
print('****************Test****************')
confusion_matrix_display(confusion_matrix(Y_test,LR.predict(test_TFIDF_standardised)),Y
_test,'Test')
```

```
**********Train**********

The TPR is :  0.8988427313384665
The TNR is :  0.8956585763496279
The FPR is :  0.10434142365037208
The FNR is :  0.1011572686615335
```

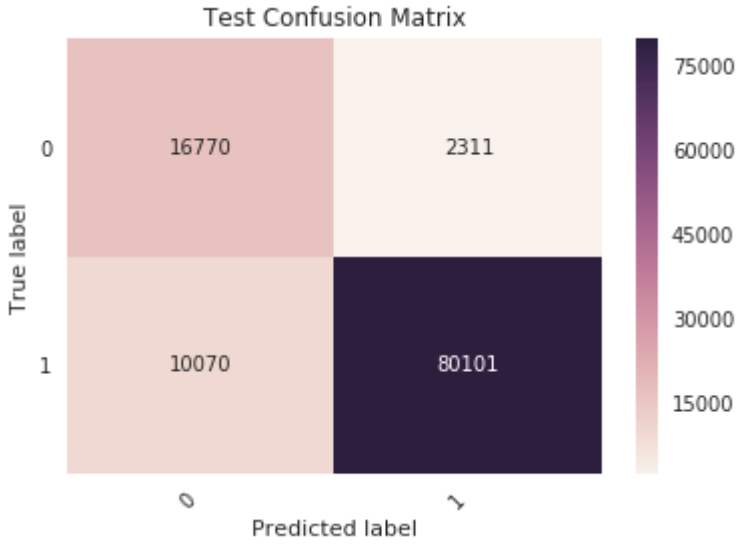
Out[0]:




```
**********Test**********

The TPR is :  0.8883232968471017
The TNR is :  0.8788847544677952
The FPR is :  0.1211152455322048
The FNR is :  0.11167670315289838
```

Out[0]:



Choosing the one with the lowest FPR that is C = 0.01.

Pertubation Test

Before Pertubation

In [0]:

```
LR = LogisticRegression(penalty = 'l1',C = 0.001, class_weight='balanced',n_jobs = -1,r
andom_state = 42)
LR.fit(train_TFIDF_standardised,Y_train)
before_pert = LR.coef_
```

After Pertubation

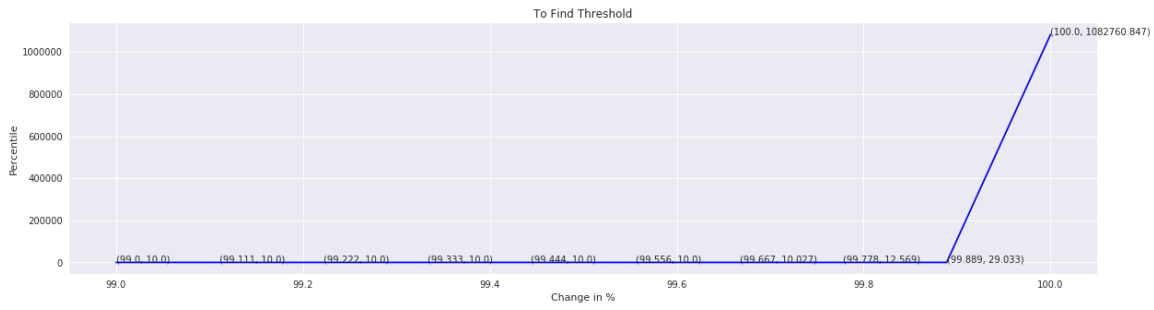
In [0]:

```
epsilon = np.random.normal(loc=0.0, scale=0.001)
train_TFIDF_standardised.data = train_TFIDF_standardised.data + epsilon
LR.fit(train_TFIDF_standardised,Y_train)
after_pert = LR.coef_
```

In [0]:

```
result = ( np.absolute(((before_pert - after_pert)+0.0001) / (before_pert+0.001))*100)
l = np.linspace(99,100,10)
vals = []
for i in l:
  vals.append(np.percentile(result,i))
plt.figure(figsize=(20, 5))
plt.plot(l, vals, c='blue')
plt.xlabel('Change in %')
plt.ylabel('Percentile')
for xy in zip(np.round(l,3), np.round(vals,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
plt.title('To Find Threshold')
plt.show()
```

Out[0]:



In [0]:

```
print('The number of features greater than the threshold value of 12.569 is : ', len(np
.where(result > 12.569)[0]))
```

```
The number of features greater than the threshold value of 12.569 is :  68
03
```

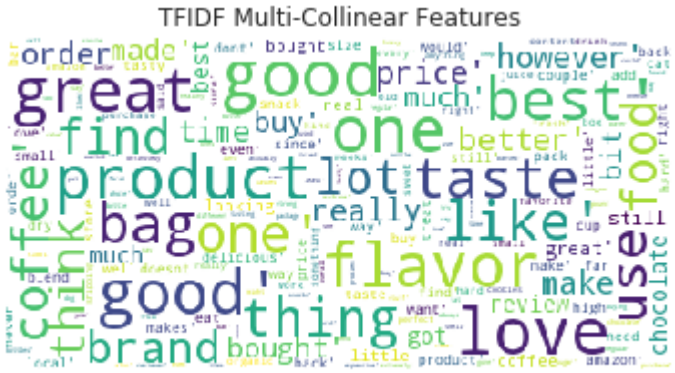
WordClouds Of Features

WordClouds Of Multi Collinear Features

In [0]:

```
indices = np.where(result > 12.569)[1].tolist()
dict_tFidF = TFIDF_dict_bigram.get_feature_names()
multicolinear_features = [dict_tFidF[i] for i in indices]
Wordcl('TFIDF Multi-Collinear Features',multicolinear_features)
```

Out[0]:



WordClouds Of Top 200 Negative Features

In [0]:

```
sorted_index = np.argsort(before_pert)[::-1]
top_20_negative = sorted_index[0][0:201].tolist()
top_20_positive = sorted_index[0][-200:].tolist()
neg = [dict_tFidF[i] for i in top_20_negative]
pos = [dict_tFidF[i] for i in top_20_positive]
Wordcl('Top 200 Negative Features',neg)
```

Out[0]:

WordClouds Of Top 200 Positive Features

In [0]:

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

Out[0]:

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
import numpy as np

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)
```

GridSearchCV

In [0]:

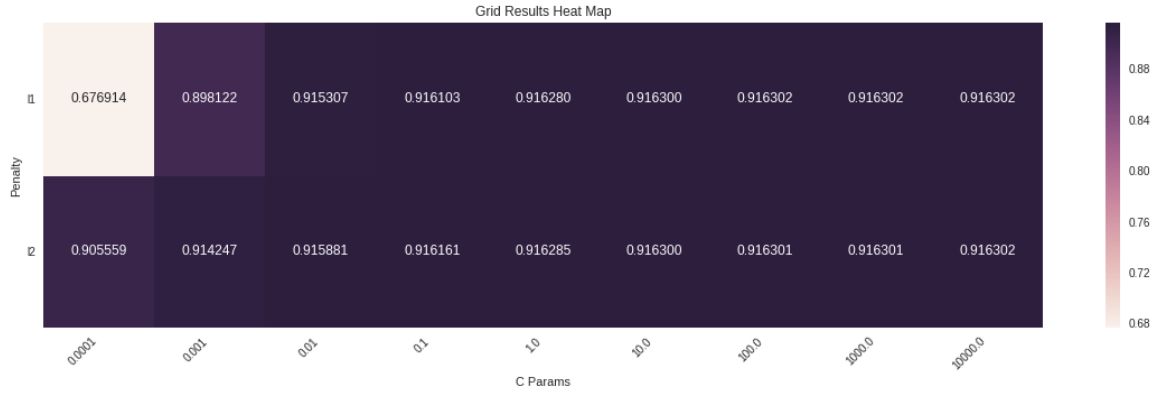
```
scores = searchLR('GS',params,sent_vectors_train,Y_train,tss)
```

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

[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed: 103.5min finished

The parameters that would gives best roc_auc is :  {'C': 10000, 'penalty':
'l1'}
The best roc_auc achieved after parameter tuning via grid search is :  0.9
163019023444949
```

Out[0]:



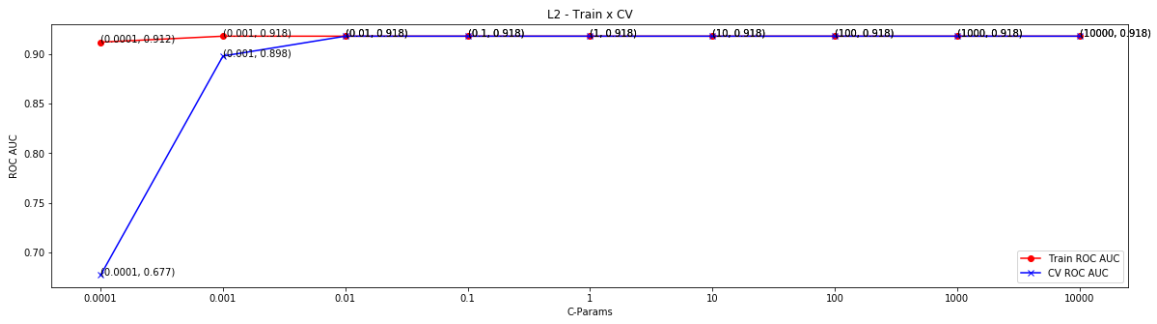
Validation Plot

In [0]:

```
from sklearn.metrics import roc_auc_score
l1_cv_scores = scores[0]
l1_train_scores = []
for i in C_params:
  LR = LogisticRegression(penalty = 'l2',C = i, class_weight='balanced',n_jobs = -1,ran
dom_state = 42)
  LR.fit(sent_vectors_train,Y_train)
  l1_train_scores.append(roc_auc_score(Y_train,LR.predict_proba(sent_vectors_train)[:,1
]))

x = [float_to_str(i) for i in C_params]
validation_curve(x,l1_train_scores,l1_cv_scores,'L2 - Train x CV')
```

Out[0]:

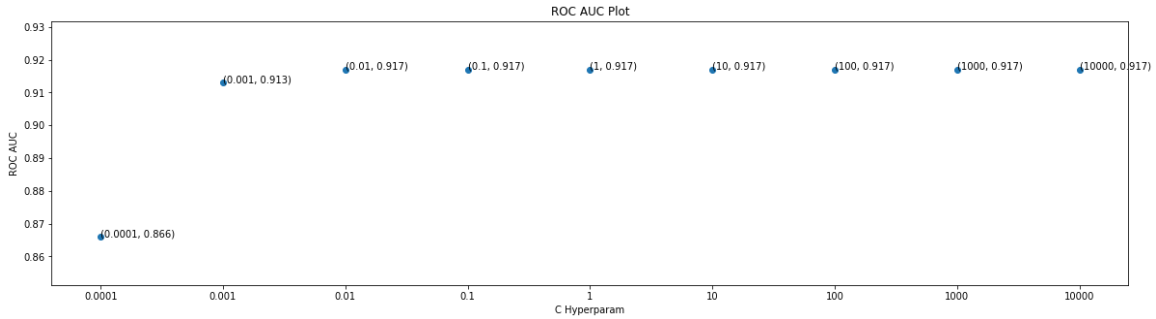


L1 Sparsity & Error

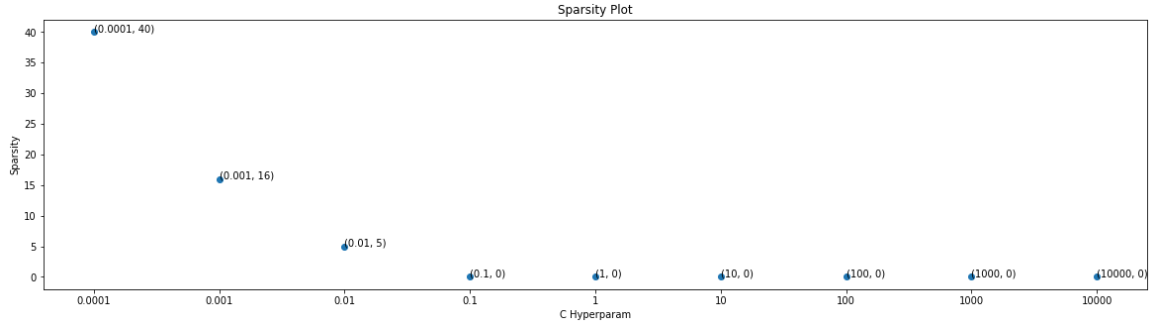
In [0]:

```
l1(C_params,sent_vectors_train,Y_train)
```

Out[0]:



Out[0]:



On Test Data

In [0]:

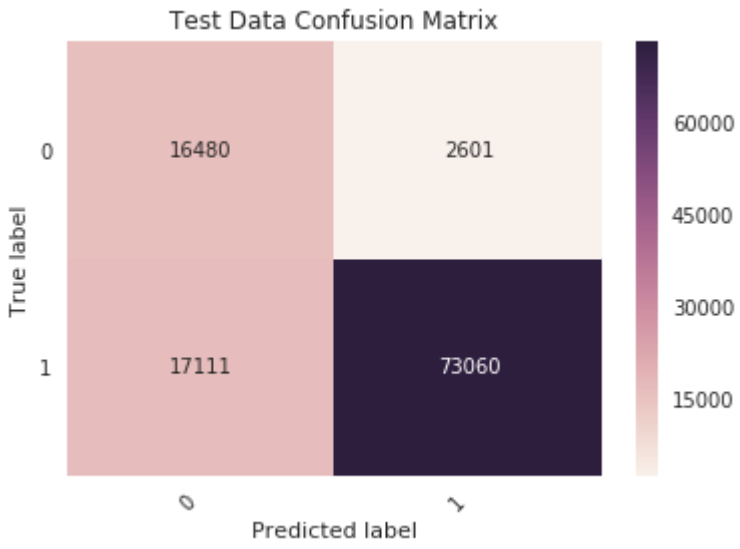
```
sent_vectors_test = []
for sent in X_test.values:
    sent_vec = np.zeros(50)
    cnt_words =0
    for word in sent.split():
        try:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass
    sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)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

In [0]:

```
LR = LogisticRegression(penalty = 'l2',C = 0.01, class_weight='balanced',n_jobs = -1,ra
ndom_state = 42)
LR.fit(sent_vectors_train,Y_train)
print('****************Train*****************')
confusion_matrix_display(confusion_matrix(Y_train,LR.predict(sent_vectors_train)),Y_tra
in,'Train Data')
print('****************Test*****************')
confusion_matrix_display(confusion_matrix(Y_test,LR.predict(sent_vectors_test)),Y_test,
'Test Data')
```

```
****************Train*****************
The TPR is :  0.8249988473419706
The TNR is :  0.8546109547976545
The FPR is :  0.14538904520234558
The FNR is :  0.17500115265802943
****************Test*****************
The TPR is :  0.8102383249603531
The TNR is :  0.8636863896022221
The FPR is :  0.1363136103977779
The FNR is :  0.1897616750396469
```

Out[0]:



Choosing the one with the lowest FPR l2 C = 1

TFIDF W2V

In [0]:

```
i=0
list_of_sent=[]
for sent in X_train.values:
    filtered_sentence=[]
    sent=cleanhtml(sent)
    str1 = ''
    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]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_dict = TfidfVectorizer()
tfx = tfidf_dict.fit_transform(X_train)
tfidf_feat = tfidf_dict.get_feature_names()
```

In [0]:

```
tfidf_sent_vectors_train = [];
row=0;
for sent in list_of_sent:
   sent_vec = np.zeros(50)
   weight_sum =0;
   for word in sent:
      try:
         vec = w2v_model.wv[word]
         tf_idf = tfx[row, tfidf_feat.index(word)]
         sent_vec += (vec * tf_idf)
         weight_sum += tf_idf
      except:
         pass
   try:
        sent_vec /= weight_sum
   except:
        pass

   tfidf_sent_vectors_train.append(sent_vec)
   row += 1
tfidf_sent_vectors_train = np.nan_to_num(tfidf_sent_vectors_train)
```

GridSearchCV

In [0]:

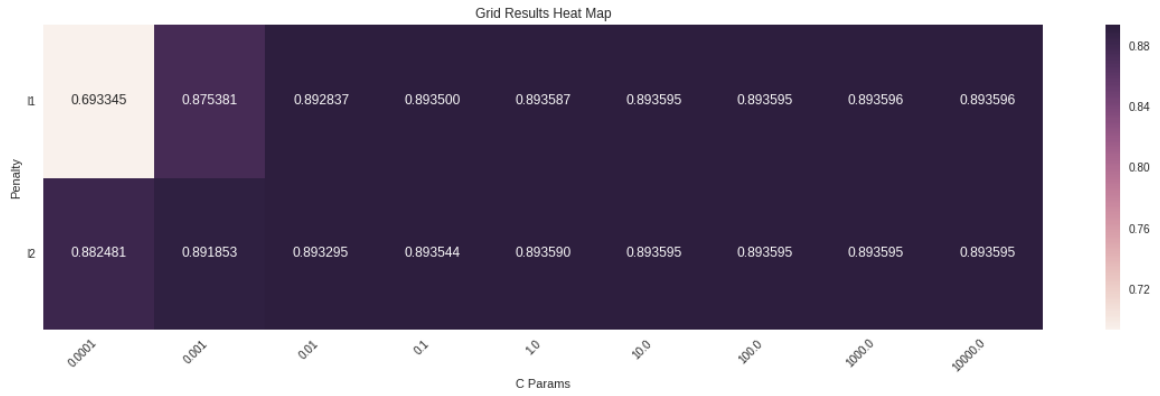
```
scores = searchLR('GS',params,tfidf_sent_vectors_train,Y_train,tss)
```

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

[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed: 53.3min finished

The parameters that would gives best roc_auc is :  {'C': 1000, 'penalty':
'l1'}
The best roc_auc achieved after parameter tuning via grid search is :  0.8
935955946698522
```

Out[0]:



Out[0]:

```
array([[0.69334506, 0.87538066, 0.89283673, 0.89349993, 0.89358713,
        0.89359492, 0.89359549, 0.89359559, 0.89359554],
       [0.88248114, 0.89185342, 0.89329538, 0.89354395, 0.89359021,
        0.89359459, 0.89359505, 0.8935949 , 0.89359488]])
```

In [0]:

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

In [0]:

```
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()
```

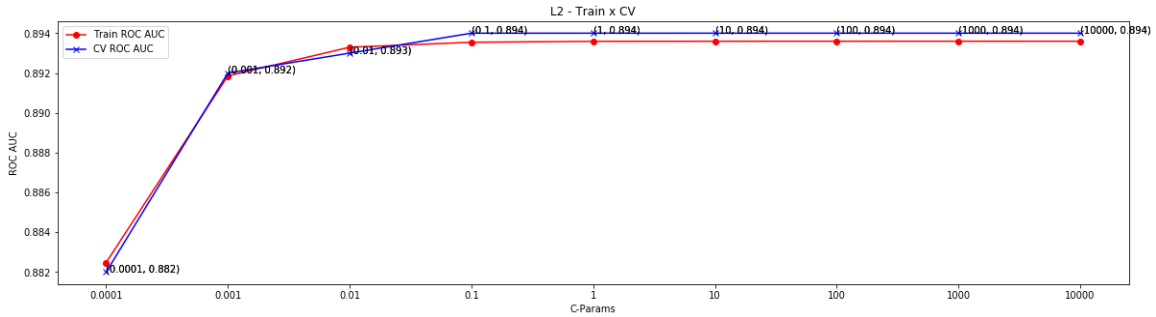
Validation Curve

In [0]:

```
l2_cv_scores = scores[1]
l2_train_scores = []
for i in C_params:
  LR = LogisticRegression(penalty = 'l2',C = i, class_weight='balanced',n_jobs = -1,ran
dom_state = 42)
  LR.fit(tfidf_sent_vectors_train,Y_train)
  l2_train_scores.append(roc_auc_score(Y_train,LR.predict_proba(tfidf_sent_vectors_trai
n)[:,1]))

x = [float_to_str(i) for i in C_params]
validation_curve(x,l2_train_scores,l2_cv_scores,'L2 - Train x CV')
```

Out[0]:

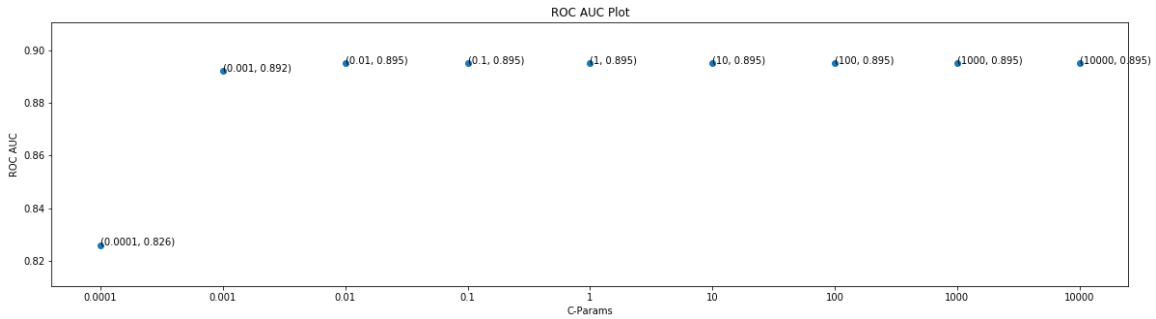


L1 Sparsity & Error

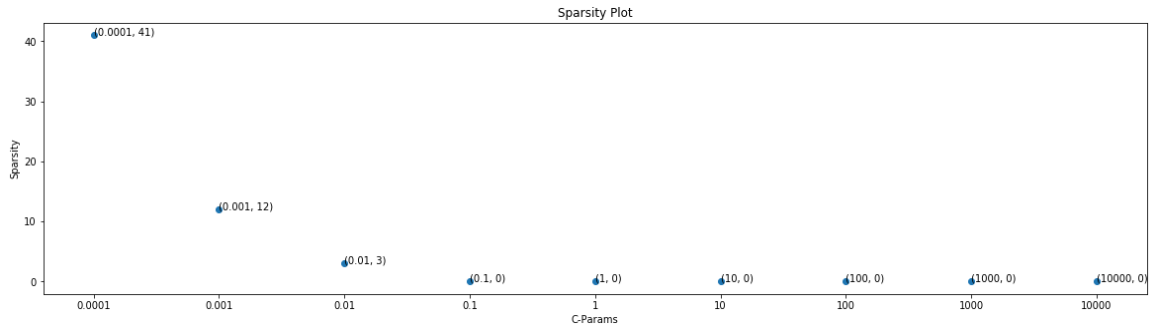
In [0]:

```
l1(C_params,tfidf_sent_vectors_train,Y_train)
```

Out[0]:



Out[0]:



On Test Data

In [0]:

```
tfidf_sent_vectors_test = np.nan_to_num(tfidf_sent_vectors_test)
```

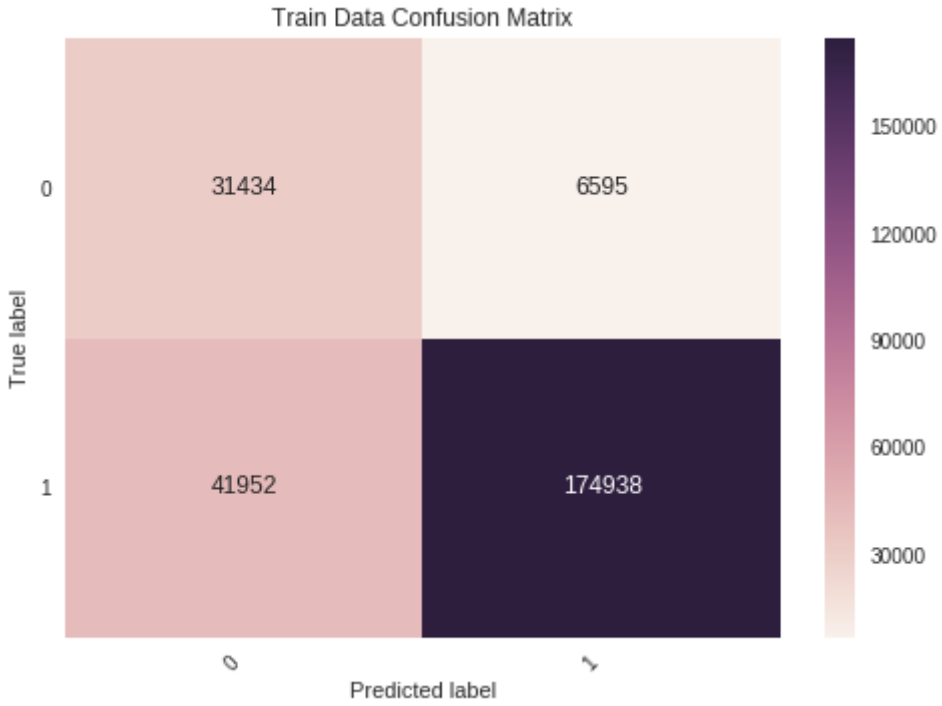
In [0]:

```
LR = LogisticRegression(penalty = 'l2',C = 0.1, class_weight='balanced',n_jobs = -1,ran
dom_state = 42)
LR.fit(tfidf_sent_vectors_train,Y_train)
print('****************Train****************')
confusion_matrix_display(confusion_matrix(Y_train,LR.predict(tfidf_sent_vectors_train
)),Y_train,'Train Data')
print('****************Test*****************')
confusion_matrix_display(confusion_matrix(Y_test,LR.predict(tfidf_sent_vectors_test)),Y
_test,'Test Data')
```

```
***************Train*****************

The TPR is :  0.8065747613997879
The TNR is :  0.8265797154802914
The FPR is :  0.17342028451970865
The FNR is :  0.1934252386002121
```

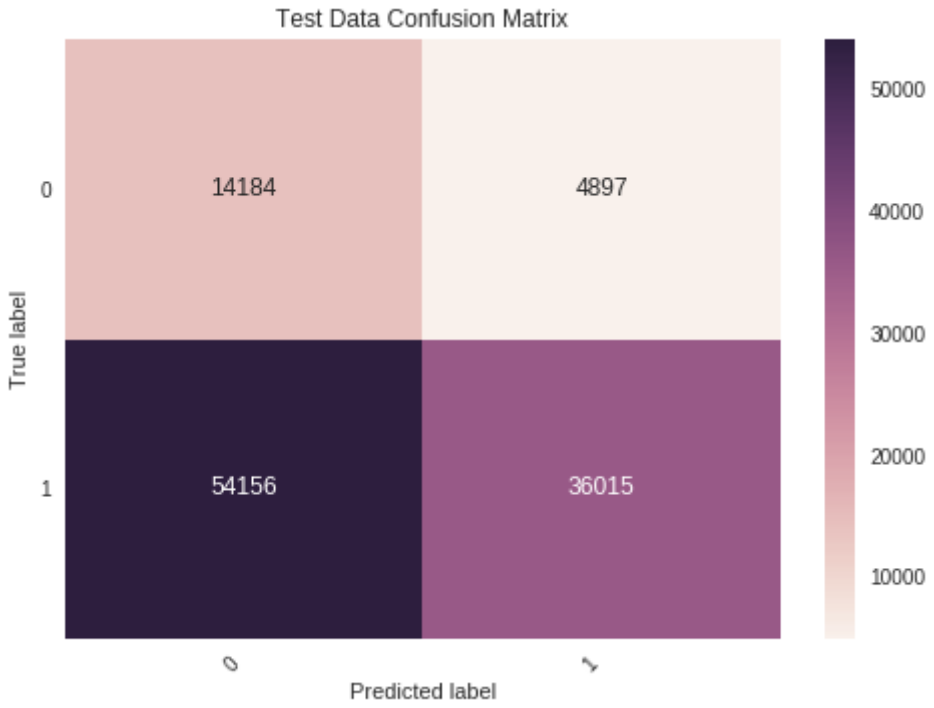
Out[0]:




```
***************Test*****************

The TPR is :  0.3994077918621286
The TNR is :  0.7433572663906504
The FPR is :  0.25664273360934964
The FNR is :  0.6005922081378714
```

Out[0]:



Conclusion

Model	Reg - C	Train FPR	Test FPR	Train FNR	Test FNR
Bag Of Words	L1 - 0.001	0.119	0.134	0.097	0.111
TFIDF	L1 - 0.001	0.104	0.121	0.101	0.111
Avg W2V	L2 - 0.1	0.145	0.136	0.175	0.190
TF-IDF W2V	L2 - 0.1	0.173	0.257	0.193	0.601

The best is : TFIDF with L1 regulariser with C is 0.001

In [0]: