# 05 Amazon Fine Food Reviews Analysis_Logistic Regression

May 7, 2019

## 1 Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454 Number of users: 256,059 Number of products: 74,258 Timespan: Oct 1999 - Oct 2012 Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

**Objective:** Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative? [Ans] We could use Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

## 2 [1]. Reading Data

### 2.1 [1.1] Loading the data

The dataset is available in two forms 1. .csv file 2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [5]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

In [6]:
```
# using SQLite Table to read data.
con = sqlite3.connect(r'D:\Sayantan\Personal\MLnAI\Assignments\LogisticRegression\datal

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data point
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 5
```

```
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 1000

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negativ
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (100000, 10)
```

```
Out[6]:    Id   ProductId          UserId                      ProfileName  \
        0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian
        1   2  B00813GRG4  A1D87F6ZCVE5NK                           dll pa
        2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"

           HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
        0                     1                       1      1  1303862400
        1                     0                       0      0  1346976000
        2                     1                       1      1  1219017600

                         Summary                                               Text
        0  Good Quality Dog Food  I have bought several of the Vitality canned d...
        1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
        2  "Delight" says it all  This is a confection that has been around a fe...
```

```
In [7]: display = pd.read_sql_query("""
        SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
        FROM Reviews
        GROUP BY UserId
        HAVING COUNT(*)>1
        """, con)
```

```
In [8]: print(display.shape)
        display.head()
```

```
(80668, 7)
```

```
Out[8]:               UserId   ProductId             ProfileName        Time  Score  \
        0  #oc-R115TNMSPFT9I7  B005ZBZLT4                 Breyton  1331510400      2
```

```
1  #oc-R11D9D7SHXIJB9  B005HG9ESG  Louis E. Emory "hoppy"  1342396800      5
2  #oc-R11DNU2NBKQ23Z  B005ZBZLT4        Kim Cieszykowski  1348531200      1
3  #oc-R1105J5ZVQE25C  B005HG9ESG           Penguin Chick  1346889600      5
4  #oc-R12KPBODL2B5ZD  B0070SBEV0   Christopher P. Presta  1348617600      1

                                                Text  COUNT(*)
0  Overall its just OK when considering the price...         2
1  My wife has recurring extreme muscle spasms, u...         3
2  This coffee is horrible and unfortunately not ...         2
3  This will be the bottle that you grab from the...         3
4  I didnt like this coffee. Instead of telling y...         2
```

In [9]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[9]:
```
              UserId   ProductId                     ProfileName        Time  \
80638  AZY10LLTJ71NX  B001ATMQK2  undertheshrine "undertheshrine"  1296691200

       Score                                               Text  COUNT(*)
80638      5  I bought this 6 pack because for the price tha...         5
```

In [10]:
```
display['COUNT(*)'].sum()
```

Out[10]:
```
393063
```

# 3 [2] Exploratory Data Analysis

## 3.1 [2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [11]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[11]:
```
       Id   ProductId         UserId      ProfileName  HelpfulnessNumerator  \
0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2
1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2
2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2
3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                     2
4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                     2

   HelpfulnessDenominator  Score        Time  \
0                       2      5  1199577600
```

```
1                           2       5  1199577600
2                           2       5  1199577600
3                           2       5  1199577600
4                           2       5  1199577600

                                  Summary  \
0  LOACKER QUADRATINI VANILLA WAFERS
1  LOACKER QUADRATINI VANILLA WAFERS
2  LOACKER QUADRATINI VANILLA WAFERS
3  LOACKER QUADRATINI VANILLA WAFERS
4  LOACKER QUADRATINI VANILLA WAFERS

                                                Text
0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8) ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [12]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fa
```

In [13]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text","Summa
final.shape
```

Out[13]: (87898, 10)

In [14]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[14]: 87.898

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [15]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[15]:
```
      Id   ProductId          UserId              ProfileName  \
0  64422  B000MIDROQ  A161DK06JJMCYF  J. E. Stephens "Jeanne"
1  44737  B001EQ55RW  A2V0I904FH7ABY                      Ram

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     3                       1      5  1224892800
1                     3                       2      4  1212883200

                                      Summary  \
0             Bought This for My Son at College
1  Pure cocoa taste with crunchy almonds inside

                                               Text
0  My son loves spaghetti so I didn't hesitate or...
1  It was almost a 'love at first bite' - the per...
```

In [16]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [17]:
```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

```
(87896, 10)
```

Out[17]:
```
1    73686
0    14210
Name: Score, dtype: int64
```

# 4 [3] Preprocessing

## 4.1 [3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags

2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [18]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
Our dog loves these treats, and since there are only 2 calories per treat, you don't have to wo
==================================================
```

In [19]:
```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
```

In [20]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-
from bs4 import BeautifulSoup
```

```
soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
Our dog loves these treats, and since there are only 2 calories per treat, you don't have to wo
```

In [21]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [22]:
```
sent_1500 = decontracted(sent_1500)
```

```
print(sent_1500)
print("="*50)
```

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

In [23]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
```

In [24]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
was way to hot for my blood took a bite and did a jig lol
```

In [25]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselve
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him'
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', '
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throug
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', '
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'a
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'to
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", '
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mi
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
            'won', "won't", 'wouldn', "wouldn't"])
```

In [26]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
```

```
            sentance = BeautifulSoup(sentance, 'lxml').get_text()
            sentance = decontracted(sentance)
            sentance = re.sub("\S*\d\S*", "", sentance).strip()
            sentance = re.sub('[^A-Za-z]+', ' ', sentance)
            # https://gist.github.com/sebleier/554280
            sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopw
            preprocessed_reviews.append(sentance.strip())
```

```
100%|| 87896/87896 [00:45<00:00, 1915.19it/s]
```

```
In [27]: preprocessed_reviews[1500]
```

```
Out[27]: 'way hot blood took bite jig lol'
```

[3.2] Preprocessing Review Summary

```
In [28]: #Deduplication of entries
         import warnings
         warnings.filterwarnings("ignore")

         print(final.shape)
         #Checking to see how much % of data still remains
         print((final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100)

         #How many positive and negative reviews are present in our dataset?
         print(final['Score'].value_counts())

         from tqdm import tqdm
         preprocessed_summary = []
         # tqdm is for printing the status bar
         for sentance in tqdm(final['Summary'].values):
             sentance = re.sub(r"http\S+", "", sentance)
             sentance = BeautifulSoup(sentance, 'lxml').get_text()
             sentance = decontracted(sentance)
             sentance = re.sub("\S*\d\S*", "", sentance).strip()
             sentance = re.sub('[^A-Za-z]+', ' ', sentance)
             # https://gist.github.com/sebleier/554280
             sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopw
             preprocessed_summary.append(sentance.strip())
```

```
(87896, 10)
87.896
1    73686
0    14210
Name: Score, dtype: int64


100%|| 87896/87896 [00:35<00:00, 2473.86it/s]
```

In [29]: preprocessed_summary[1500]

Out[29]: 'hot stuff'

# 5 [4] Featurization

## 5.1 [4.1] BAG OF WORDS

In [27]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
print("the number of unique words ", final_counts.get_shape()[1])
```

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaaaaaaaa', 'aaaaaaaaaaaaaa', 'aaaaaaa
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (87896, 54904)
the number of unique words  54904
```

## 5.2 [4.2] Bi-Grams and n-Grams.

In [28]:
```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.org/stable/mod

# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (87896, 5000)
the number of unique words including both unigrams and bigrams  5000
```

## 5.3 [4.3] TF-IDF

In [29]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf
```

```
some sample features(unique words in the corpus) ['aa', 'aafco', 'aback', 'abandon', 'abandoned
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (87896, 51811)
the number of unique words including both unigrams and bigrams  51811
```

## 5.4 [4.4] Word2Vec

In [30]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [31]:
```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
```

```
            w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
            print(w2v_model.wv.most_similar('great'))
            print('='*50)
            print(w2v_model.wv.most_similar('worst'))

        elif want_to_use_google_w2v and is_your_ram_gt_16g:
            if os.path.isfile('GoogleNews-vectors-negative300.bin'):
                w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bi
                print(w2v_model.wv.most_similar('great'))
                print(w2v_model.wv.most_similar('worst'))
            else:
                print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, t
```

```
[('fantastic', 0.8487210273742676), ('awesome', 0.8324342966079712), ('excellent', 0.810306847
==================================================
[('greatest', 0.793619692325592), ('best', 0.7024627327919006), ('tastiest', 0.702161371707916
```

```
In [32]: w2v_words = list(w2v_model.wv.vocab)
         print("number of words that occured minimum 5 times ",len(w2v_words))
         print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  17404
sample words  ['dogs', 'loves', 'chicken', 'product', 'china', 'wont', 'buying', 'anymore', 'ha
```

## 5.5 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

### [4.4.1.1] Avg W2v

```
In [33]: # average Word2Vec
         # compute average word2vec for each review.
         sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
         for sent in tqdm(list_of_sentance): # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need t
             cnt_words =0; # num of words with a valid vector in the sentence/review
             for word in sent: # for each word in a review/sentence
                 if word in w2v_words:
                     vec = w2v_model.wv[word]
                     sent_vec += vec
                     cnt_words += 1
             if cnt_words != 0:
                 sent_vec /= cnt_words
             sent_vectors.append(sent_vec)
         print(len(sent_vectors))
         print(len(sent_vectors[0]))
```

```
100%|| 87896/87896 [06:02<00:00, 242.41it/s]
```

```
87896
50
```

#### [4.4.1.2] TFIDF weighted W2v

```
In [34]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
         model = TfidfVectorizer()
         tf_idf_matrix = model.fit_transform(preprocessed_reviews)
         # we are converting a dictionary with word as a key, and the idf as a value
         dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [35]: # TF-IDF weighted Word2Vec
         tfidf_feat = model.get_feature_names() # tfidf words/col-names
         # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

         tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this l
         row=0;
         for sent in tqdm(list_of_sentance): # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length
             weight_sum =0; # num of words with a valid vector in the sentence/review
             for word in sent: # for each word in a review/sentence
                 if word in w2v_words and word in tfidf_feat:
                     vec = w2v_model.wv[word]
         #            tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                     # to reduce the computation we are
                     # dictionary[word] = idf value of word in whole courpus
                     # sent.count(word) = tf valeus of word in this review
                     tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                     sent_vec += (vec * tf_idf)
                     weight_sum += tf_idf
             if weight_sum != 0:
                 sent_vec /= weight_sum
             tfidf_sent_vectors.append(sent_vec)
             row += 1
```

```
 45%|                                         | 39523/87896 [37:04<45:17, 17.80it/s]


        ---------------------------------------------------------------------------

        KeyboardInterrupt                         Traceback (most recent call last)

        <ipython-input-35-0d742f22e115> in <module>
         16             # sent.count(word) = tf valeus of word in this review
         17             tf_idf = dictionary[word]*(sent.count(word)/len(sent))
    ---> 18             sent_vec += (vec * tf_idf)
         19             weight_sum += tf_idf
```

```
20     if weight_sum != 0:


KeyboardInterrupt:
```

# 6 [5] Assignment 5: Apply Logistic Regression

```
<li><strong>Apply Logistic Regression on these feature sets</strong>
    <ul>
        <li><font color='red'>SET 1:</font>Review text, preprocessed one converted into vector
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vector
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vector
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vector
    </ul>
</li>
<br>
<li><strong>Hyper paramter tuning (find best hyper parameters corresponding the algorithm that
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaico
<li>Find the best hyper paramter using k-fold cross validation or simple cross validation data
<li>Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this t
    </ul>
</li>
<br>
<li><strong>Pertubation Test</strong>
    <ul>
<li>Get the weights W after fit your model with the data X i.e Train data.</li>
<li>Add a noise to the X (X' = X + e) and get the new data set X' (if X is a sparse
```

matrix, X.data+=e)

```
<li>Fit the model again on data X' and get the weights W'</li>
<li>Add a small eps value(to eliminate the divisible by zero error) to W and W i.e
```

W=W+10^-6 and W′ = W′+10^-6

```
<li>Now find the % change between W and W' (| (W-W') / (W) |)*100)</li>
<li>Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden rise in
<li> Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there is su
        <li> Print the feature names whose % change is more than a threshold x(in our example
    </ul>
</li>
<br>
<li><strong>Sparsity</strong>
    <ul>
<li>Calculate sparsity on weight vector obtained after using L1 regularization</li>
    </ul>
```

```
</li>
<br><font color='red'>NOTE: Do sparsity and multicollinearity for any one of the vectorizers.
<br>
<br>
<li><strong>Feature importance</strong>
    <ul>
<li>Get top 10 important features for both positive and negative classes separately.</li>
    </ul>
</li>
<br>
<li><strong>Feature engineering</strong>
    <ul>
<li>To increase the performance of your model, you can also experiment with with feature engine
        <ul>
        <li>Taking length of reviews as another feature.</li>
        <li>Considering some features from review summary as well.</li>
    </ul>
    </ul>
</li>
<br>
<li><strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='train_cv_auc.JPG' width=300px></li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table fo
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

# 7 Applying Logistic Regression

```
In [30]: #total data length taken 100K, as instructed in the assigment and then it got prepros
         print("Data length after preprocessing: ",len(preprocessed_reviews))

         Y = final['Score'].values
         X = np.asarray(preprocessed_reviews)

         print(type(X))
         print(type(Y))

         #Splitting data in train, cv and test
         #error calc code taken from https://stackabuse.com/k-nearest-neighbors-algorithm-in-p
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import roc_auc_score

         X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20,shuffle=False
         X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.20,shuf
         print(X_train.shape, y_train.shape)
         print(X_cv.shape, y_cv.shape)
         print(X_test.shape, y_test.shape)
```

```
Data length after preprocessing:  87896
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
(56252,) (56252,)
(14064,) (14064,)
(17580,) (17580,)
```

```
In [38]: #Defination of the functions used for analysis in the following sections
         from sklearn.linear_model import LogisticRegression
         C_val = [0.00001, 0.001, 1, 100, 10000]
         C_idx = [1, 2, 3, 4, 5]

         def getOptimalLamda(X_train, y_train, X_cv, y_cv, regulizer):
             train_auc = []
             cv_auc = []
             for i in C_val:
                 logisticRegressor = LogisticRegression(penalty = regulizer, C = i)
                 logisticRegressor.fit(X_train, y_train)
                 y_train_pred =  logisticRegressor.predict_proba(X_train)[:,1]
                 y_cv_pred =  logisticRegressor.predict_proba(X_cv)[:,1]

                 train_auc.append(roc_auc_score(y_train,y_train_pred))
                 cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

             plt.plot(C_idx, train_auc, label='Train AUC')
```

```
    plt.scatter(C_idx, train_auc, label='Train AUC')
    plt.xticks(C_idx,C_val, rotation=45)
    plt.plot(C_idx, cv_auc, label='CV AUC')
    plt.scatter(C_idx, cv_auc, label='CV AUC')
    plt.legend()
    plt.xlabel("C: hyperparameter")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()

    return logisticRegressor
```

In [49]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

def getLRAnalysis(CVal, X_train, y_train, X_test, y_test, regulizer):
    logisticRegressor = LogisticRegression(penalty = regulizer, C = CVal)
    logisticRegressor.fit(X_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
    # not the predicted outputs

    train_fpr, train_tpr, thresholds = roc_curve(y_train, logisticRegressor.predict_pr
    test_fpr, test_tpr, thresholds = roc_curve(y_test, logisticRegressor.predict_proba

    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.xlabel("C: hyperparameter")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()

    train_conf_matix = confusion_matrix(y_train, logisticRegressor.predict(X_train))
    test_conf_matrix = confusion_matrix(y_test, logisticRegressor.predict(X_test))

    return(logisticRegressor, train_conf_matix, test_conf_matrix)
```

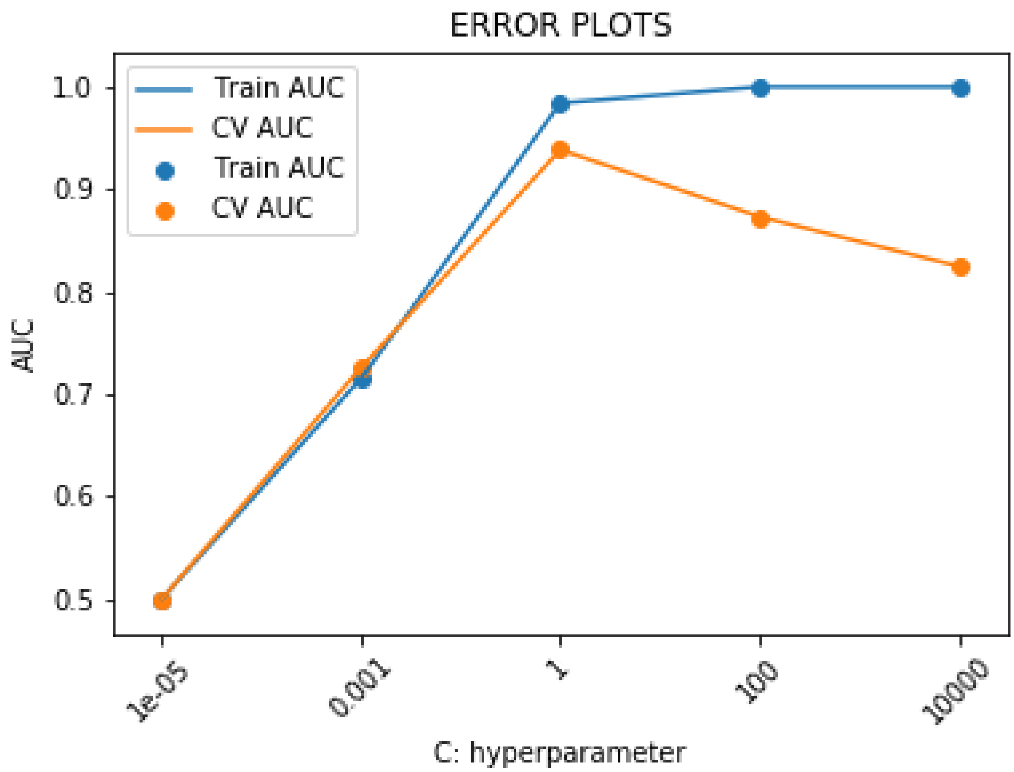
In [34]:
```
def showConfusionMatrix(confMatrix, titleText):
    print(confMatrix)
    df_train = pd.DataFrame(confMatrix, index=["-ve", "+ve"],columns=["-ve", "+ve"])
    sns.heatmap(df_train, annot=True, fmt='d')
    plt.title(titleText)
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()
```

## 7.1 [5.1] Logistic Regression on BOW, SET 1

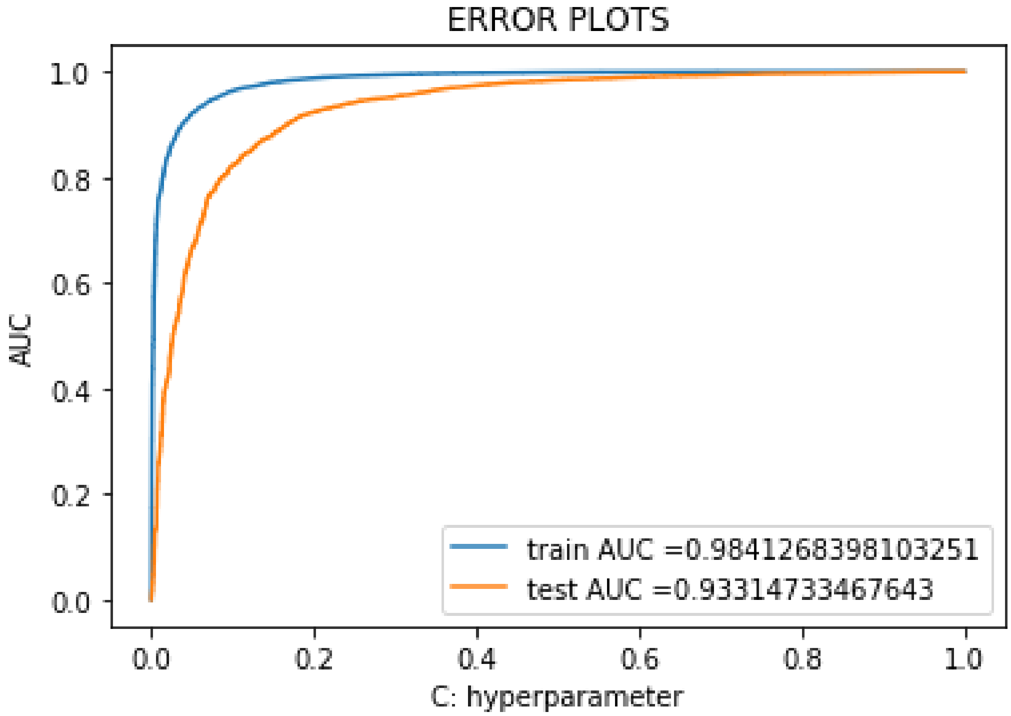
### 7.1.1 [5.1.1] Applying Logistic Regression with L1 regularization on BOW, SET 1

In [35]:
```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
#applying the method fit_transform() on you train data, and apply the method transfor
X_train_bow = vectorizer.fit_transform(X_train)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)
```

In [39]:
```
logRegressor = getOptimalLamda(X_train_bow, y_train, X_cv_bow, y_cv,'l1')
```

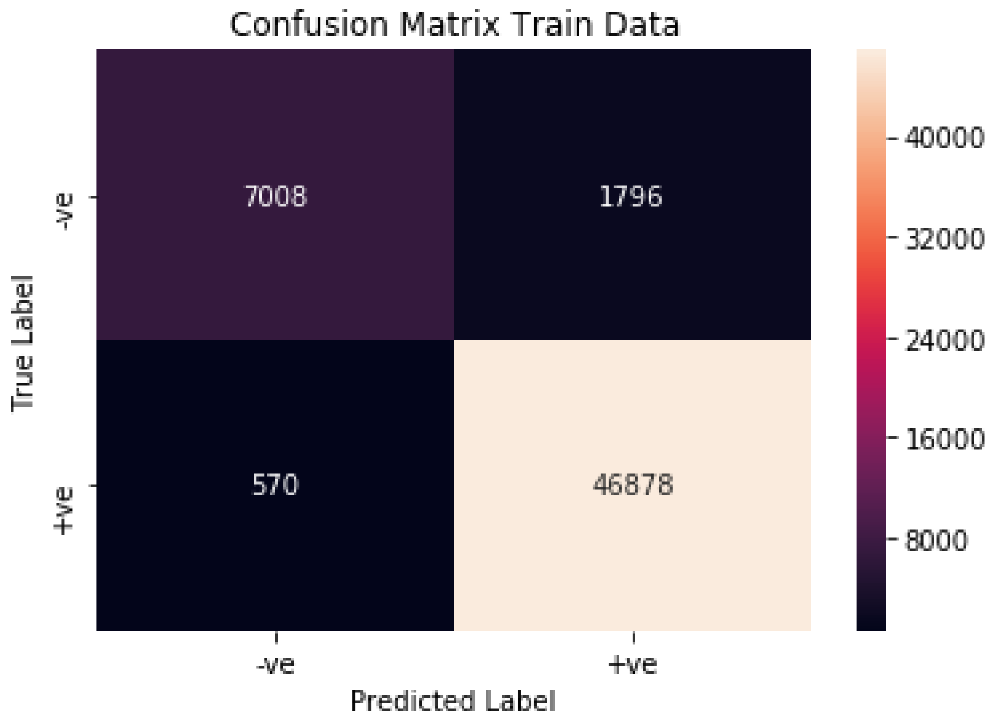



In [56]:
```
#Best value of AUC at C=1
bow_l1_logRegressor, train_confusion_matrix, test_confusion_matrix = getLRAnalysis(1,
```



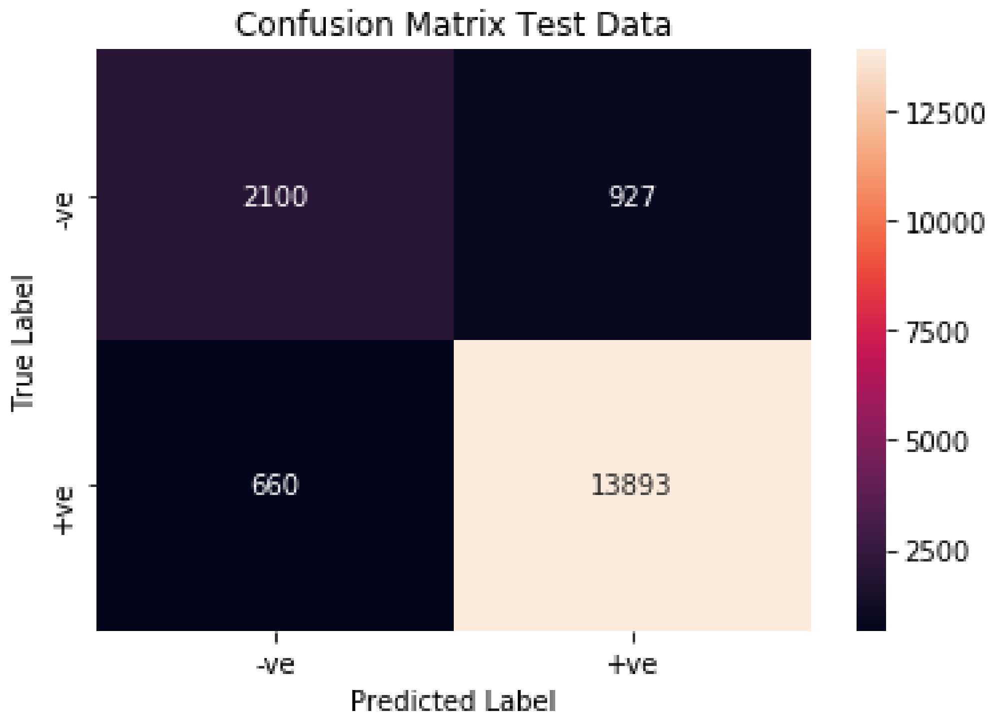

```
In [57]: showConfusionMatrix(train_confusion_matrix, "Confusion Matrix Train Data")
```

```
[[ 7008  1796]
 [  570 46878]]
```




```
In [58]: showConfusionMatrix(test_confusion_matrix, "Confusion Matrix Test Data")
```

```
[[ 2100   927]
 [  660 13893]]
```




Conclusion: There are 1587 misclassified data. Accuracy is about 91%

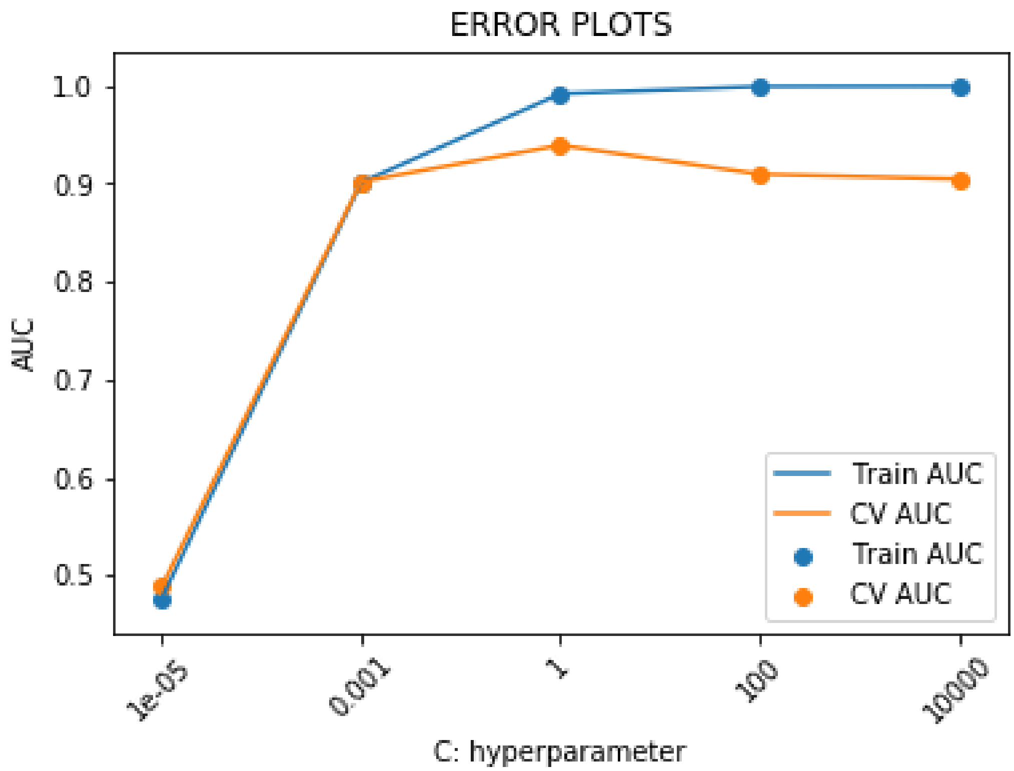
**[5.1.1.1] Calculating sparsity on weight vector obtained using L1 regularization on BOW, SET 1**

```
In [59]: #bow_l1_logRegressor = logRegressor
         w_bow = bow_l1_logRegressor.coef_
         print(type(w_bow))
         print(w_bow.shape)
         print("Number of non-zero weights : ", np.count_nonzero(w_bow))
```

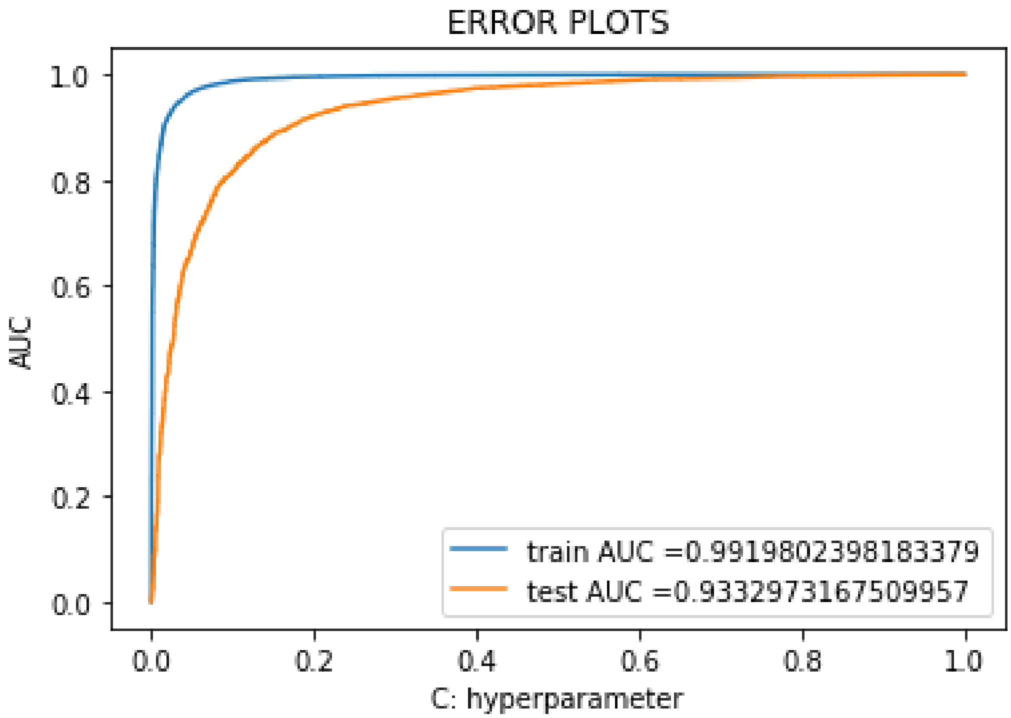
```
<class 'numpy.ndarray'>
(1, 44053)
Number of non-zero weights :  4435
```

### 7.1.2 [5.1.2] Applying Logistic Regression with L2 regularization on BOW, SET 1

```
In [48]: bow_l2_logRegressor = getOptimalLamda(X_train_bow, y_train, X_cv_bow, y_cv,'l2')
```

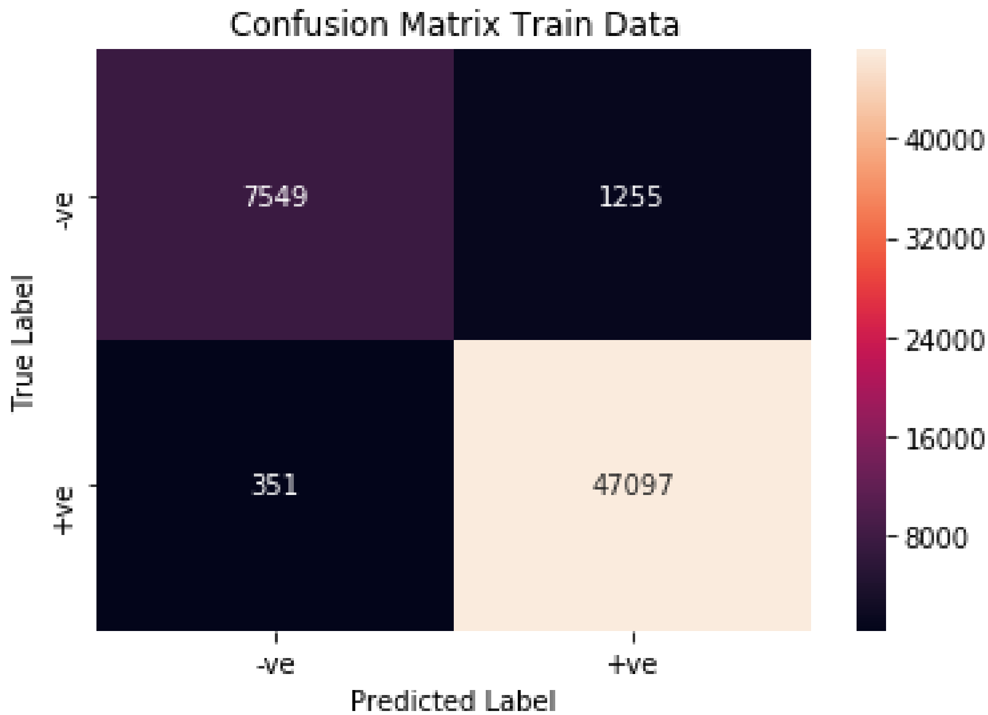



```
In [60]: #Best value of AUC at C=1
         bow_l2_logRegressor, train_confusion_matrix, test_confusion_matrix = getLRAnalysis(1,
```



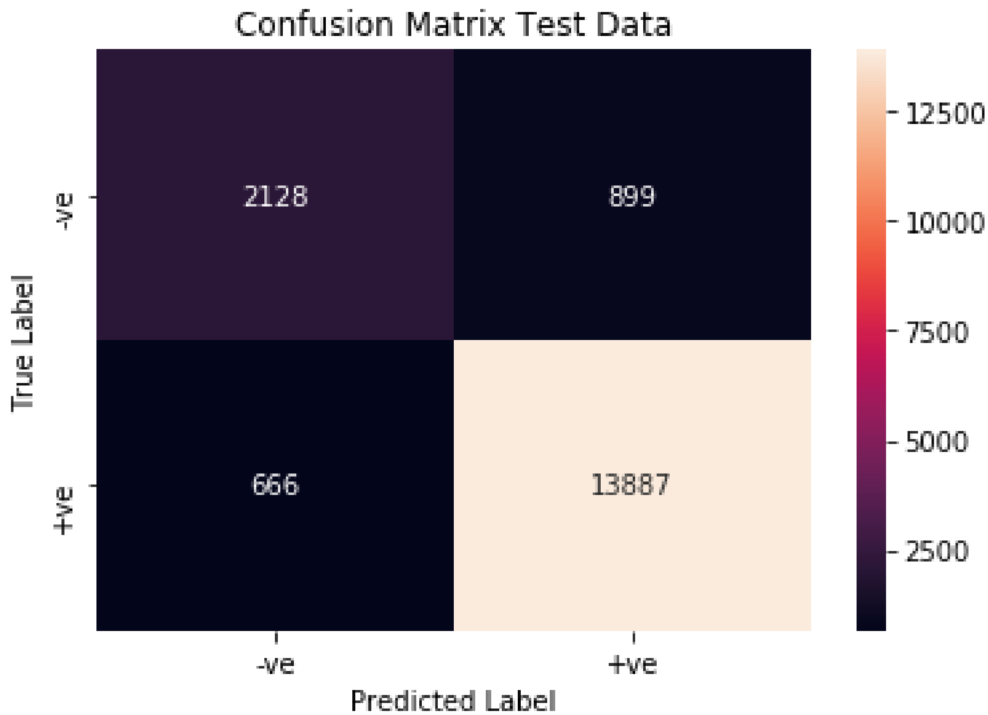

```
In [61]: showConfusionMatrix(train_confusion_matrix, "Confusion Matrix Train Data")
```

```
[[ 7549  1255]
 [  351 47097]]
```




```
In [62]: showConfusionMatrix(test_confusion_matrix, "Confusion Matrix Test Data")
```

```
[[ 2128   899]
 [  666 13887]]
```




Conclusion : Total misclassified data 1565. Accuracy = 91.1%

**[5.1.2.1] Performing pertubation test (multicollinearity check) on BOW, SET 1**

```
In [64]: #getting a small number for pertubation
         e = np.random.uniform(0,0.1)
         print(e)
```

```
0.08327752613862262
```

```
In [68]: #Getting weight vector after LR using l2 regularizer
         weight_l2 = bow_l2_logRegressor.coef_
         print(type(weight_l2))
         print(weight_l2[:10])
```

```
<class 'numpy.ndarray'>
[[-5.17165861e-01  1.46767645e-02  3.31208892e-02 ... -6.48945854e-02
   1.70171983e-04  9.42933425e-02]]
```

```
In [80]: print(type(X_train_bow))
         print(X_train_bow.shape)
         print(X_train_bow.todense())
         print("----After Pertubation----")
```

```
print(X_train_bow.data)
X_train_bow.data = X_train_bow.data + e;
print(X_train_bow.todense())
print(X_train_bow.shape)
```

```
<class 'scipy.sparse.csr.csr_matrix'>
(56252, 44053)
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
----After Pertubation----
[1 1 1 ... 1 1 1]
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
(56252, 44053)
```

In [81]:
```
pertubatedLogisticRegressorl2 = LogisticRegression(penalty = 'l2', C = 1)
pertubatedLogisticRegressorl2.fit(X_train_bow, y_train)
```

Out[81]:
```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [82]:
```
#Getting weight vector after LR using l2 regularizer
weight_l2_pertubated = pertubatedLogisticRegressorl2.coef_
print(type(weight_l2_pertubated))
print(weight_l2_pertubated[:10])
```

```
<class 'numpy.ndarray'>
[[-5.32610321e-01  1.43616627e-02  3.39167645e-02 ... -6.35213250e-02
   1.58243496e-04  9.67655737e-02]]
```

In [83]:
```
#adding a small number 0.000001 to each element in weight vector
#to eliminate division by error for furthur calculations
weight_l2_new = weight_l2 +  0.000001
weight_l2_pertubated_new = weight_l2_pertubated + 0.000001
print(weight_l2_new[:10])
print(weight_l2_pertubated_new[:10])
```

```
[[-5.17164861e-01  1.46777645e-02  3.31218892e-02 ... -6.48935854e-02
   1.71171983e-04  9.42943425e-02]]
[[-5.32609321e-01  1.43626627e-02  3.39177645e-02 ... -6.35203250e-02
   1.59243496e-04  9.67665737e-02]]
```

```
In [84]: #calculate percentage change in Weight vector
         w_per = abs((weight_l2_new - weight_l2_pertubated_new)/weight_l2_new)*100
         print(w_per)
```

```
[[2.98637086 2.14679729 2.40286791 ... 2.1161728  6.96871462 2.62182348]]
```

```
In [97]: percentile_list = [0,10,20,30,40,50,60,70,80,90,100]

         perentile_output = []

         for i in percentile_list:
             p = np.percentile(w_per, i)
             perentile_output.append(p)
         print(perentile_output)
```

```
[0.00010411663672361845, 0.8190461015338346, 1.7277127892761195, 2.697974471816388, 3.82699128
```

```
In [98]: print("99 percentile : ", np.percentile(w_per, 99))
         print("100 percentile : ", np.percentile(w_per, 100))
```

```
99 percentile :  289.43794539052897
100 percentile :  151031.01192276357
```

We can see sudden change in percentile from 99 to 100, so now will calculate percentile from 99.1 till 99.9

```
In [99]: percentile99_list = [99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9]
         perentile99_output = []

         for i in percentile99_list:
             p = np.percentile(w_per, i)
             perentile99_output.append(p)
         print(perentile99_output)
```

```
[324.6795182973906, 343.3149643929928, 393.1446108599696, 509.49457241774655, 623.60237264831
```

After 1099.58 there is sudden rise. So wee need to get all the features vaing % change more than 1099.58

```
In [103]: feature_names= vectorizer.get_feature_names()
          dictionary = dict(zip(feature_names, w_per[0]))
          filteredOutFeature = {k:v for (k,v) in dictionary.items() if (v>1099.58).any()}
          print(len(filteredOutFeature))
          filteredOutFeature
```

```
133
```

```
Out[103]: {'abita': 1662.5583415140482,
           'abv': 2579.7674853276644,
           'adust': 15060.48386156829,
           'amounting': 15060.48386156829,
           'anticipated': 1294.274166255276,
           'arouma': 151031.01192276357,
           'bamilton': 1564.7896256756396,
           'beated': 4573.891537793525,
           'bht': 1808.1871214855212,
           'bige': 15060.48386156829,
           'biscit': 2275.759845777143,
           'bloodroot': 15060.48386156829,
           'bounty': 3531.3308291117687,
           'bythe': 1215.801452818658,
           'caffeic': 1723.1808521256223,
           'calbom': 1561.0777757855492,
           'calcelled': 1463.0874548510244,
           'candying': 6320.8552563496805,
           'cannned': 1215.801452818658,
           'carbonating': 4225.487287021573,
           'cassrole': 2253.0950715192075,
           'charitable': 1161.8380514093953,
           'cherie': 1561.0777757855492,
           'choicest': 1564.7896256756396,
           'cichoric': 1723.1808521256223,
           'cinni': 3684.099315261642,
           'coughing': 1416.754981726983,
           'cranny': 3075.944666251149,
           'cumminty': 1662.5583415140482,
           'cuprecommended': 1718.3958560892218,
           'curtiss': 1120.7754319123308,
           'dicission': 1662.5583415140482,
           'doggone': 1704.5611182042953,
           'dozers': 2275.759845777143,
           'drizzing': 2253.0950715192075,
           'drizzles': 2253.0950715192075,
           'drowsiness': 2061.8911465821325,
           'emergence': 2275.759845777143,
           'encounted': 1103.6685675784308,
```

```
'epidemiologists': 1463.0874548510244,
'epilespy': 1662.5583415140482,
'exacto': 4057.9141258665663,
'facter': 1662.5583415140482,
'familiarize': 3960.778575209162,
'forensic': 1463.0874548510244,
'funded': 2275.759845777143,
'fusili': 14091.094857734717,
'goldfish': 2976.552595941047,
'guatemalan': 21538.398125384494,
'havery': 2022.8489824739786,
'herbalism': 1561.0777757855492,
'hut': 2579.7674853276644,
'hyson': 1707.4557587149109,
'importation': 1463.0874548510244,
'interferons': 1723.1808521256223,
'irrigating': 15060.48386156829,
'ivs': 15060.48386156829,
'jaws': 2009.519145180398,
'kimberly': 1662.5583415140482,
'kiss': 51612.40583635464,
'latches': 4057.9141258665663,
'laughing': 1139.2846343840813,
'managers': 3345.414687074175,
'merlots': 2579.7674853276644,
'metabolize': 1787.6675997949508,
'microbiologists': 1463.0874548510244,
'montepulciano': 2579.7674853276644,
'motepulciano': 2579.7674853276644,
'multifaceted': 1463.0874548510244,
'musketters': 4154.575148226861,
'nerdy': 39049.66467460291,
'normalize': 1764.3383307867125,
'nostils': 3075.944666251149,
'occassion': 1433.4114578863148,
'occuring': 1856.8095803665262,
'oila': 2253.0950715192075,
'ojibwah': 15060.48386156829,
'ouncesservings': 14091.094857734717,
'outlier': 5353.984984717582,
'pandan': 2329.7089921602987,
'partly': 11279.22068303081,
'persuade': 2315.172091486652,
'pitcairn': 2275.759845777143,
'playfully': 15060.48386156829,
'polysaccharides': 1723.1808521256223,
'portraits': 15060.48386156829,
'prebiotic': 1371.2367672468424,
```

'prodding': 2275.759845777143,
'prophylactically': 15060.48386156829,
'propped': 3075.944666251149,
'prove': 5394.56256700215,
'provinces': 1463.0874548510244,
'purification': 15060.48386156829,
'purifying': 15060.48386156829,
'racquette': 2501.4123241820585,
'reccomendationthe': 3735.281640047943,
'recruiter': 2275.759845777143,
'reisling': 2579.7674853276644,
'reschedule': 1190.511232320253,
'researches': 2275.759845777143,
'rhe': 1561.0777757855492,
'rouses': 1662.5583415140482,
'sammay': 2579.7674853276644,
'sanitary': 86787.78426717786,
'sao': 1961.3514678191236,
'screeming': 15060.48386156829,
'shorman': 1662.5583415140482,
'shot': 2803.5467266315654,
'shriveling': 55669.47981656627,
'slider': 2348.3261195640653,
'smoky': 9375.488079954303,
'softish': 2022.8489824739786,
'sparkling': 1350.8955790253863,
'stretch': 1143.739495452195,
'substle': 2253.0950715192075,
'sustainably': 5828.783248402409,
'suv': 15060.48386156829,
'target': 4133.69647389276,
'tempature': 1662.5583415140482,
'tome': 1961.3514678191236,
'toxicologists': 1463.0874548510244,
'tracing': 2275.759845777143,
'triticale': 2022.8489824739786,
'untreatable': 15060.48386156829,
'usnea': 15060.48386156829,
'valerian': 1982.7248146545544,
'vaseline': 3769.7772857253353,
'vida': 2579.7674853276644,
'winded': 2275.759845777143,
'wired': 1211.500496396248,
'wulongs': 1503.2127302333765,
'zinc': 27626.30172417036,
'zinfandels': 2579.7674853276644}

### 7.1.3 [5.1.3] Feature Importance on BOW, SET 1

```
In [133]: coef = bow_l2_logRegressor.coef_[0]
          sortedIdx = np.argsort(coef)
          top10PositiveFeatureIdx = sortedIdx[-10:]
          topTenNegativeFeatureIdx = sortedIdx[:10]
          print(top10PositiveFeatureIdx)
          print(topTenNegativeFeatureIdx)
```

```
[13321  1234 43858  2645 18379  7868 43885 10104  3241 29115]
[43379 38466  5521 38824 10943 10945 32812  5518 10682 39327]
```

```
In [134]: feature_names= vectorizer.get_feature_names()

          topTenPositiveFeatureList = [feature_names[i] for i in top10PositiveFeatureIdx]

          topTenNegativeFeatureList = [feature_names[i] for i in topTenNegativeFeatureIdx]


          topTenPositiveWeight = [coef[i] for i in top10PositiveFeatureIdx]
          topTenNegativeWeight = [coef[i] for i in topTenNegativeFeatureIdx]

          print("Tope ten positive features: ",topTenPositiveFeatureList)
          print("Tope ten positive weights: ",topTenPositiveWeight)

          print("Top ten negative features: ",topTenNegativeFeatureList)
          print("Top ten negative weights: ",topTenNegativeWeight)
```

```
Tope ten positive features:  ['excellent', 'amazing', 'yum', 'awesome', 'hooked', 'complaint',
Tope ten positive weights:  [1.9174087932681387, 1.9393521400993876, 1.9557003317600297, 1.962
Top ten negative features:  ['worst', 'tasteless', 'cancelled', 'terrible', 'disappointing', 'c
Top ten negative weights:  [-3.203397934417593, -2.4598172074475197, -2.3462030141843795, -2.2
```

**[5.1.3.1] Top 10 important features of positive class from SET 1**

```
In [135]:     print("Postive feature list : ",dict(zip(topTenPositiveFeatureList, topTenPositi
```

```
Postive feature list :  {'excellent': 1.9174087932681387, 'amazing': 1.9393521400993876, 'yum'
```

**[5.1.3.2] Top 10 important features of negative class from SET 1**

```
In [136]: print("Negative feature list : ",dict(zip(topTenNegativeFeatureList, topTenNegativeWe
```

```
Negative feature list :  {'worst': -3.203397934417593, 'tasteless': -2.4598172074475197, 'canc
```

## 7.2 [5.2] Logistic Regression on TFIDF, SET 2

### 7.2.1 [5.2.1] Applying Logistic Regression with L1 regularization on TFIDF, SET 2

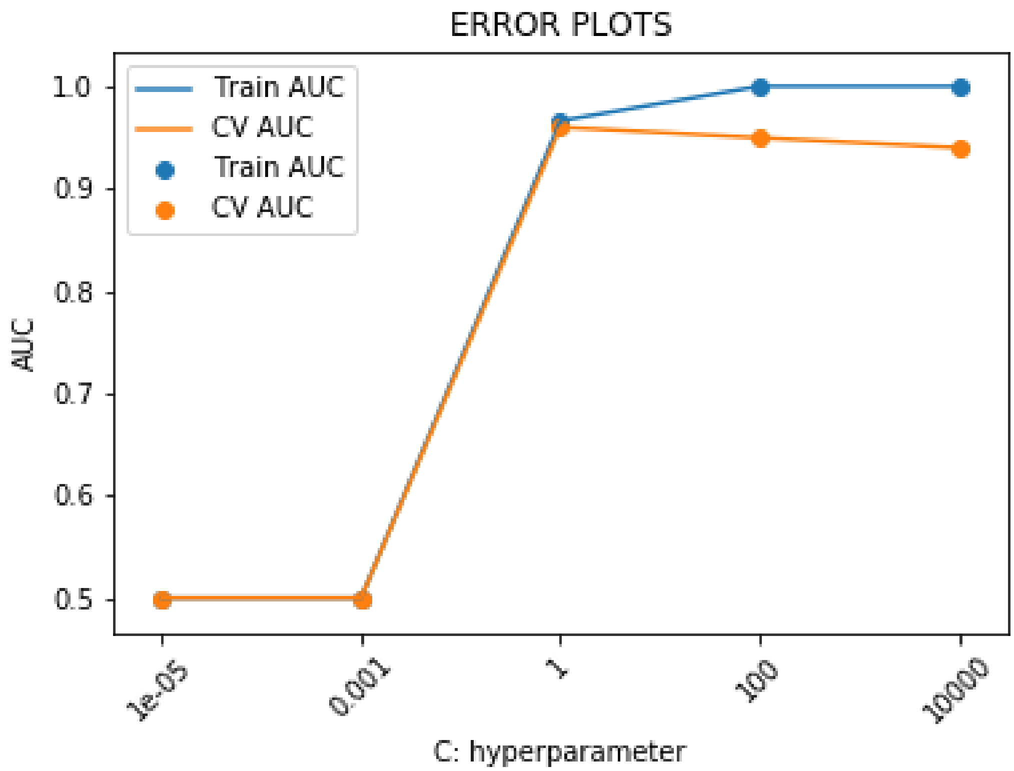
In [137]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
#Apply tf-idf on splitted data sets
X_train_tfidf = tf_idf_vect.fit_transform(X_train)
X_cv_tfidf = tf_idf_vect.transform(X_cv)
X_test_tfidf = tf_idf_vect.transform(X_test)

print(type(X_train_tfidf))
print(type(X_cv_tfidf))
print(type(X_test_tfidf))
```

```
<class 'scipy.sparse.csr.csr_matrix'>
<class 'scipy.sparse.csr.csr_matrix'>
<class 'scipy.sparse.csr.csr_matrix'>
```

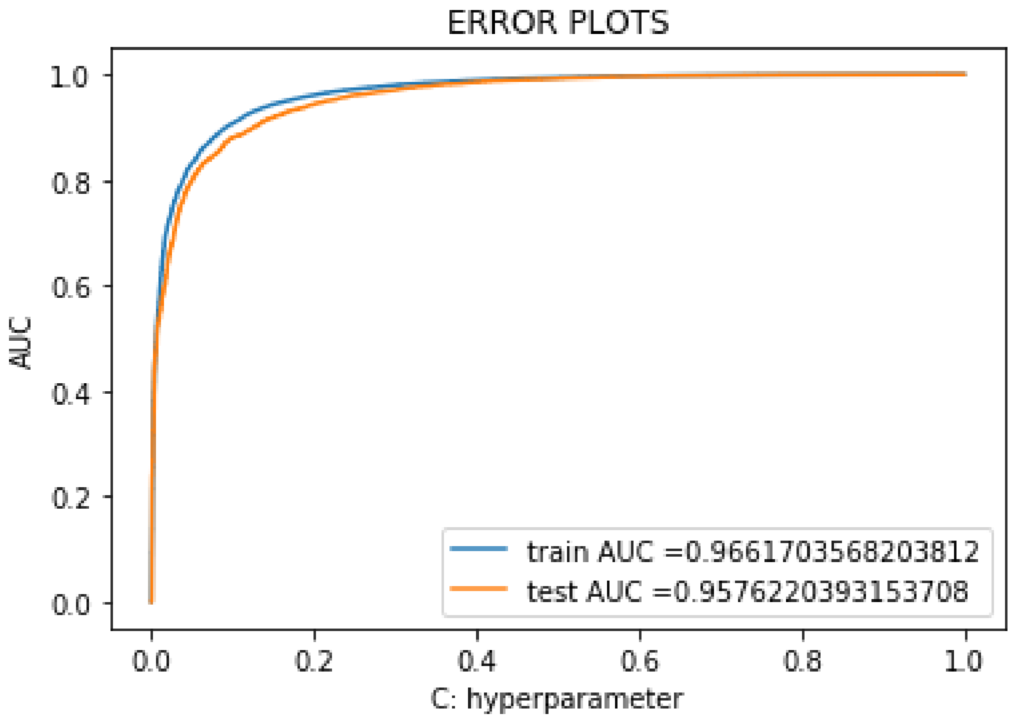
In [138]:
```
print(X_train_tfidf.shape)
print(X_cv_tfidf.shape)
print(X_test_tfidf.shape)
```

```
(56252, 32863)
(14064, 32863)
(17580, 32863)
```

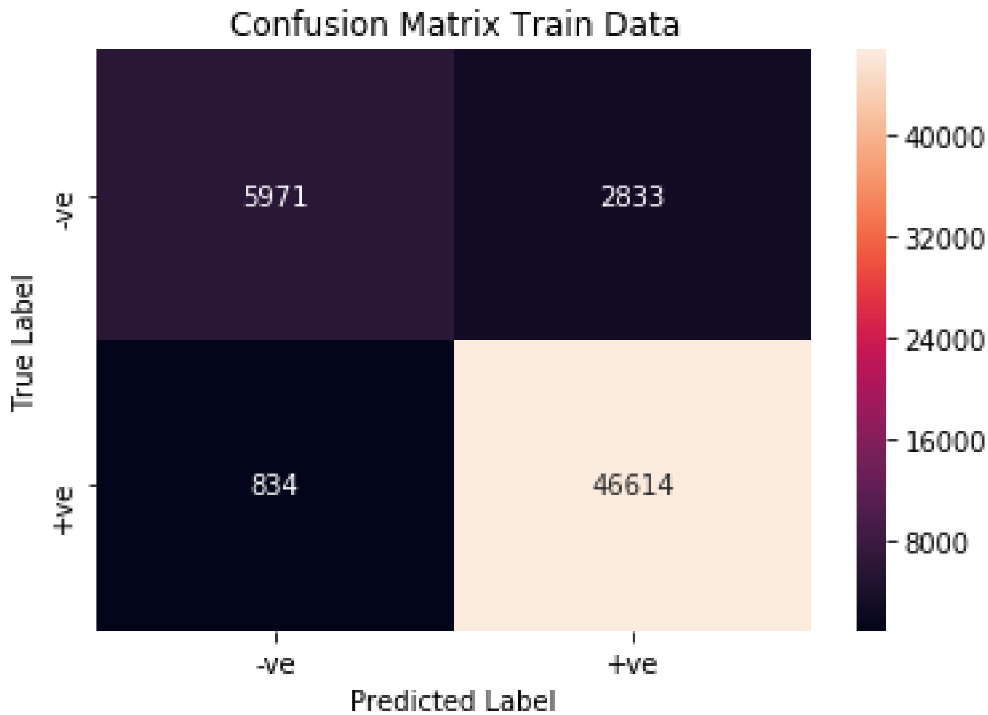
In [139]:
```
logRegressor_l1_tfidf = getOptimalLamda(X_train_tfidf, y_train, X_cv_tfidf, y_cv,'l1
```




```
In [140]: #Best value of AUC at C=1
          tfidf_l1_logRegressor, train_confusion_matrix, test_confusion_matrix = getLRAnalysis
```

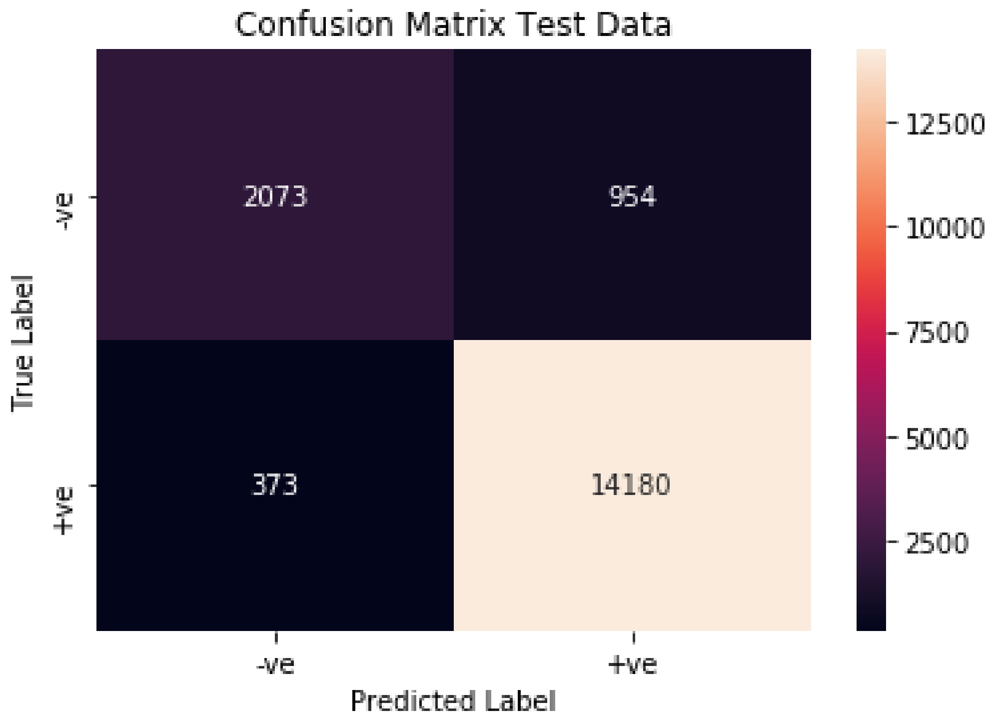



```
In [141]: showConfusionMatrix(train_confusion_matrix, "Confusion Matrix Train Data")

[[ 5971  2833]
 [  834 46614]]
```




```
In [142]: showConfusionMatrix(test_confusion_matrix, "Confusion Matrix Test Data")
```

```
[[ 2073   954]
 [  373 14180]]
```

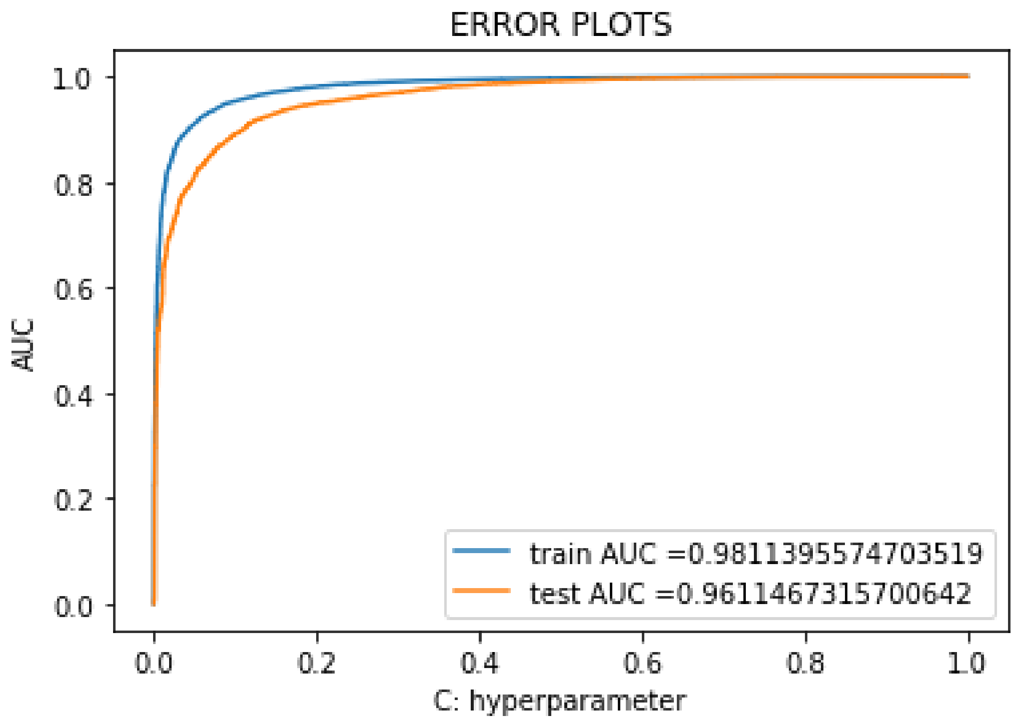



Conclusion: total misclassified points are (373+954) = 1327. Accuracy = 92.45%

#### 7.2.2 [5.2.2] Applying Logistic Regression with L2 regularization on TFIDF, SET 2

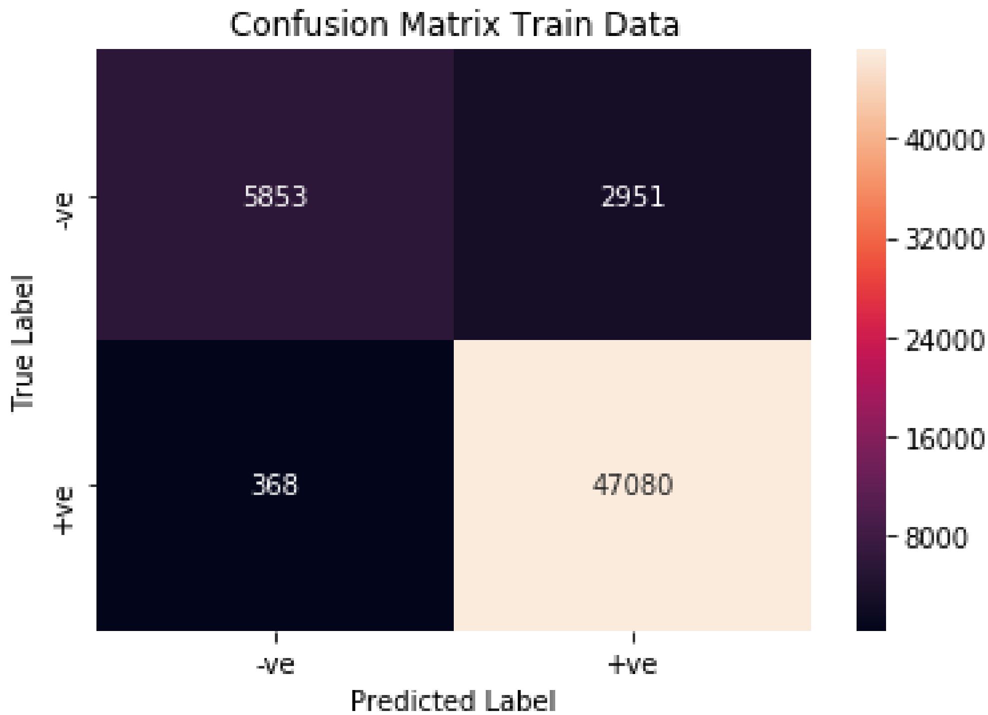
```
In [143]: logRegressor_l2_tfidf = getOptimalLamda(X_train_tfidf, y_train, X_cv_tfidf, y_cv,'l2
```


```
In [144]: #Best value of AUC at C=1
          tfidf_l2_logRegressor, train_confusion_matrix, test_confusion_matrix = getLRAnalysis
```



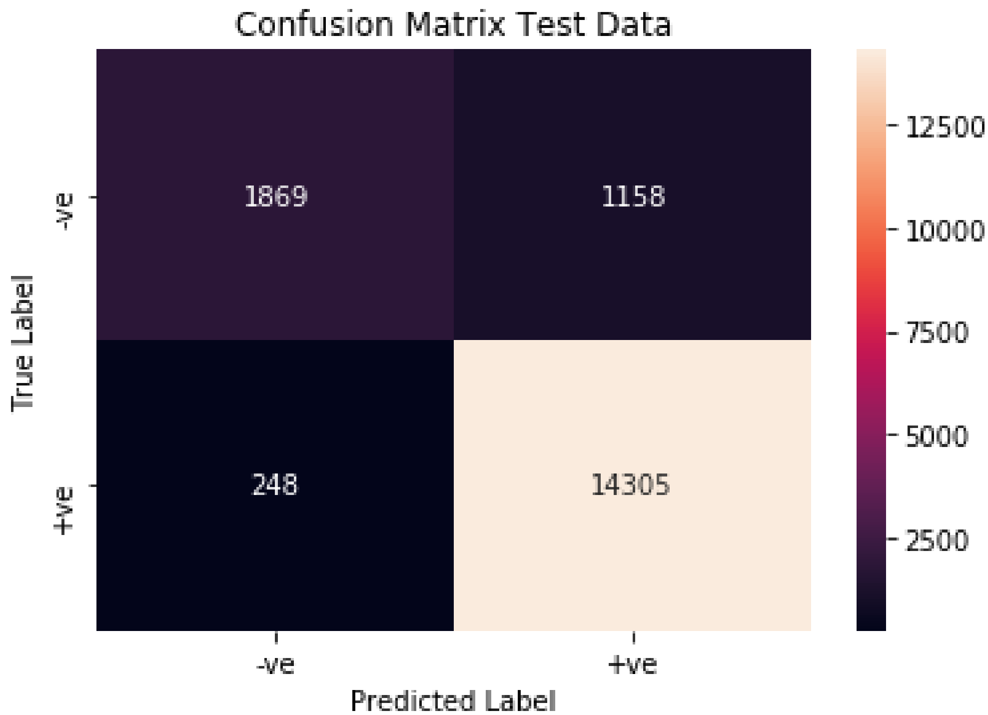

In [145]: showConfusionMatrix(train_confusion_matrix, "Confusion Matrix Train Data")

```
[[ 5853  2951]
 [  368 47080]]
```




```
In [146]: showConfusionMatrix(test_confusion_matrix, "Confusion Matrix Test Data")
```

```
[[ 1869  1158]
 [  248 14305]]
```




Conclusion : Total misclassified points 1406. Accuracy = 92%

### 7.2.3 [5.2.3] Feature Importance on TFIDF, SET 2

In [162]:
```
coef = tfidf_l2_logRegressor.coef_[0]
sortedIdx = np.argsort(coef)
top10PositiveFeatureIdx = sortedIdx[-10:]
topTenNegativeFeatureIdx = sortedIdx[:10]
print(top10PositiveFeatureIdx)
print(topTenNegativeFeatureIdx)

feature_names= tf_idf_vect.get_feature_names()

topTenPositiveFeatureList = [feature_names[i] for i in top10PositiveFeatureIdx]

topTenNegativeFeatureList = [feature_names[i] for i in topTenNegativeFeatureIdx]


topTenPositiveWeight = [coef[i] for i in top10PositiveFeatureIdx]
topTenNegativeWeight = [coef[i] for i in topTenNegativeFeatureIdx]

print("Tope ten positive features: ",topTenPositiveFeatureList)
print("Tope ten positive weights: ",topTenPositiveWeight)
```

```
print("Top ten negative features: ",topTenNegativeFeatureList)
print("Top ten negative weights: ",topTenNegativeWeight)
```

```
[18854 32249  8989 16515 21556 16766 12084  2260  6848 12638]
[ 7298 19196 32394 29073 19458 19695 19894  1619  7314 13751]
Tope ten positive features:  ['nice', 'wonderful', 'excellent', 'love', 'perfect', 'loves', 'g
Tope ten positive weights:  [5.135908370078327, 5.429225554964319, 5.779207640612991, 6.240055
Top ten negative features:  ['disappointed', 'not', 'worst', 'terrible', 'not good', 'not reco
Top ten negative weights:  [-7.959040793265328, -7.262654893590533, -6.721229673125427, -5.763
```

#### [5.2.3.1] Top 10 important features of positive class from SET 2

```
In [163]: print("Postive feature list : ",dict(zip(topTenPositiveFeatureList, topTenPositiveWe
```

```
Postive feature list :  {'nice': 5.135908370078327, 'wonderful': 5.429225554964319, 'excellent
```

#### [5.2.3.2] Top 10 important features of negative class from SET 2

```
In [164]: print("Negative feature list : ",dict(zip(topTenNegativeFeatureList, topTenNegativeW
```

```
Negative feature list :  {'disappointed': -7.959040793265328, 'not': -7.262654893590533, 'wors
```

### 7.3 [5.3] Logistic Regression on AVG W2V, SET 3

```
In [150]: from tqdm import tqdm
          import numpy as np
          from gensim.models import Word2Vec
          from gensim.models import KeyedVectors
```

```
In [151]: i=0
          list_of_sentance_train=[]
          for sentance in X_train:
              list_of_sentance_train.append(sentance.split())

          # this line of code trains your w2v model on the give list of sentances
          w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)

          w2v_words = list(w2v_model.wv.vocab)
```

```
In [152]: #converting cv data
          i=0
          list_of_sentance_cv=[]
          for sentance in X_cv:
              list_of_sentance_cv.append(sentance.split())
```

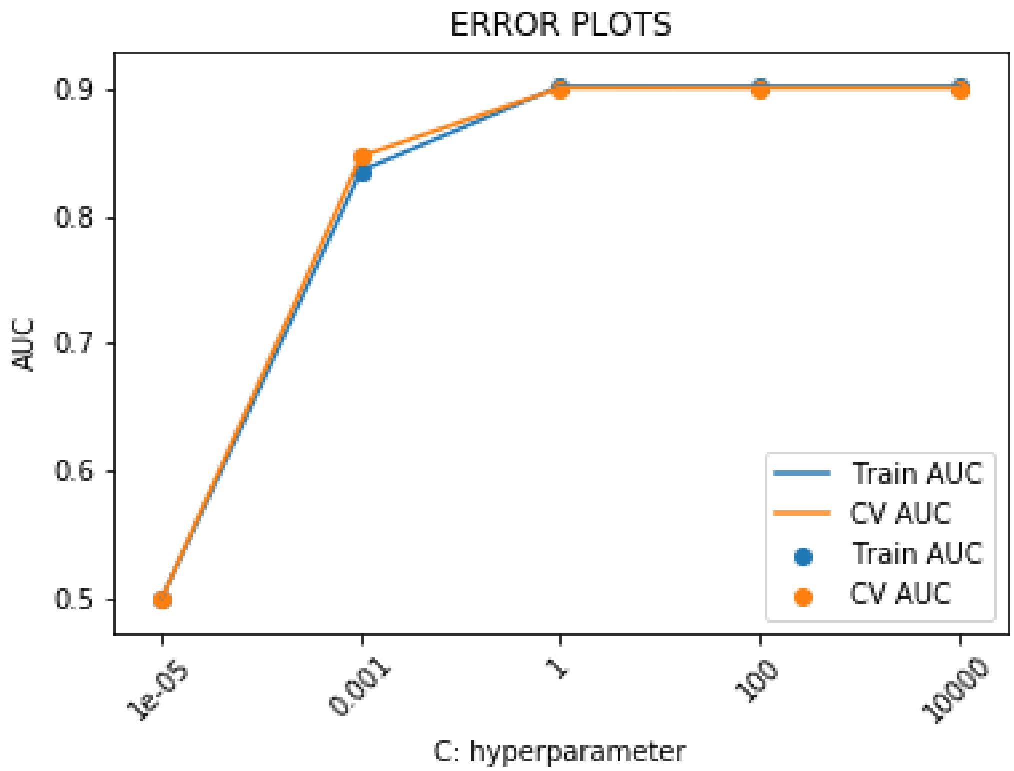
```
In [153]: #Converting for test data
          i=0
          list_of_sentance_test=[]
          for sentance in X_test:
              list_of_sentance_test.append(sentance.split())

In [159]: def getAvgW2V(list_of_sentance):
              # average Word2Vec
              # compute average word2vec for each review.
              sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
              for sent in tqdm(list_of_sentance): # for each review/sentence
                  sent_vec = np.zeros(50)
                  cnt_words =0; # num of words with a valid vector in the sentence/review
                  for word in sent: # for each word in a review/sentence
                      if word in w2v_words:
                          vec = w2v_model.wv[word]
                          sent_vec += vec
                          cnt_words += 1
                  if cnt_words != 0:
                      sent_vec /= cnt_words
                  sent_vectors.append(sent_vec)
              sent_vectors = np.array(sent_vectors)
              print(sent_vectors.shape)
              return sent_vectors
```

### 7.3.1 [5.3.1] Applying Logistic Regression with L1 regularization on AVG W2V SET 3

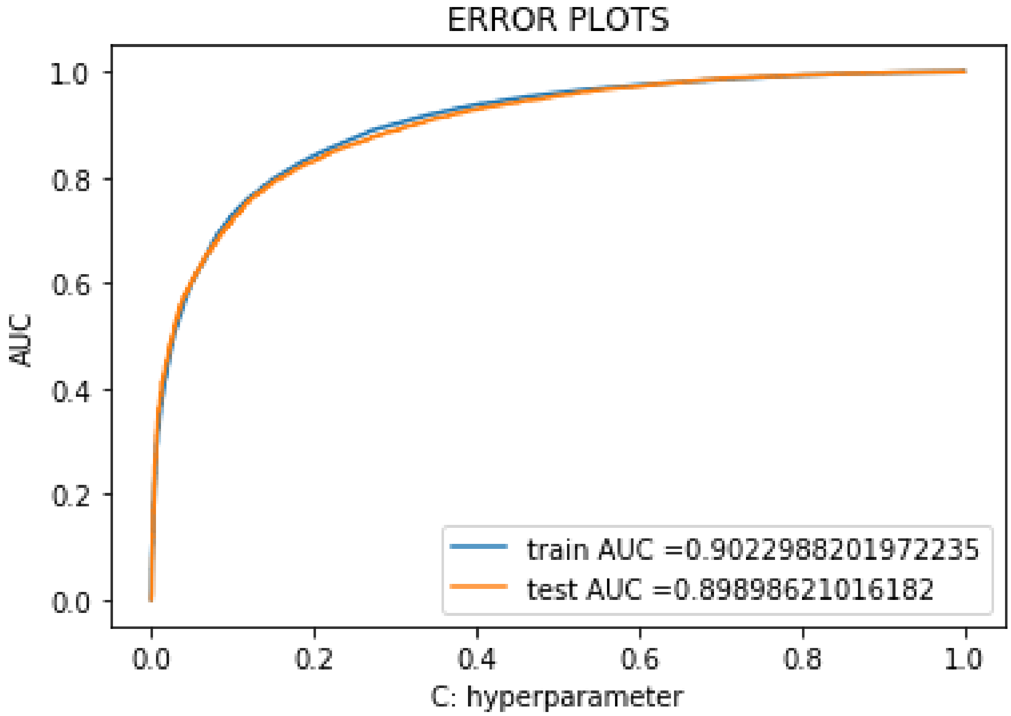
```
In [160]: sent_vectors_train = getAvgW2V(list_of_sentance_train)
          sent_vectors_cv = getAvgW2V(list_of_sentance_cv)
          sent_vectors_test = getAvgW2V(list_of_sentance_test)
```

```
100%|| 56252/56252 [03:33<00:00, 263.36it/s]

(56252, 50)

100%|| 14064/14064 [00:57<00:00, 245.19it/s]

(14064, 50)

100%|| 17580/17580 [01:13<00:00, 240.73it/s]

(17580, 50)
```

```
In [161]: logRegressor_l1_avgW2V = getOptimalLamda(sent_vectors_train, y_train, sent_vectors_cv
```

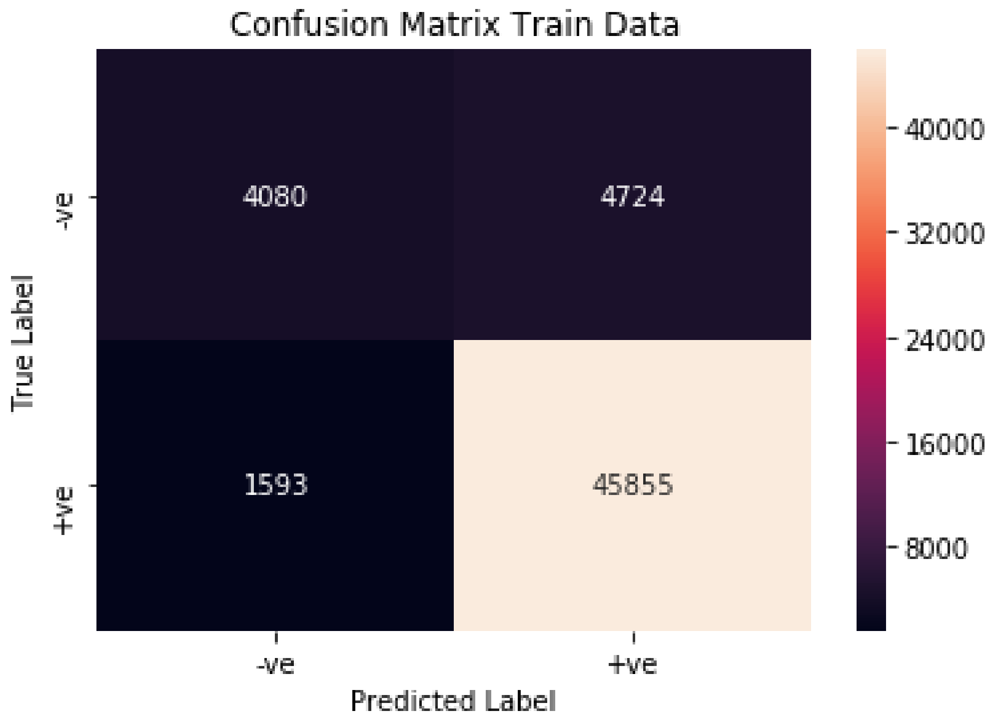



```
In [165]: #Best value of AUC at C=1
          avgW2V_l1_logRegressor, train_confusion_matrix, test_confusion_matrix = getLRAnalysis
```

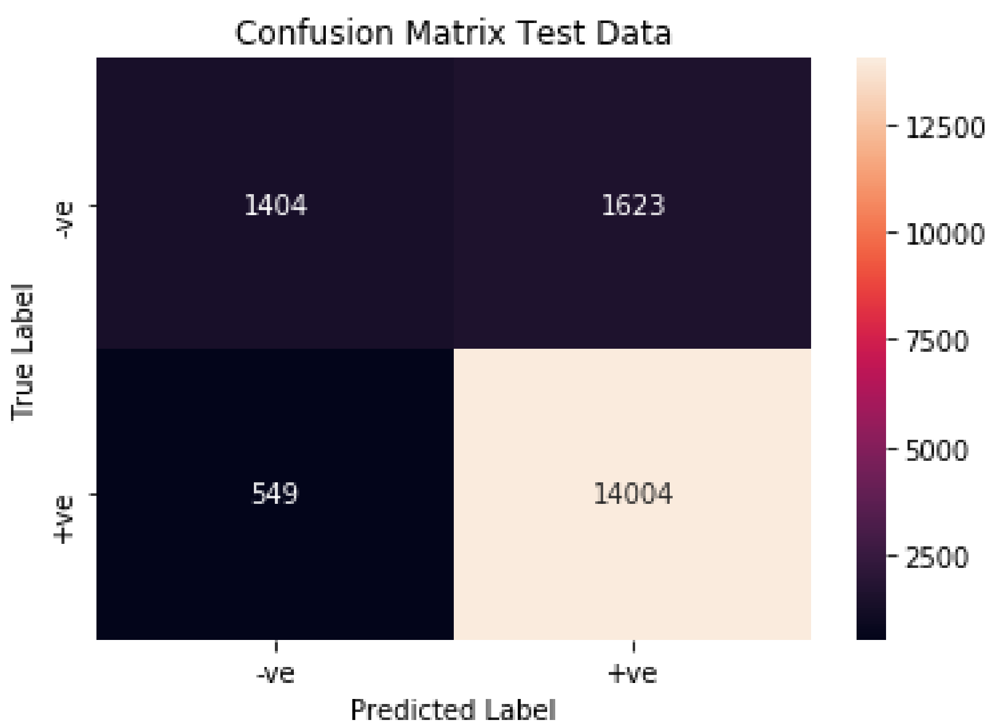



```
In [166]: showConfusionMatrix(train_confusion_matrix, "Confusion Matrix Train Data")

          showConfusionMatrix(test_confusion_matrix, "Confusion Matrix Test Data")
```

```
[[ 4080  4724]
 [ 1593 45855]]
```



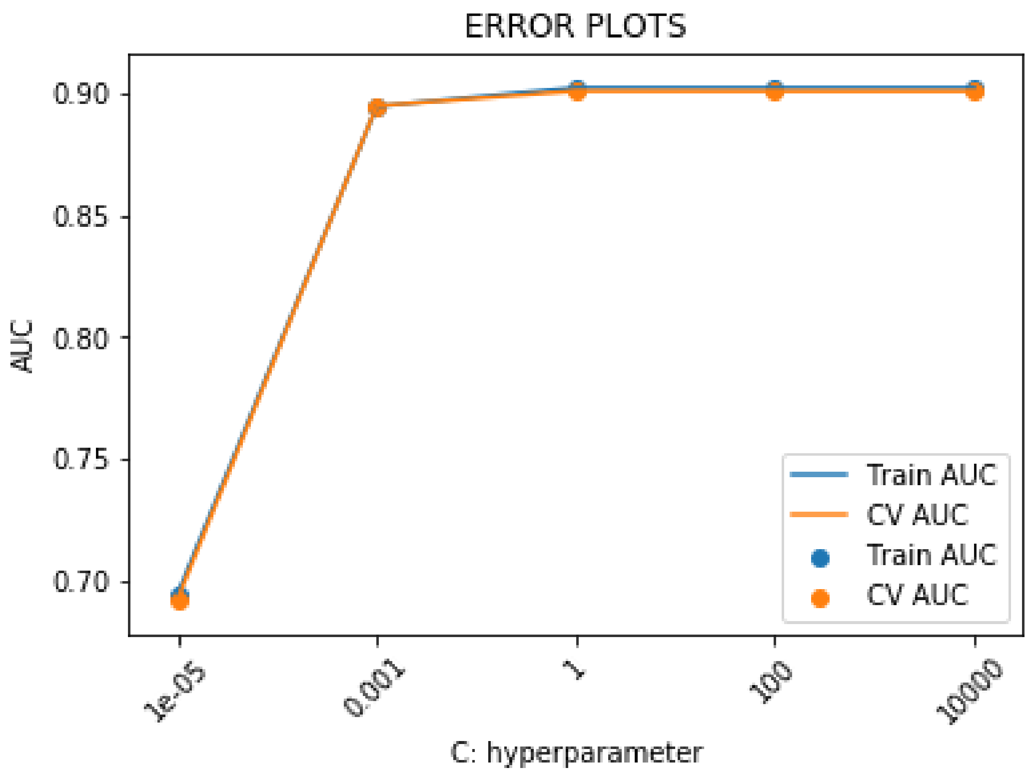

```
[[ 1404  1623]
 [  549 14004]]
```



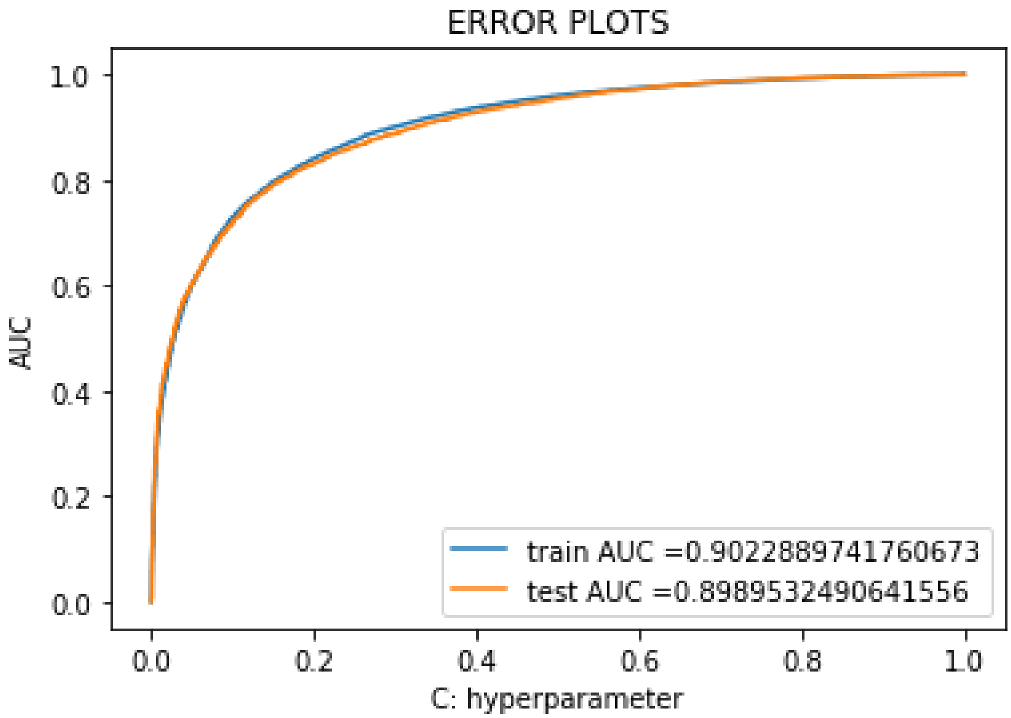
Conclusion : Total misclassified points (549+1623) = 2172 and Accuracy = 88%

#### 7.3.2 [5.3.2] Applying Logistic Regression with L2 regularization on AVG W2V, SET 3

```
In [167]: logRegressor_l2_avgW2V = getOptimalLamda(sent_vectors_train, y_train, sent_vectors_cv
```

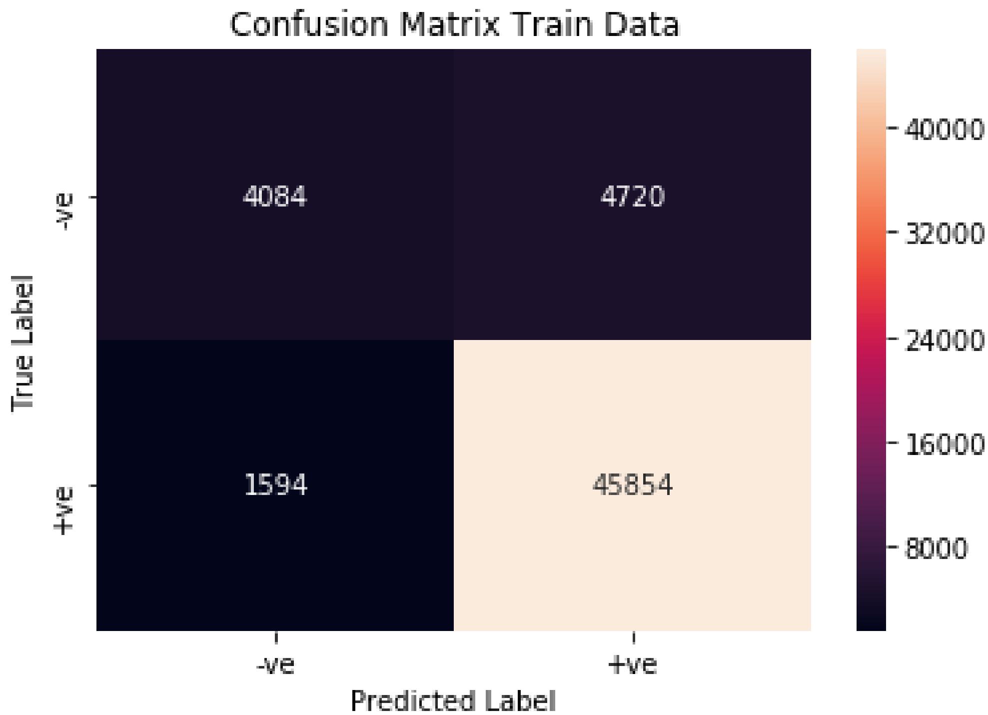



```
In [168]: #Best value of AUC heightst at C=1, 100 and 10000, so we are taking 1
          avgW2V_l2_logRegressor, train_confusion_matrix, test_confusion_matrix = getLRAnalysis
```

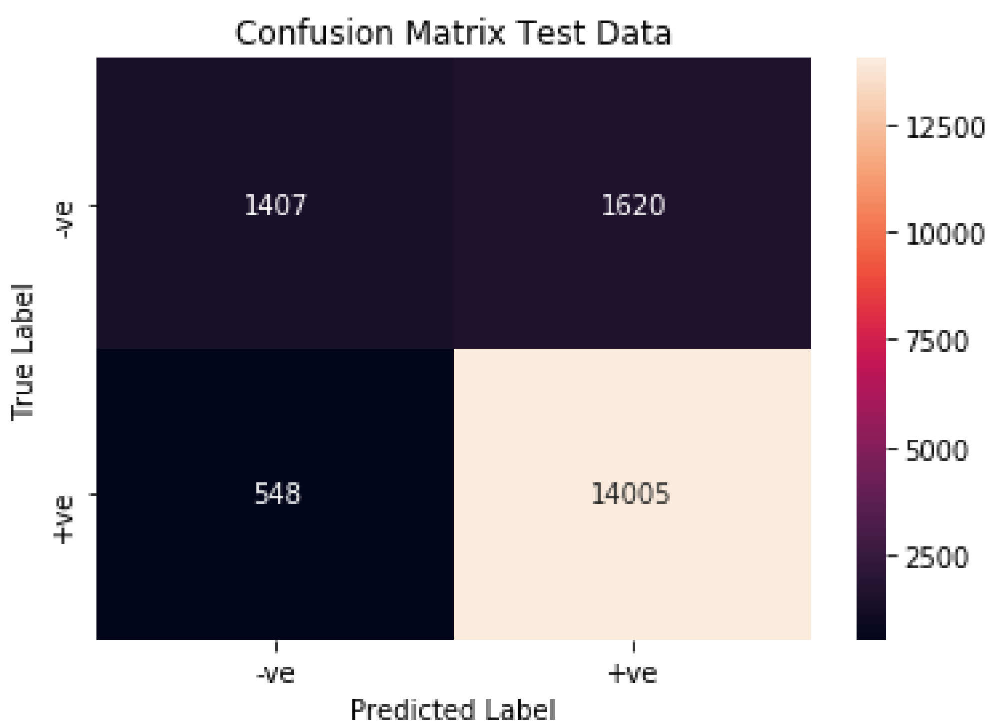



```
In [169]: showConfusionMatrix(train_confusion_matrix, "Confusion Matrix Train Data")

          showConfusionMatrix(test_confusion_matrix, "Confusion Matrix Test Data")
```

```
[[ 4084  4720]
 [ 1594 45854]]
```




```
[[ 1407  1620]
 [  548 14005]]
```



Conclusion : Total misclassified points = 2168 and Accuracy = 88%

## 7.4 [5.4] Logistic Regression on TFIDF W2V, SET 4

In [170]:
```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))

# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfid
```

In [174]:
```
i=0
list_of_sentance_train=[]
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())

i=0
list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())

i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
```

In [178]:
```
print(type(tf_idf_matrix))
print(type(tfidf_feat))
```

```
<class 'scipy.sparse.csr.csr_matrix'>
<class 'list'>
```

### 7.4.1 [5.4.1] Applying Logistic Regression with L1 regularization on TFIDF W2V, SET 4

In [180]:
```
tfidf_sent_vectors_train = []; # the tfidf-w2v for each sentence/review is stored in
row=0;
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
```

```
                # dictionary[word] = idf value of word in whole courpus
                # sent.count(word) = tf valeus of word in this review
                tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                sent_vec += (vec * tf_idf)
                weight_sum += tf_idf
        if weight_sum != 0:
            sent_vec /= weight_sum
        tfidf_sent_vectors_train.append(sent_vec)
        row += 1
```

```
100%|| 56252/56252 [1:01:35<00:00, 15.22it/s]
```

```
In [179]: tfidf_sent_vectors_cv = []; # the tfidf-w2v for each sentence/review is stored in th
          row=0;
          for sent in tqdm(list_of_sentance_cv): # for each review/sentence
              sent_vec = np.zeros(50) # as word vectors are of zero length
              weight_sum =0; # num of words with a valid vector in the sentence/review
              for word in sent: # for each word in a review/sentence
                  if word in w2v_words and word in tfidf_feat:
                      vec = w2v_model.wv[word]
          #             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                      # to reduce the computation we are
                      # dictionary[word] = idf value of word in whole courpus
                      # sent.count(word) = tf valeus of word in this review
                      tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                      sent_vec += (vec * tf_idf)
                      weight_sum += tf_idf
              if weight_sum != 0:
                  sent_vec /= weight_sum
              tfidf_sent_vectors_cv.append(sent_vec)
              row += 1
```

```
100%|| 14064/14064 [14:37<00:00, 16.02it/s]
```

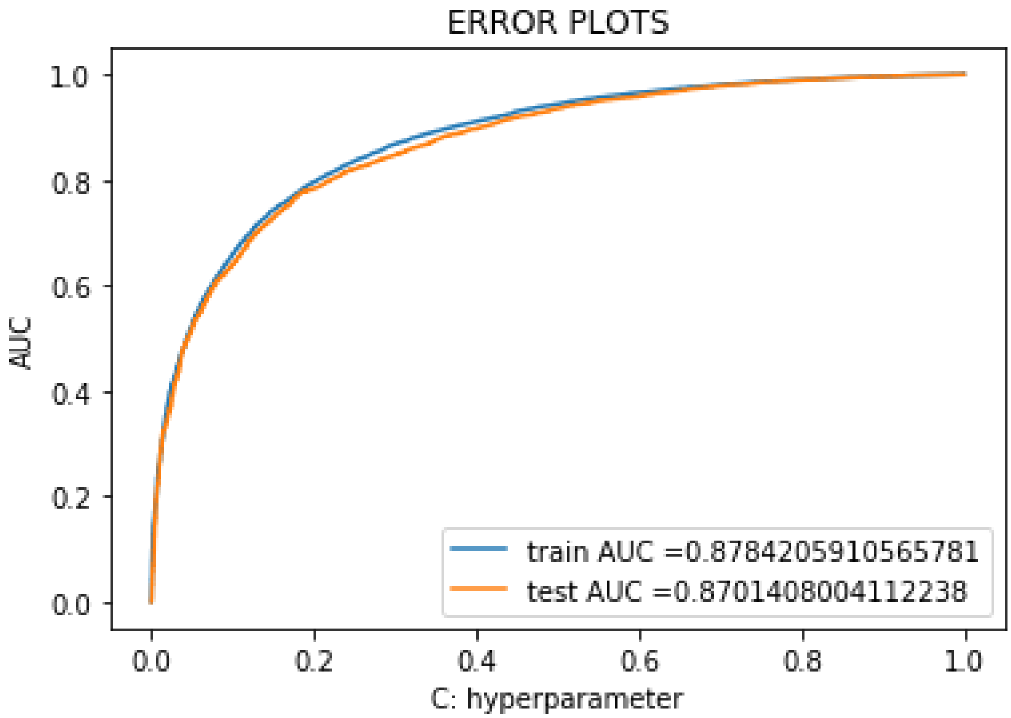
```
In [181]: tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review is stored in
          row=0;
          for sent in tqdm(list_of_sentance_test): # for each review/sentence
              sent_vec = np.zeros(50) # as word vectors are of zero length
              weight_sum =0; # num of words with a valid vector in the sentence/review
              for word in sent: # for each word in a review/sentence
                  if word in w2v_words and word in tfidf_feat:
                      vec = w2v_model.wv[word]
          #             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                      # to reduce the computation we are
                      # dictionary[word] = idf value of word in whole courpus
                      # sent.count(word) = tf valeus of word in this review
                      tf_idf = dictionary[word]*(sent.count(word)/len(sent))
```

```
                sent_vec += (vec * tf_idf)
                weight_sum += tf_idf
        if weight_sum != 0:
            sent_vec /= weight_sum
        tfidf_sent_vectors_test.append(sent_vec)
        row += 1
```

100%|| 17580/17580 [22:22<00:00, 13.10it/s]

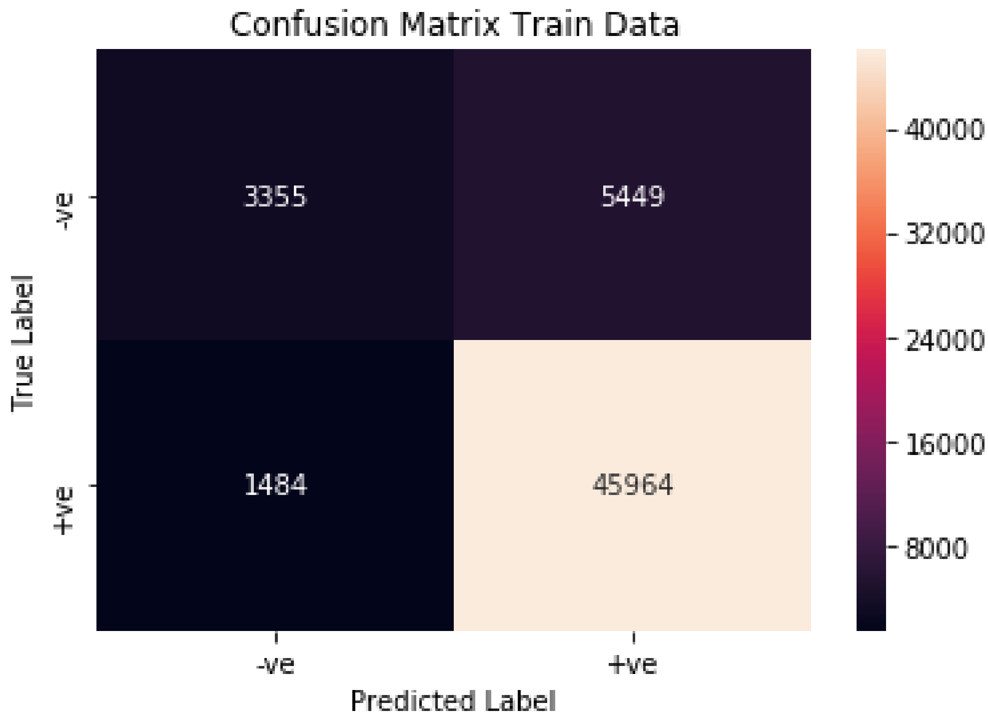
In [185]:
```
logRegressor_l1_tfidfW2V = getOptimalLamda(tfidf_sent_vectors_train, y_train, tfidf_s
```


In [186]:
```
#Best value of AUC heightst at C=1, 100 and 10000, so we are taking 1
tfidfW2V_l1_logRegressor, train_confusion_matrix, test_confusion_matrix = getLRAnalys
```

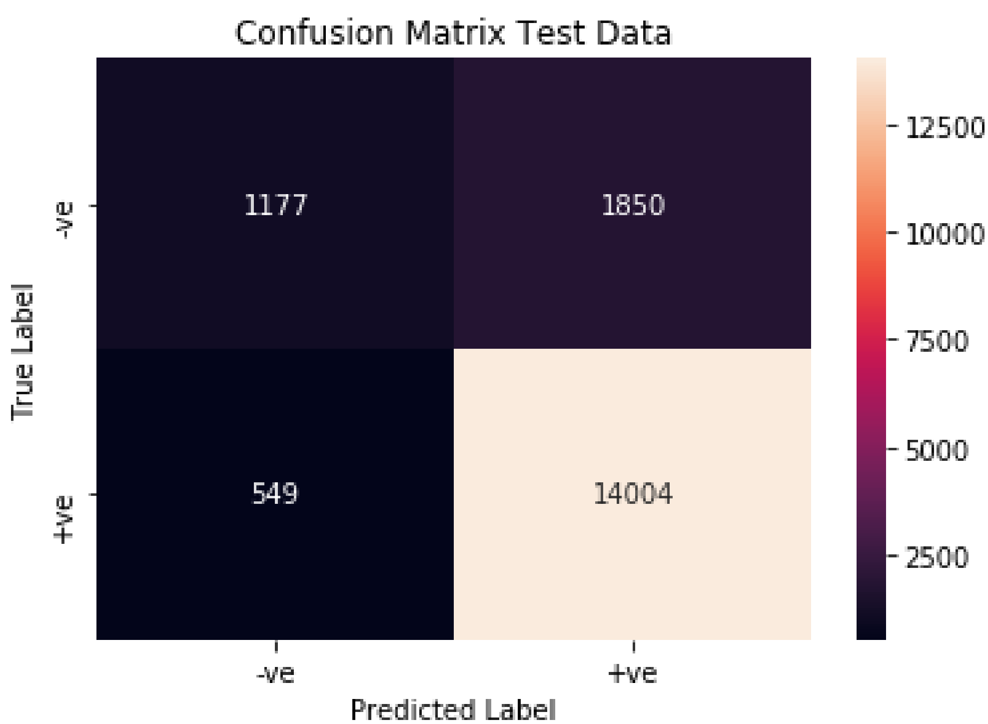



```
In [187]: showConfusionMatrix(train_confusion_matrix, "Confusion Matrix Train Data")

          showConfusionMatrix(test_confusion_matrix, "Confusion Matrix Test Data")
```

```
[[ 3355  5449]
 [ 1484 45964]]
```



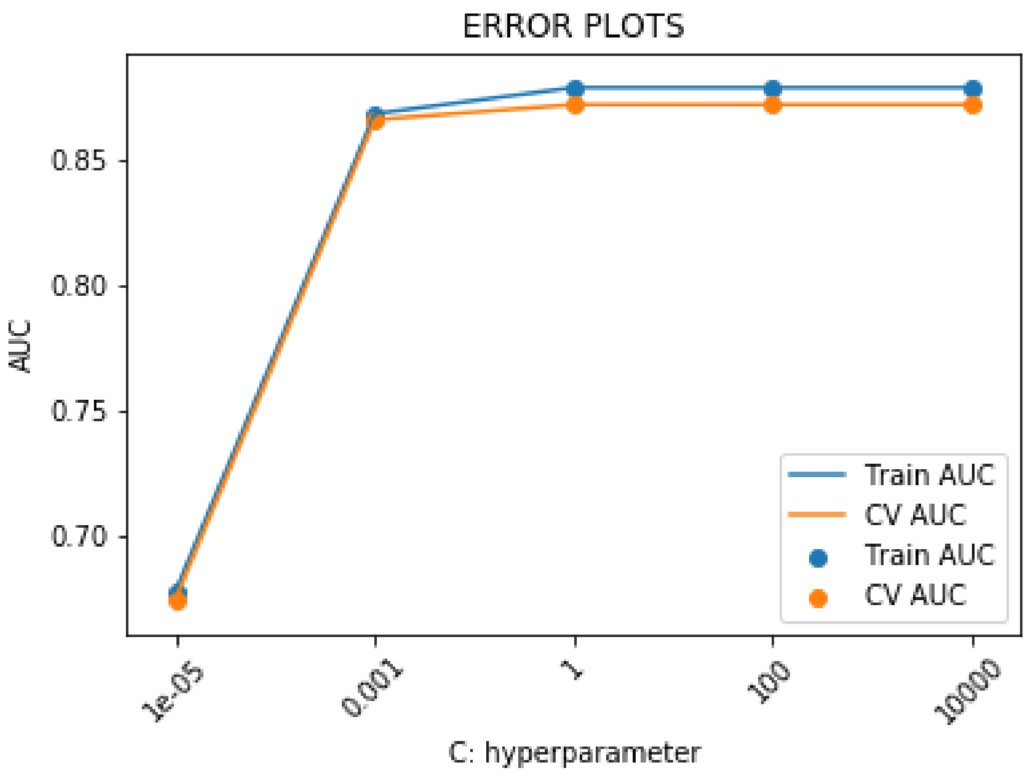

```
[[ 1177  1850]
 [  549 14004]]
```



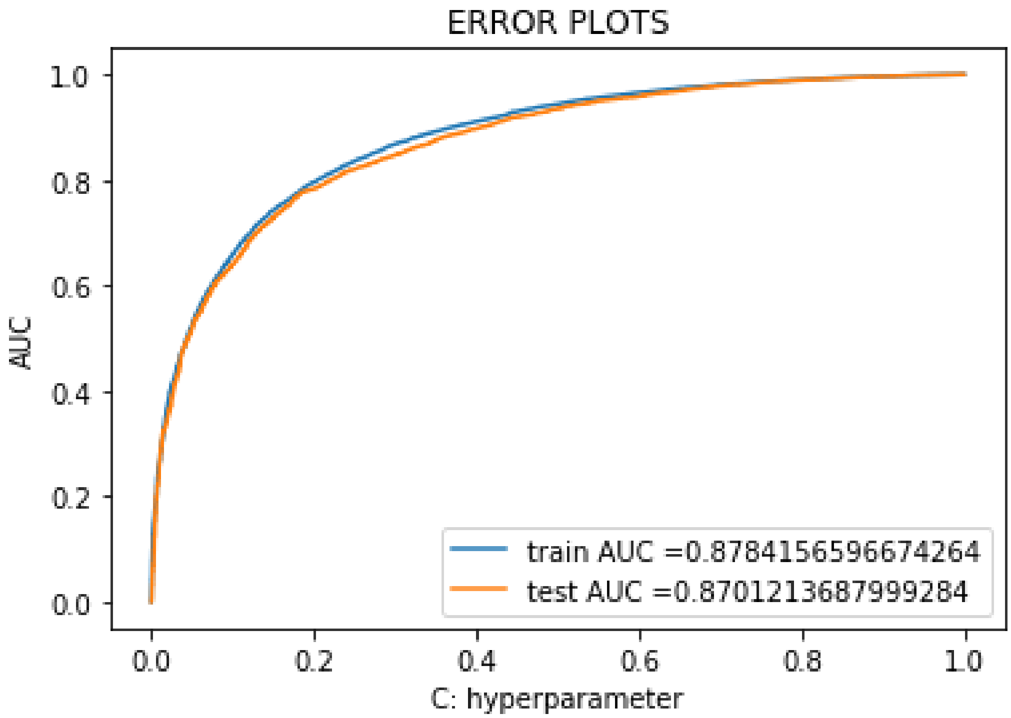
Conclusion : Total misclassified points 2399, Accuracy 86.4%

### 7.4.2 [5.4.2] Applying Logistic Regression with L2 regularization on TFIDF W2V, SET 4

```
In [188]: logRegressor_l2_tfidfW2V = getOptimalLamda(tfidf_sent_vectors_train, y_train, tfidf_s
```

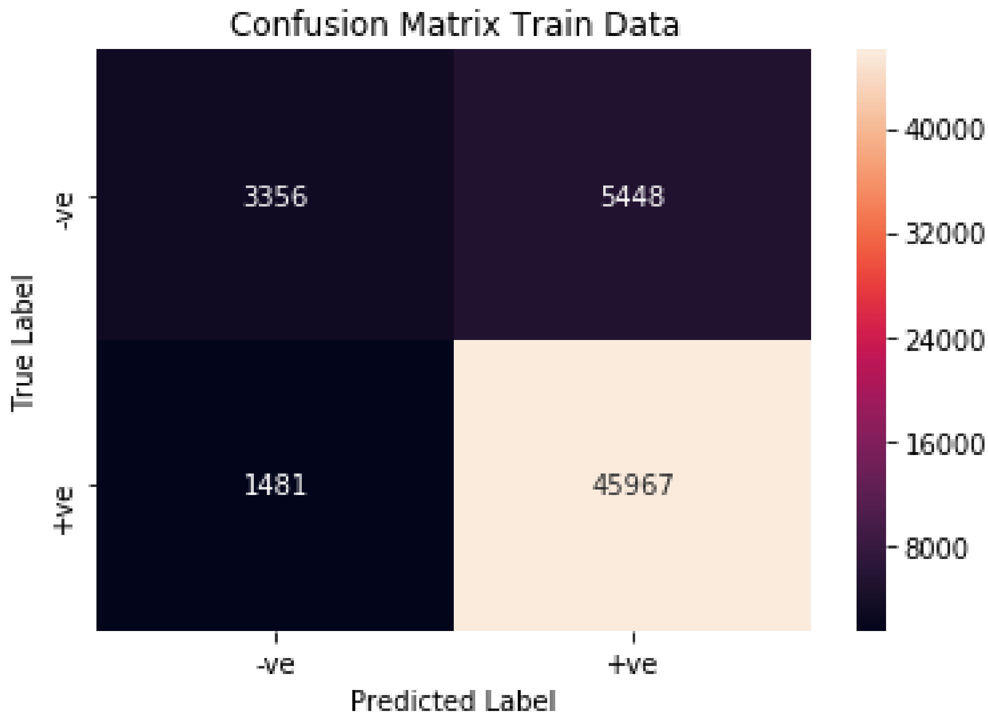



```
In [189]: #Best value of AUC heightst at C=1, 100 and 10000, so we are taking 1
          tfidfW2V_l2_logRegressor, train_confusion_matrix, test_confusion_matrix = getLRAnalys
```

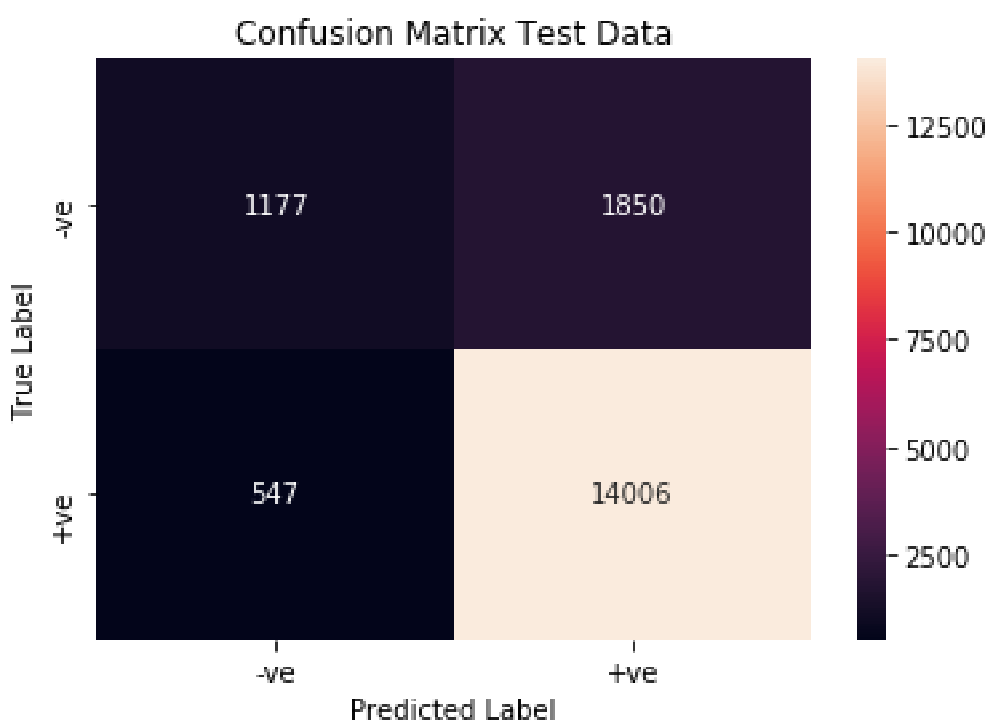



```
In [190]: showConfusionMatrix(train_confusion_matrix, "Confusion Matrix Train Data")

          showConfusionMatrix(test_confusion_matrix, "Confusion Matrix Test Data")
```

```
[[ 3356  5448]
 [ 1481 45967]]
```




```
[[ 1177  1850]
 [  547 14006]]
```



Conclusion : Total misclassified points 1850+547= 2397 and Accuracy = 86.4%

# 8 [6] Conclusions

In [191]:
```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Regulizer", "Hyper parameter", "AUC", "Accuracy"]

x.add_row(["BOW", "l1", 1, 0.93, "91%"])
x.add_row(["BOW", "l2", 1, 0.93, "91.1%"])
x.add_row(["TFIDF", "l1", 1, 0.96, "92.45%"])
x.add_row(["TFIDF", "l2", 1, 0.961, "92%"])
x.add_row(["AVG W2V", "l1", 1, 0.899, "88%"])
x.add_row(["AVG W2V", "l2", 1, 0.899, "88%"])
x.add_row(["TFIDF W2V", "l1", 1, 0.87, "86.4%"])
x.add_row(["TFIDF W2V", "l2", 1, 0.87, "86.4%"])

print(x)
```

```
+------------+-----------+-----------------+-------+----------+
| Vectorizer | Regulizer | Hyper parameter |  AUC  | Accuracy |
+------------+-----------+-----------------+-------+----------+
|    BOW     |     l1    |        1        |  0.93 |   91%    |
|    BOW     |     l2    |        1        |  0.93 |  91.1%   |
|   TFIDF    |     l1    |        1        |  0.96 |  92.45%  |
|   TFIDF    |     l2    |        1        | 0.961 |   92%    |
|  AVG W2V   |     l1    |        1        | 0.899 |   88%    |
|  AVG W2V   |     l2    |        1        | 0.899 |   88%    |
| TFIDF W2V  |     l1    |        1        |  0.87 |  86.4%   |
| TFIDF W2V  |     l2    |        1        |  0.87 |  86.4%   |
+------------+-----------+-----------------+-------+----------+
```

From the above table we can see that model wise TFIDF with l1 or l2 regularization is performing better than the rest. Though all model are equally comparable.