Amazon Fine Food Reviews Analysis

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

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

[1]. Reading Data

[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 [1]:
```
%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
```

```
C:\Users\Admin\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning:
detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [2]:
```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# 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 p
# 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 LIM
# for tsne assignment you can take 5k data points

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

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a neg
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[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominat
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	

In [3]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [4]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [5]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...	5

In [6]:
```
display['COUNT(*)'].sum()
```

Out[6]:
```
393063
```

[2] Exploratory Data Analysis

[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 [7]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomir
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	

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 [8]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace
```

In [9]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, k
final.shape
```

Out[9]: (87775, 10)

In [10]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[10]: 87.775

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 [11]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomir
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	

In [12]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [13]:
```
#Before starting the next phase of preprocessing lets see the number of entries le
print(final.shape)

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

```
(87773, 10)
```

Out[13]:
```
1    73592
0    14181
Name: Score, dtype: int64
```

[3] Preprocessing

[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 [14]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying
it anymore.  Its very hard to find any chicken products made in the USA but th
ey are out there, but this one isnt.  Its too bad too because its a good produ
ct but I wont take any chances till they know what is going on with the china
imports.
==================================================
```

In [15]:
```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)

print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying
it anymore.  Its very hard to find any chicken products made in the USA but th
ey are out there, but this one isnt.  Its too bad too because its a good produ
ct but I wont take any chances till they know what is going on with the china
imports.
```

In [16]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, '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 very hard to find any chicken products made in the USA but th
ey are out there, but this one isnt.  Its too bad too because its a good produ
ct but I wont take any chances till they know what is going on with the china
imports.
```

In [17]:
```
# 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 [18]:
```
# sent_500 = decontracted(sent_500)
# print(sent_500)
# print("="*50)
```

In [19]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
# sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
# print(sent_0)
```

In [20]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
# sent_500 = re.sub('[^A-Za-z0-9]+', ' ', sent_500)
# print(sent_500)
```

In [21]:
```python
# 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 ste

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ours
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'h
            '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',
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'thr
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off'
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than',
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've"
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "did
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't
            'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:
```python
# 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 st
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████████████████████████████| 87773/87773 [00:46<00:00, 1887.75it/
s]
```

Train & Test Split

In [25]:
```
from sklearn.model_selection import train_test_split

final['Text'] = preprocessed_reviews

# Created new feature TextLength (Length of Review) in our preprocessed data
final['TextLength'] = final['Text'].str.len()

X = final
Y = final['Score'].values

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)

print(X_train.shape, Y_train.shape)
print(X_test.shape, Y_test.shape)
```

```
(61441, 11) (61441,)
(26332, 11) (26332,)
```

[4] Featurization

[4.1] BAG OF WORDS

In [26]:
```
from scipy.sparse import hstack, coo_matrix
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(X_train['Text'])
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

print(X_train.shape)
X_train_bow = count_vect.transform(X_train['Text'])
X_test_bow = count_vect.transform(X_test['Text'])

# Adding new feature Review Length to our featurized train and test data
X_train_bow=hstack([X_train_bow,np.matrix(X_train['TextLength'].values).reshape(X_
X_test_bow=hstack([X_test_bow,np.matrix(X_test['TextLength'].values).reshape(X_tes

# Adding new feature Review Length to list of features
feature_bow = count_vect.get_feature_names()
feature_bow.append('TextLength')

print("the type of count vectorizer ",type(X_train_bow))
print("the shape of out text BOW vectorizer ",X_train_bow.get_shape())
print("the number of unique words ", X_train_bow.get_shape()[1])
```

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaaaaaaaa', 'aaaaaaaaaa
aaaaa', 'aaaaaaahhhhhh', 'aaaaaaarrrrrggghhh', 'aaaaaawwwwwwwwwww', 'aaaaah']
==================================================
(61441, 11)
the type of count vectorizer  <class 'scipy.sparse.coo.coo_matrix'>
the shape of out text BOW vectorizer  (61441, 46284)
the number of unique words  46284
```

[4.2] Bi-Grams and n-Grams.

[4.3] TF-IDF

In [27]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train['Text'])
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_n
print('='*50)

X_train_tfidf = tf_idf_vect.transform(X_train['Text'])
X_test_tfidf = tf_idf_vect.transform(X_test['Text'])

# Adding new feature Review Length to our featurized train and test data
X_train_tfidf=hstack([X_train_tfidf,np.matrix(X_train['TextLength'].values).reshap
X_test_tfidf=hstack([X_test_tfidf,np.matrix(X_test['TextLength'].values).reshape(X

# Adding new feature Review Length to list of features
feature_tfidf = tf_idf_vect.get_feature_names()
feature_tfidf.append('TextLength')

print("the type of count vectorizer ",type(X_train_tfidf))
print("the shape of out text TFIDF vectorizer ",X_train_tfidf.get_shape())
print("the number of unique words including both unigrams and bigrams ", X_train_t
```

```
some sample features(unique words in the corpus) ['aa', 'aback', 'abandoned',
'ability', 'able', 'able buy', 'able chew', 'able drink', 'able eat', 'able en
joy']
==================================================
the type of count vectorizer  <class 'scipy.sparse.coo.coo_matrix'>
the shape of out text TFIDF vectorizer  (61441, 36182)
the number of unique words including both unigrams and bigrams  36182
```

[4.4] Word2Vec

In [28]:
```
# Train your own Word2Vec model using your own text corpus
list_of_sentance_train=[]
for sentance in X_train['Text']:
    list_of_sentance_train.append(sentance.split())

w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=2)
```

In [29]:
```
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  14747
sample words  ['like', 'flavor', 'mix', 'best', 'crystal', 'light', 'beverage
s', 'pleasantly', 'sweet', 'tart', 'great', 'good', 'person', 'go', 'wrong',
'combination', 'sometimes', 'use', 'baking', 'often', 'enjoy', 'eating', 'squa
re', 'right', 'bag', 'recipe', 'calls', 'chopped', 'candied', 'crystallized',
'ginger', 'need', 'cut', 'smaller', 'pieces', 'slap', 'chopper', 'works', 'wel
l', 'get', 'really', 'gummy', 'trying', 'chop', 'large', 'quantity', 'chips',
'much', 'easier', 'also']
```

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

[4.4.1.1] Avg W2v

In [30]:
```
# average Word2Vec
# compute average word2vec for each review.
def avgw2v(list_of_sentance):
    sent_vectors = []; # the avg-w2v for each sentence/review is stored in this li
    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
        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)
    return sent_vectors
```

In [31]:
```
sent_vectors_train = avgw2v(list_of_sentance_train)
print(len(sent_vectors_train[0]))
print(len(list_of_sentance_train))
```

```
100%|██████████████████████████████| 61441/61441 [02:48<00:00, 364.32it/
s]

50
61441
```

In [32]:
```
list_of_sentance_test=[]
for sentance in X_test['Text']:
    list_of_sentance_test.append(sentance.split())

sent_vectors_test = avgw2v(list_of_sentance_test)
print(len(sent_vectors_test))
print(len(sent_vectors_test[0]))
```

```
100%|██████████████████████████████| 26332/26332 [01:23<00:00, 315.84it/
s]

26332
50
```

[4.4.1.2] TFIDF weighted W2v

In [33]:
```
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tf_idf_vect.get_feature_names(), list(tf_idf_vect.idf_)))
```

In [34]:
```
# TF-IDF weighted Word2Vec
def tfidfw2v(list_of_sentance):
    tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
    # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val

    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in
    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
    return tfidf_sent_vectors
```

In [35]:
```
tfidf_sent_vectors_train = tfidfw2v(list_of_sentance_train)
```

```
100%|██████████████████████████████| 61441/61441 [42:40<00:00, 24.00it/
s]
```

In [36]:
```
tfidf_sent_vectors_test = tfidfw2v(list_of_sentance_test)
```

```
100%|██████████████████████████████| 26332/26332 [18:28<00:00, 23.76it/
s]
```

Applying Decision Trees

[5.1] Applying Decision Trees on BOW, SET 1

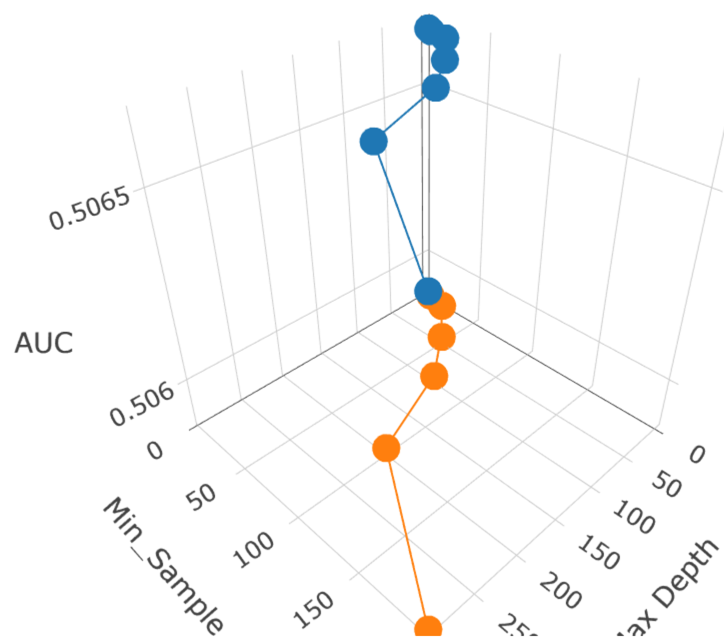
In [37]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
# plot 3D Plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
```

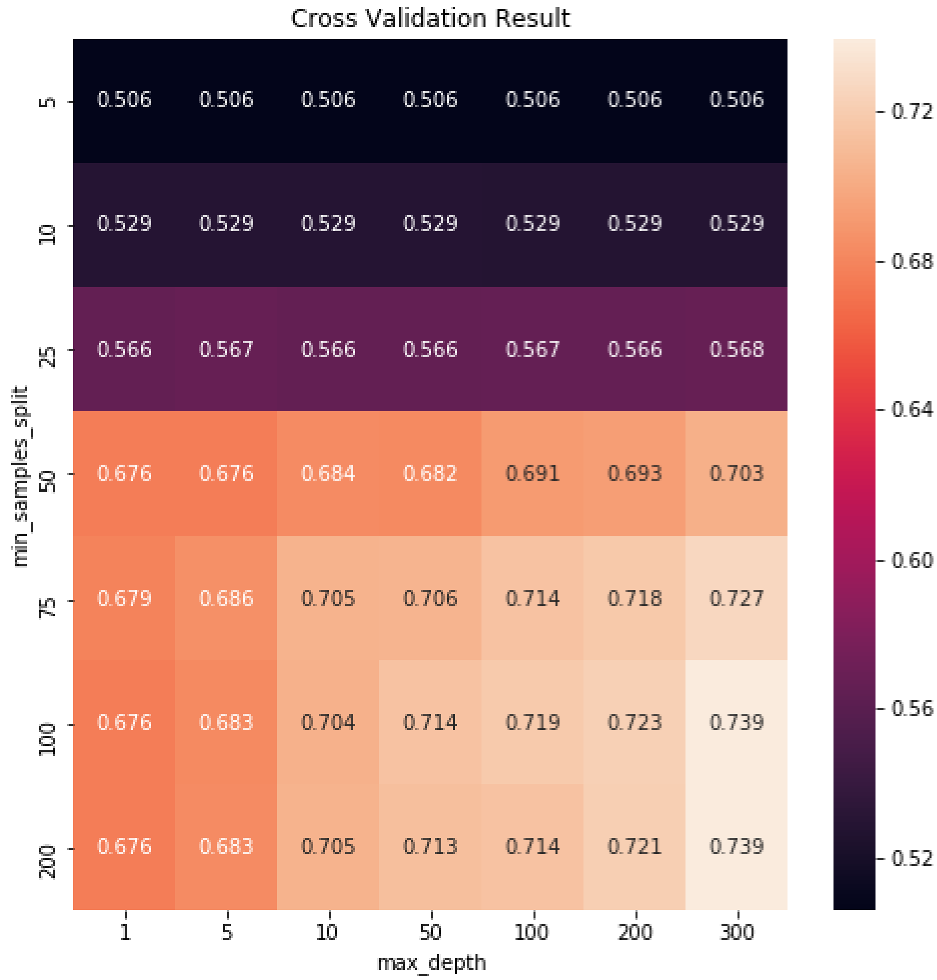
In [38]:
```
# Plot 3rd graph for hyperparameters and AUC of train and test results
def D3_Plot(max_depth, min_sample_split, train_auc, test_auc):

    # https://plot.ly/python/3d-axes/
    train = go.Scatter3d(x=max_depth,y=min_sample_split,z=train_auc, name = 'train
    test = go.Scatter3d(x=max_depth,y=min_sample_split,z=test_auc, name = 'Cross v
    data = [train, test]

    layout = go.Layout(scene = dict(
            xaxis = dict(title='Max Depth'),
            yaxis = dict(title='Min_Sample_Split'),
            zaxis = dict(title='AUC'),))

    fig = go.Figure(data=data, layout=layout)
    offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [39]:
```
def Get_Hyperparameter(X_train_vector):
    max_depth =  [1, 5, 10, 50, 100, 200, 300]
    min_samples_split = [5, 10, 25, 50, 75, 100, 200]
    parameters = { 'max_depth' : max_depth, 'min_samples_split' : min_samples_spli

    clf = DecisionTreeClassifier(class_weight='balanced', random_state=0, max_feat

    grid = GridSearchCV(clf, parameters, cv=10, scoring='roc_auc')
    grid.fit(X_train_vector, Y_train)

    print("Best Estimator: ",grid.best_estimator_)
    print("Best cross-validation score: {:.2f}".format(grid.best_score_))  # best
    print("Best hyperparameters: ", grid.best_params_)

    best_depth = grid.best_params_['max_depth']  # best max_depth value after 10 f
    best_min_samples_split = grid.best_params_['min_samples_split']  # best min sp

    train_auc= grid.cv_results_['mean_train_score']
    cv_auc = grid.cv_results_['mean_test_score']

    # Plot 3rd graph for hyperparameters and AUC of train and test results
    D3_Plot(max_depth, min_samples_split, train_auc, cv_auc)

    # Transform CV results(list of 49) to 7X7
    test_score = cv_auc.reshape(len(max_depth), len(min_samples_split))

    # Creating HeatMap to display best hyperparameter value
    plt.figure(figsize=(8,8))
    sns.heatmap(test_score, annot=True, fmt=".3f", xticklabels=max_depth, yticklab
    plt.xlabel("max_depth")
    plt.ylabel("min_samples_split")
    plt.title("Cross Validation Result")
    plt.show()

    return best_depth, best_min_samples_split
```

In [40]:
```
best_max_depth_bow, best_min_samples_split_bow = Get_Hyperparameter(X_train_bow)
```

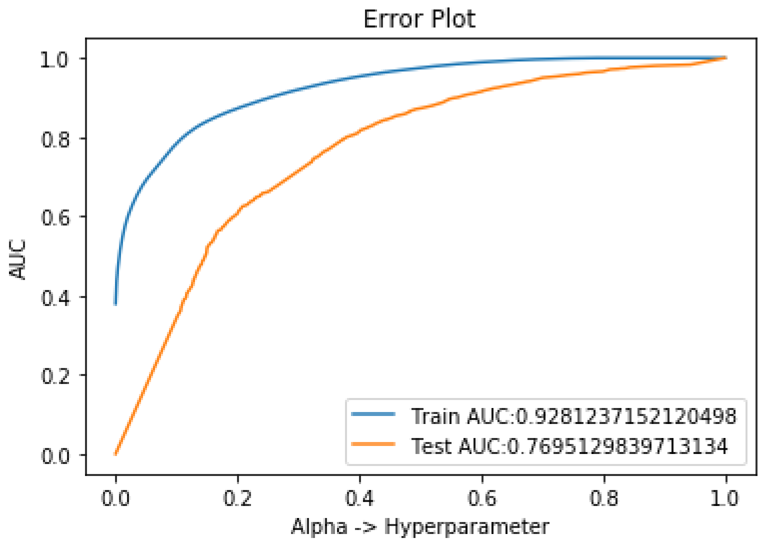
```
Best Estimator:  DecisionTreeClassifier(class_weight='balanced', criterion='gi
ni',
            max_depth=300, max_features='sqrt', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=200,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
Best cross-validation score: 0.74
Best hyperparameters:  {'max_depth': 300, 'min_samples_split': 200}
```

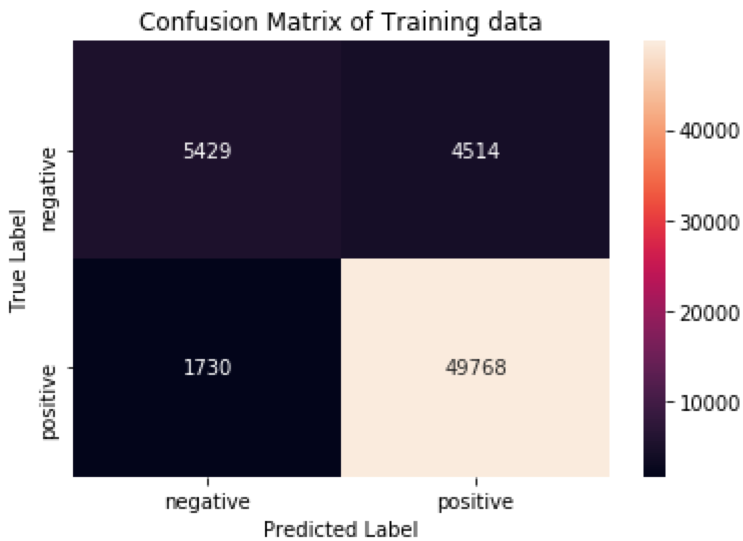


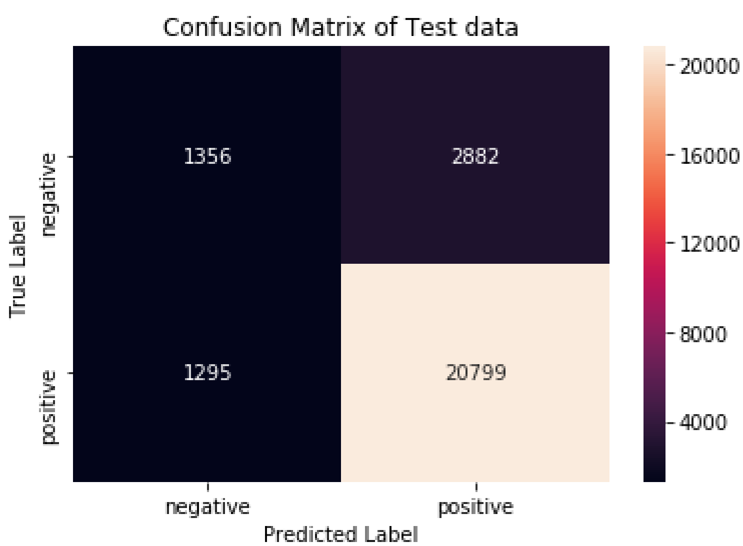

Cross Validation Result
min_samples_split
max_depth
5
10
25
50
75
100
200
1
5
10
50
100
200
300
0.506 0.506 0.506 0.506 0.506 0.506 0.506
0.529 0.529 0.529 0.529 0.529 0.529 0.529
0.566 0.567 0.566 0.566 0.567 0.566 0.568
0.676 0.676 0.684 0.682 0.691 0.693 0.703
0.679 0.686 0.705 0.706 0.714 0.718 0.727
0.676 0.683 0.704 0.714 0.719 0.723 0.739
0.676 0.683 0.705 0.713 0.714 0.721 0.739
0.72
0.68
0.64
0.60
0.56
0.52

In [41]:
```
def DT_test(X_train_vector, X_test_vector, best_max_depth, best_min_samples_split)
    clf = DecisionTreeClassifier(max_depth=best_max_depth, min_samples_split=best_
                                 max_features ='sqrt')
    clf.fit(X_train_vector, Y_train)

    # Get ROC Curve
    train_fpr, train_tpr, threshold = roc_curve(Y_train, clf.predict_proba(X_train
    test_fpr, test_tpr, threshold = roc_curve(Y_test, clf.predict_proba(X_test_vec

    # Get AUC using FPR and TPR
    test_auc = auc(test_fpr, test_tpr)
    plt.plot(train_fpr, train_tpr, label = "Train AUC:"+str(auc(train_fpr,train_tp
    plt.plot(test_fpr, test_tpr, label = "Test AUC:"+str(test_auc))

    plt.legend()
    plt.xlabel("Alpha -> Hyperparameter")
    plt.ylabel("AUC")
    plt.title("Error Plot")
    plt.show()

    # plot confusion matrix to describe the performance of classifier.
    class_label = ["negative", "positive"]
    df_cm_train = pd.DataFrame(confusion_matrix(Y_train, clf.predict(X_train_vecto
    sns.heatmap(df_cm_train, annot = True, fmt = "d")
    plt.title("Confusion Matrix of Training data")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()

    df_cm_test = pd.DataFrame(confusion_matrix(Y_test, clf.predict(X_test_vector))
    sns.heatmap(df_cm_test, annot = True, fmt = "d")
    plt.title("Confusion Matrix of Test data")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()

    return test_auc, clf
```

In [42]:
```
auc_bow, clf_bow = DT_test(X_train_bow, X_test_bow, best_max_depth_bow, best_min_s
```







[5.1.1] Top 20 important features from SET 1

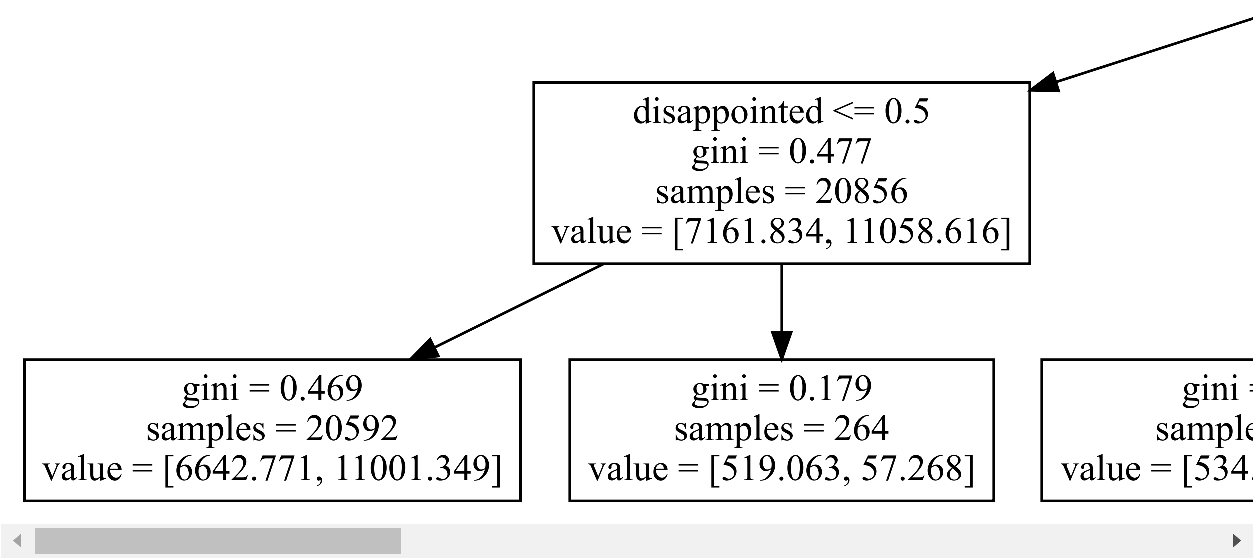
In [43]:
```
# Print top 20 important features

n=20
# features = count_vect.get_feature_names()
coefs = sorted(zip(clf_bow.feature_importances_, feature_bow))
# [start: end: reverse]  --> [: -21: -1] --> display features from -1 to -20
top = coefs[:-(n + 1):-1]

print("Top Features")
for (coef1, feat1) in top:
    print("%.4f\t%s" % (coef1, feat1))
```

```
Top Features
0.0390	horrible
0.0332	bad
0.0306	disappointed
0.0272	refund
0.0176	great
0.0128	awful
0.0101	not
0.0081	worst
0.0080	wonderful
0.0079	poor
0.0075	threw
0.0071	disappointment
0.0068	love
0.0068	trash
0.0067	label
0.0065	perfect
0.0056	disappointing
0.0053	unless
0.0051	ok
0.0049	delicious
```

[5.1.2] Graphviz visualization of Decision Tree on BOW, SET 1

In [44]:
```
import graphviz
from sklearn import tree
from graphviz import Source

clf = DecisionTreeClassifier(max_depth=3, class_weight='balanced', random_state=0)
clf.fit(X_train_bow, Y_train)

Source(tree.export_graphviz(clf, out_file = None, feature_names = feature_bow))
```

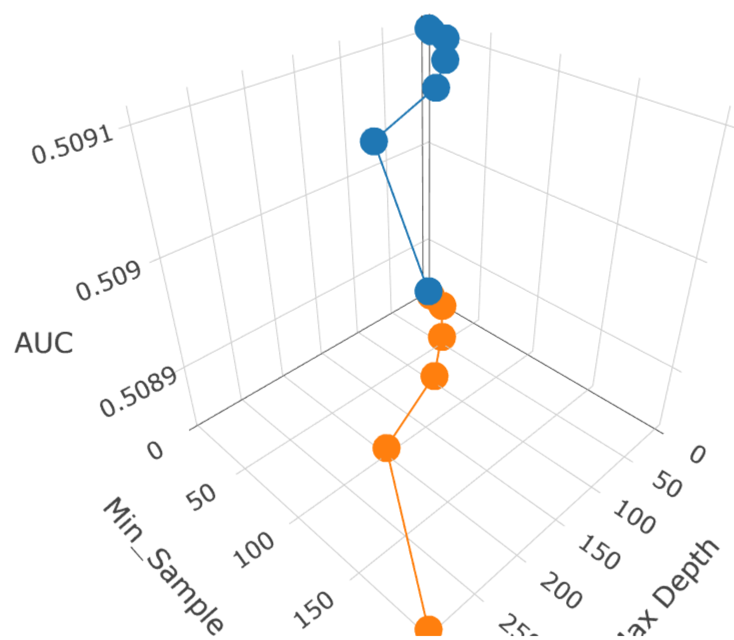
Out[44]:

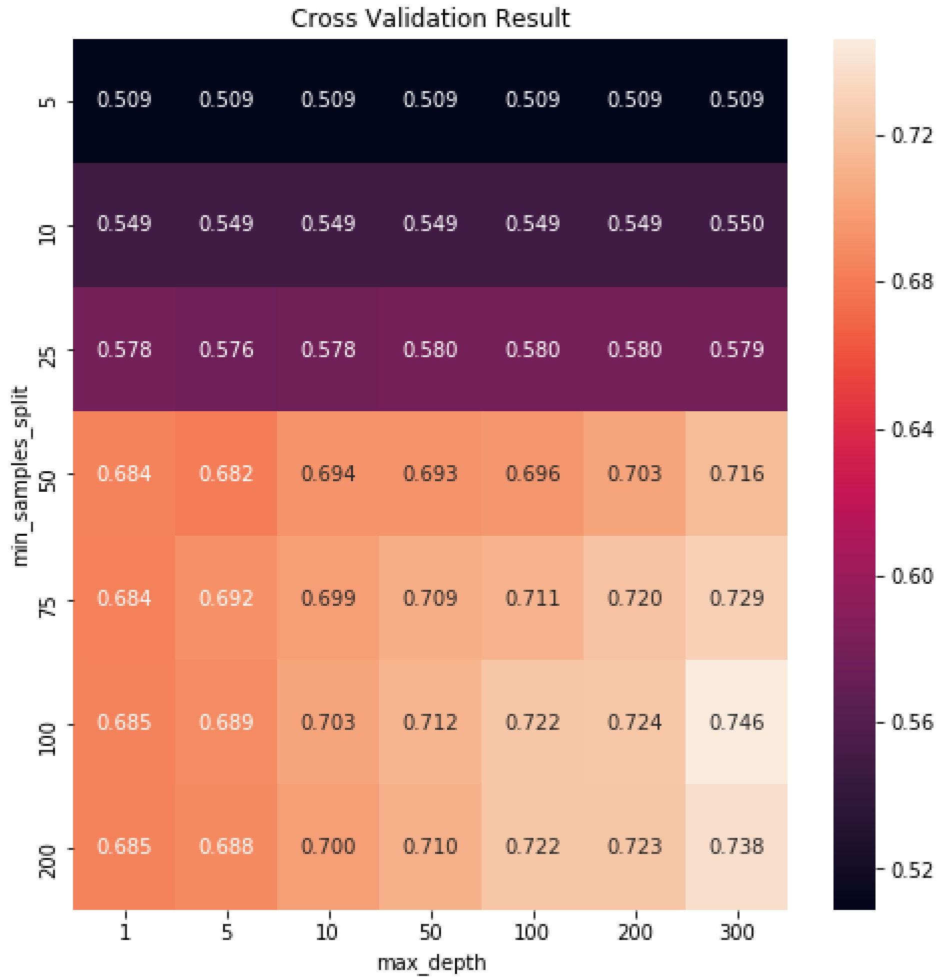


[5.2] Applying Decision Trees on TFIDF, SET 2

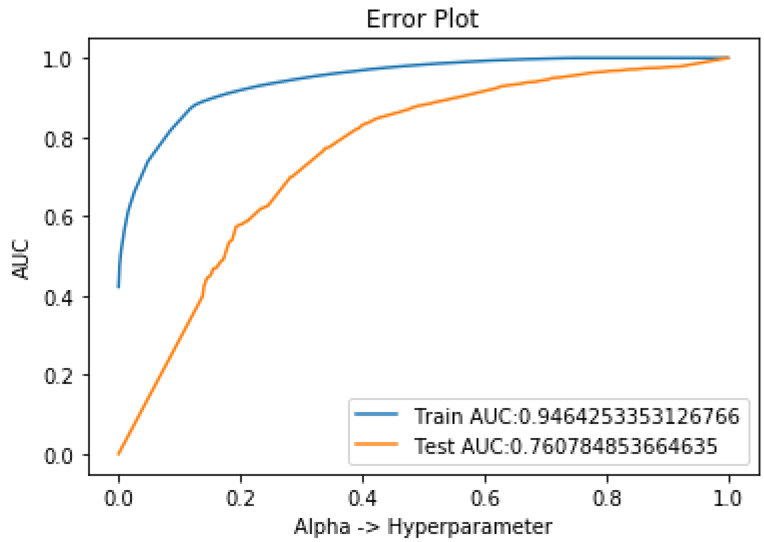
In [45]:
```
best_max_depth_tfidf, best_min_samples_split_tfidf = Get_Hyperparameter(X_train_tf
```

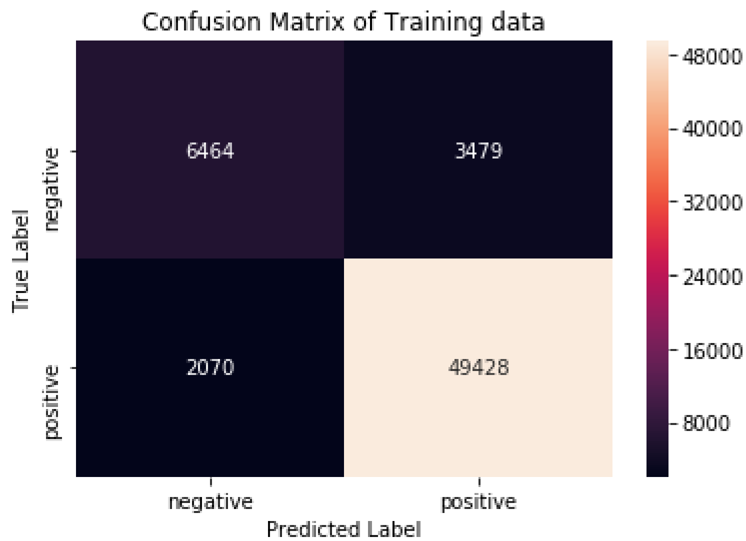
```
Best Estimator:  DecisionTreeClassifier(class_weight='balanced', criterion='gi
ni',
            max_depth=200, max_features='sqrt', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=200,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
Best cross-validation score: 0.75
Best hyperparameters:  {'max_depth': 200, 'min_samples_split': 200}
```

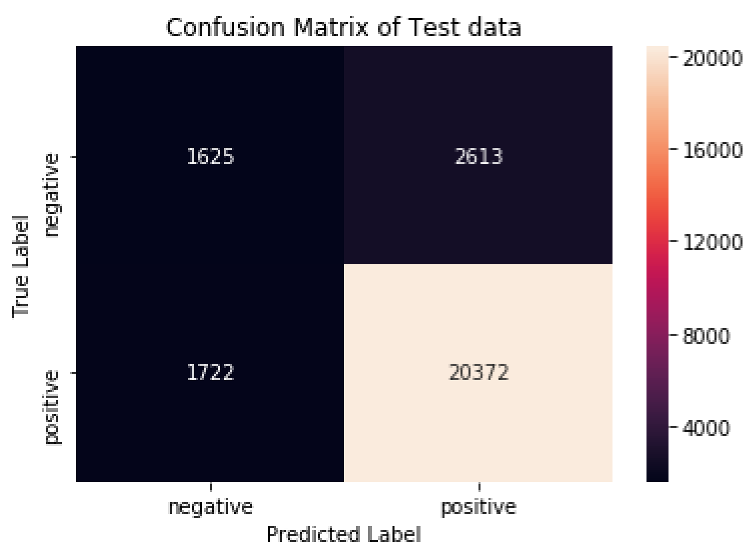



Cross Validation Result
min_samples_split
max_depth
5
10
25
50
75
100
200
1
5
10
50
100
200
300
0.509 0.509 0.509 0.509 0.509 0.509 0.509
0.549 0.549 0.549 0.549 0.549 0.549 0.550
0.578 0.576 0.578 0.580 0.580 0.580 0.579
0.684 0.682 0.694 0.693 0.696 0.703 0.716
0.684 0.692 0.699 0.709 0.711 0.720 0.729
0.685 0.689 0.703 0.712 0.722 0.724 0.746
0.685 0.688 0.700 0.710 0.722 0.723 0.738
0.72
0.68
0.64
0.60
0.56
0.52

```
In [46]: auc_tfidf, clf_tfidf = DT_test(X_train_tfidf, X_test_tfidf, best_max_depth_tfidf,
```







[5.2.1] Top 20 important features from SET 2

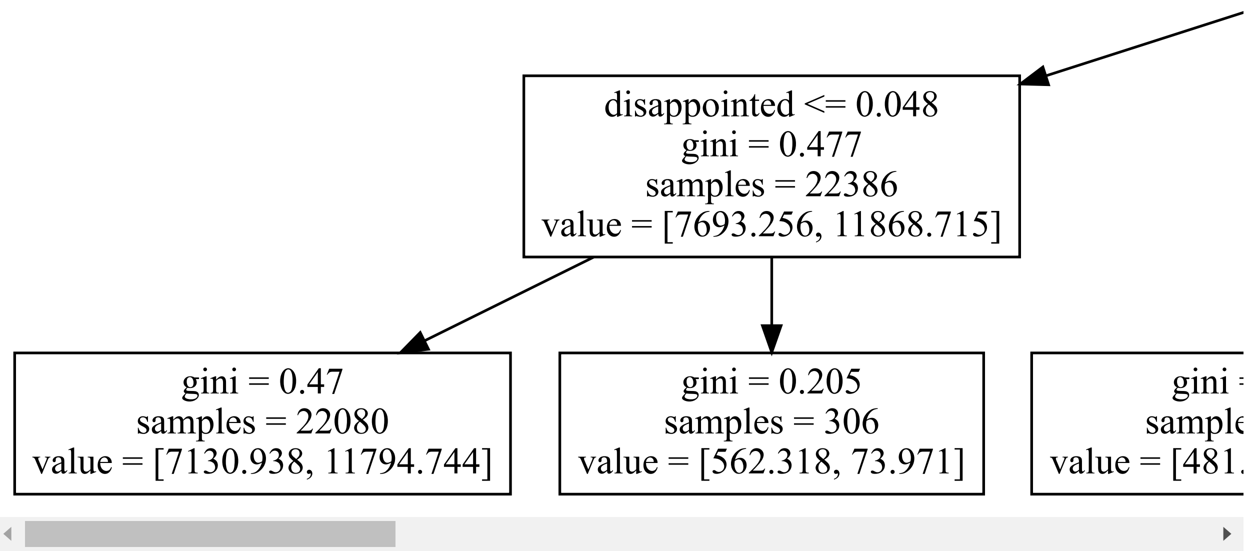
In [47]:
```
n=20
coefs = sorted(zip(clf_tfidf.feature_importances_, feature_tfidf))
top = coefs[:-(n + 1):-1]

print("Top Features")
for (coef1, feat1) in top:
    print("%.4f\t%s" % (coef1, feat1))
```

```
Top Features
0.0275  worst
0.0163  disappointing
0.0146  disgusting
0.0143  refund
0.0141  awful
0.0117  description
0.0106  not
0.0094  could
0.0091  disappointed
0.0084  not buy
0.0077  not good
0.0071  shame
0.0066  date
0.0063  not order
0.0062  stick
0.0055  save money
0.0054  high hopes
0.0054  would
0.0051  item
0.0048  great
```

[5.2.2] Graphviz visualization of Decision Tree on TFIDF, SET 2

In [48]:
```
clf = DecisionTreeClassifier(max_depth=3, class_weight='balanced', random_state=0)
clf.fit(X_train_tfidf, Y_train)

Source(tree.export_graphviz(clf, out_file = None, feature_names = feature_tfidf))
```

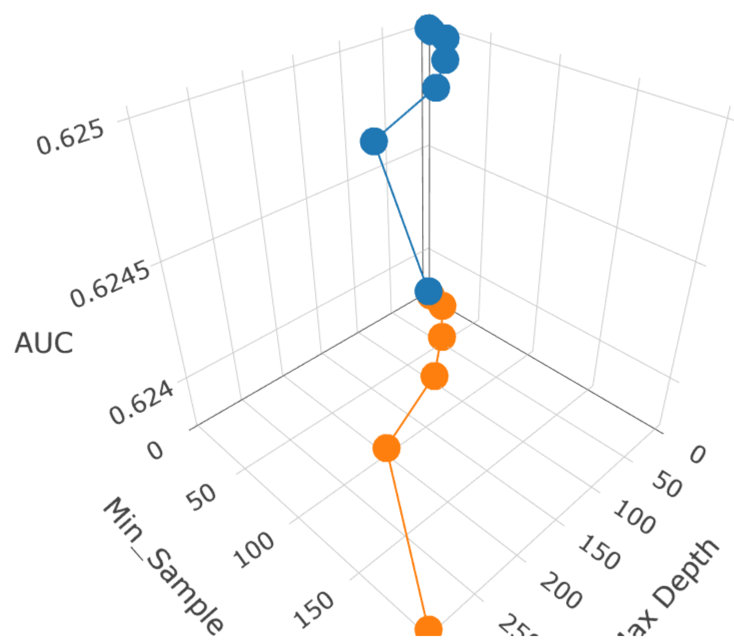
Out[48]:

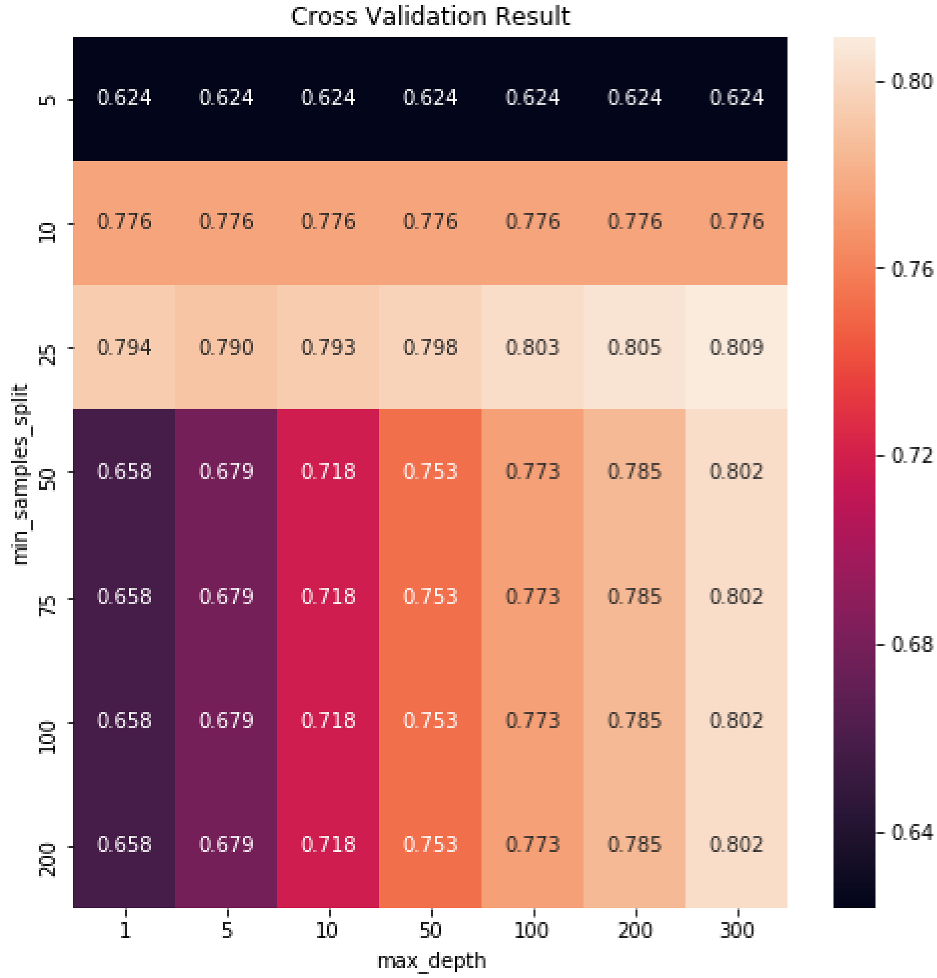


[5.3] Applying Decision Trees on AVG W2V, SET 3

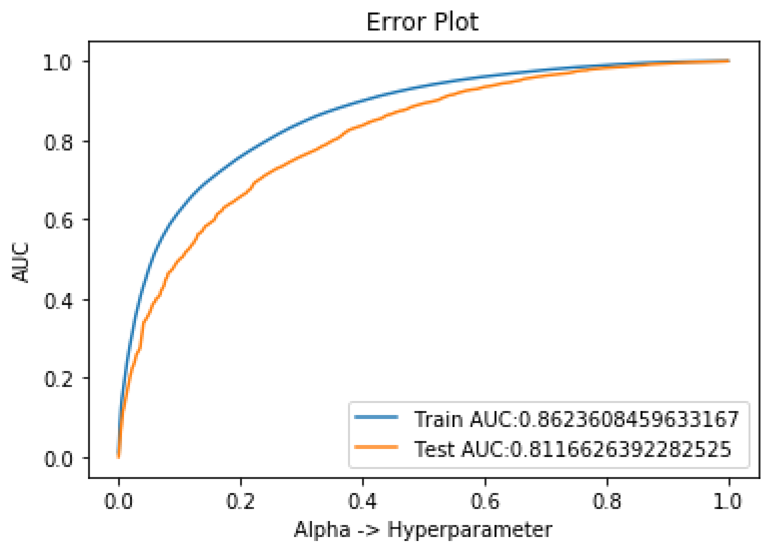
In [49]:
```
best_max_depth_avgw2v, best_min_samples_split_avgw2v = Get_Hyperparameter(sent_vec
```

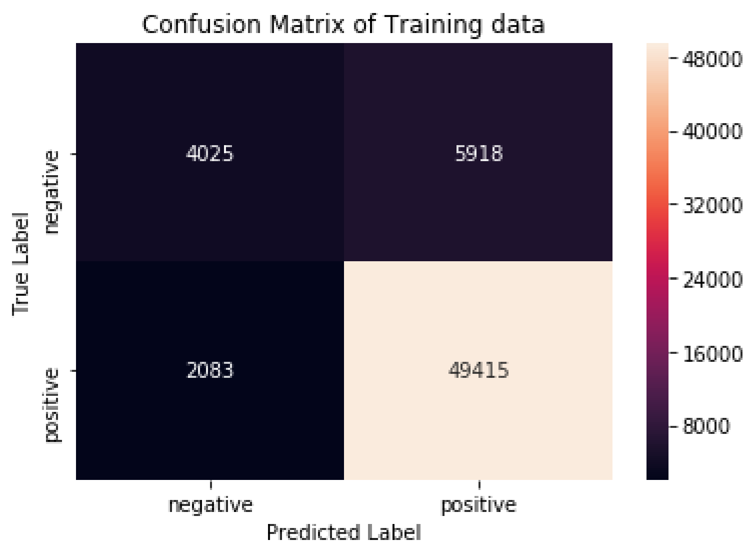
```
Best Estimator:  DecisionTreeClassifier(class_weight='balanced', criterion='gi
ni',
            max_depth=10, max_features='sqrt', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=200,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
Best cross-validation score: 0.81
Best hyperparameters:  {'max_depth': 10, 'min_samples_split': 200}
```

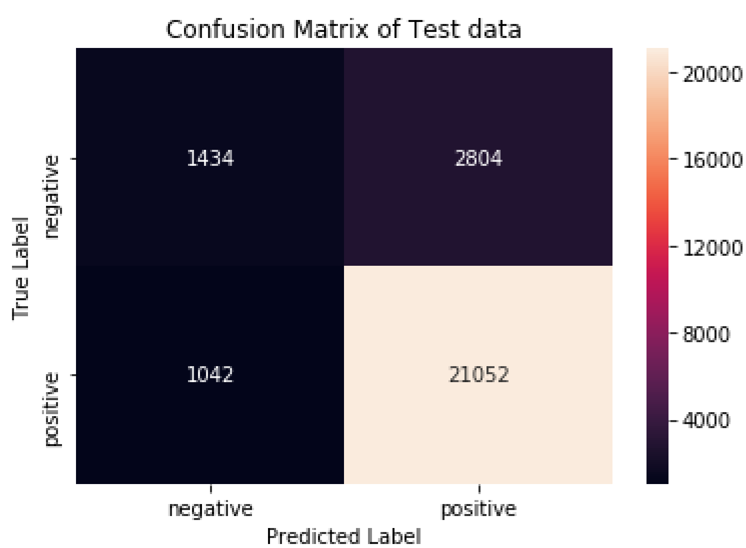



Cross Validation Result
min_samples_split
5
10
25
50
75
100
200
max_depth
1
5
10
50
100
200
300
0.624 0.624 0.624 0.624 0.624 0.624 0.624
0.776 0.776 0.776 0.776 0.776 0.776 0.776
0.794 0.790 0.793 0.798 0.803 0.805 0.809
0.658 0.679 0.718 0.753 0.773 0.785 0.802
0.658 0.679 0.718 0.753 0.773 0.785 0.802
0.658 0.679 0.718 0.753 0.773 0.785 0.802
0.658 0.679 0.718 0.753 0.773 0.785 0.802
0.80
0.76
0.72
0.68
0.64

In [50]:
```
auc_avgw2v, clf_avgw2v = DT_test(sent_vectors_train, sent_vectors_test, best_max_d
```



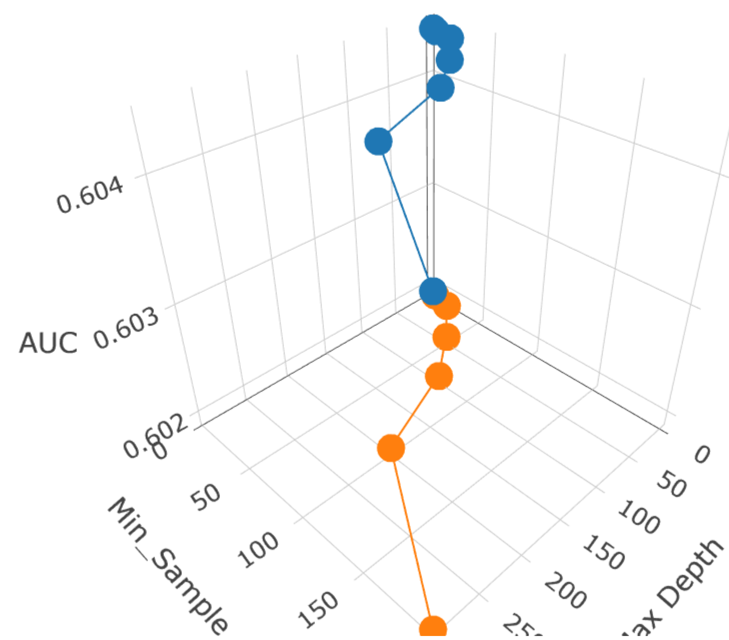


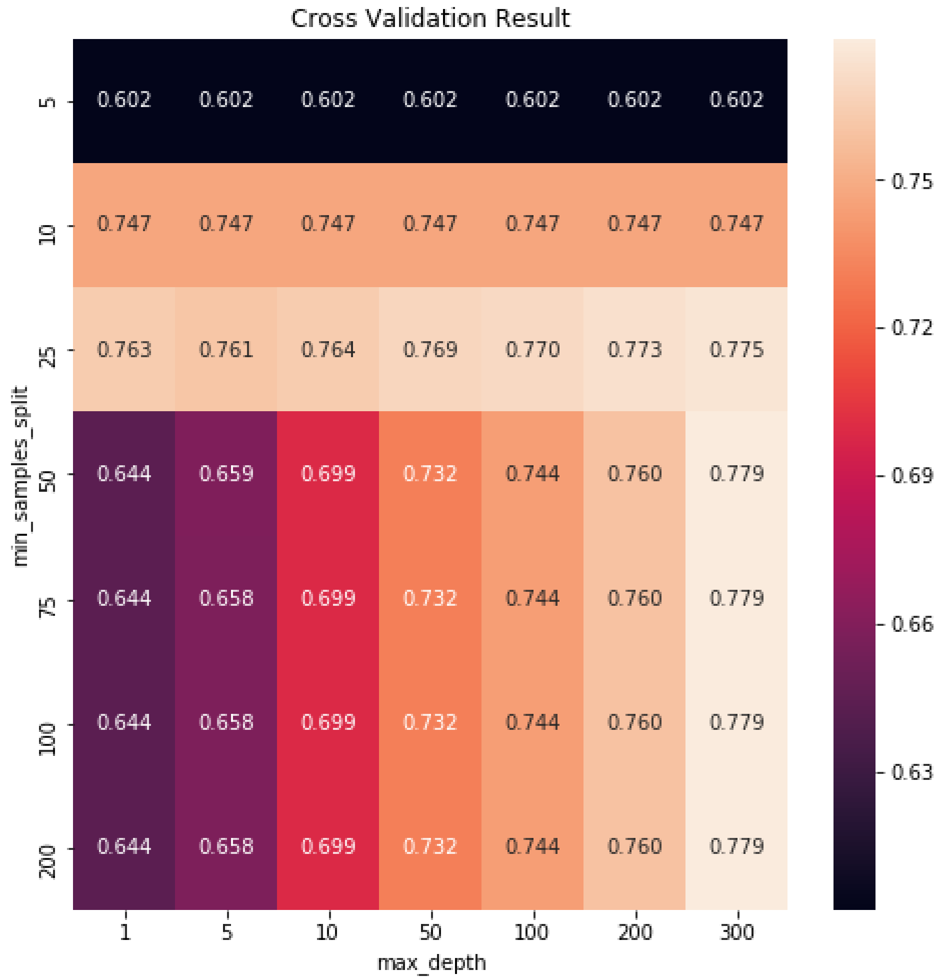


[5.4] Applying Decision Trees on TFIDF W2V, SET 4

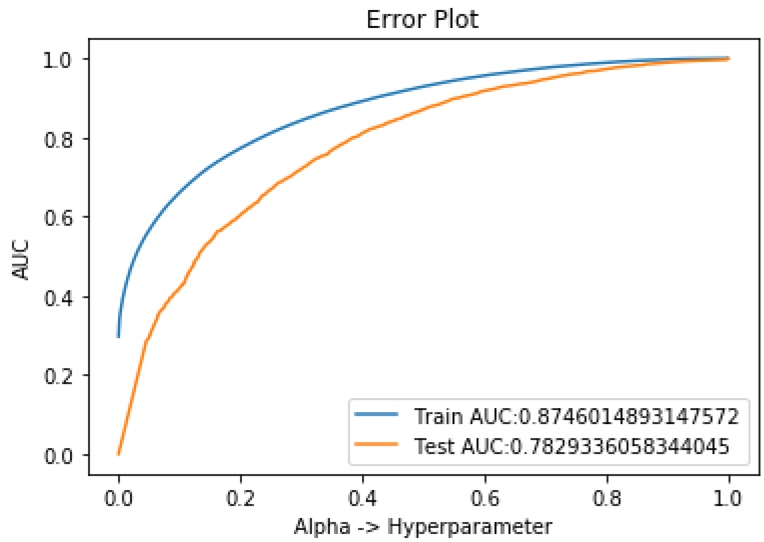
In [51]:
```
best_max_depth_tfidfw2v, best_min_samples_split_tfidfw2v = Get_Hyperparameter(tfid
```

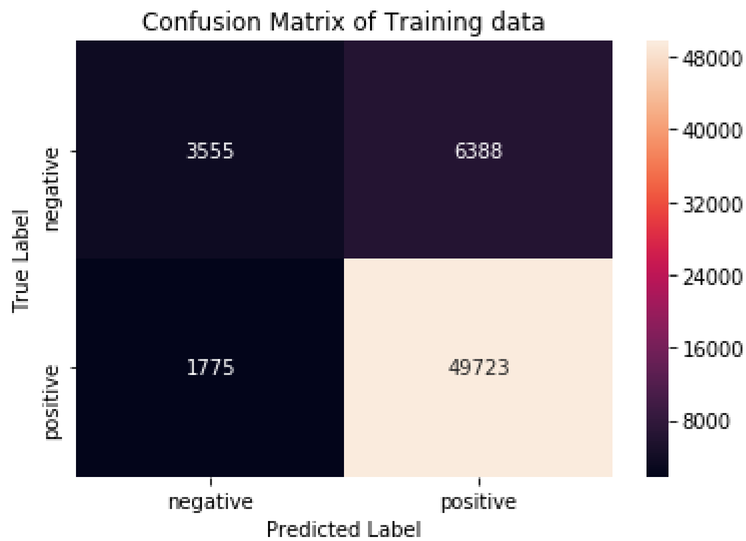
```
Best Estimator:  DecisionTreeClassifier(class_weight='balanced', criterion='gi
ni',
            max_depth=50, max_features='sqrt', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=200,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
Best cross-validation score: 0.78
Best hyperparameters:  {'max_depth': 50, 'min_samples_split': 200}
```

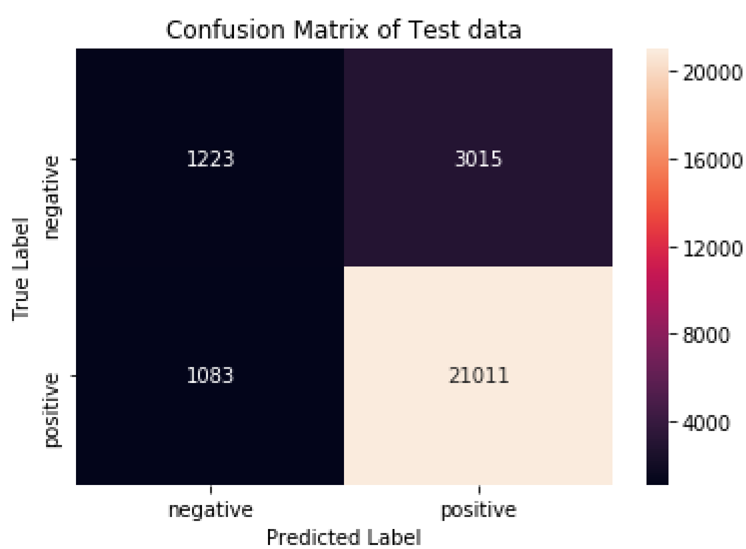



Cross Validation Result
min_samples_split
5
10
25
50
75
100
200
max_depth
1
5
10
50
100
200
300
0.602 0.602 0.602 0.602 0.602 0.602 0.602
0.747 0.747 0.747 0.747 0.747 0.747 0.747
0.763 0.761 0.764 0.769 0.770 0.773 0.775
0.644 0.659 0.699 0.732 0.744 0.760 0.779
0.644 0.658 0.699 0.732 0.744 0.760 0.779
0.644 0.658 0.699 0.732 0.744 0.760 0.779
0.644 0.658 0.699 0.732 0.744 0.760 0.779
0.75
0.72
0.69
0.66
0.63

In [52]:
```
auc_tfidfw2v, clf_tfidfw2v = DT_test(tfidf_sent_vectors_train, tfidf_sent_vectors_
                                     best_max_depth_tfidfw2v, best_min_samples_spl
```







[6] Conclusions

In [53]:
```
models = pd.DataFrame({
'Vectorizer': ["BOW", "TFIDF", "AVGW2V", "TFIDFW2V"],
'Model' :     ['DecisionTree', 'DecisionTree', 'DecisionTree', 'DecisionTree'],
'HyperPara(Depth, Sample_Split)': [{'Depth': best_max_depth_bow, 'Split': best_min
                                   {'Depth': best_max_depth_tfidf, 'Split': best_m
                                   {'Depth': best_max_depth_avgw2v, 'Split': best_
                                   {'Depth': best_max_depth_tfidfw2v, 'Split': bes
'AUC':       [auc_bow, auc_tfidf, auc_avgw2v, auc_tfidfw2v]},
columns =    ["Vectorizer", "Model", "HyperPara(Depth, Sample_Split)", "AUC"])
print(models)
```

```
  Vectorizer         Model HyperPara(Depth, Sample_Split)       AUC
0        BOW  DecisionTree   {'Depth': 300, 'Split': 200}  0.769513
1      TFIDF  DecisionTree   {'Depth': 200, 'Split': 200}  0.760785
2     AVGW2V  DecisionTree    {'Depth': 10, 'Split': 200}  0.811663
3   TFIDFW2V  DecisionTree    {'Depth': 50, 'Split': 200}  0.782934
```

- Done the featurization of reviews using BOW, TFIDF, AVGW2V and TFIDF-W2V
- Created new feature of Review Length
- Define a function to get the optimized hyperparameters for maximum depth and min_sample_split for all 4 featurization using cross-validation with GridSearchCV.
- Define a function to plot 3D graph using plotly for hyperparamter and AUC (Train and Test)
- Created HeatMap for AUC using hyperparameter values
- Define a function to test Decision tree on optimized/best hyperparameter for all 4 featurization and visually represent the errors vs hyperparameter plot.
- Used ROC-AUC and Confusion matrix as performance metric
- Calculated top 20 features from both positive and negative review classes for BOW and TFIDF
- Visualization of Decision Tree using graphviz upto 3 levels