

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 - unique 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 considered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered neutral 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 will 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

```

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 points
# 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 500000""", co
n)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 """, con)

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rating(0).
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 (525814, 10)

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertised
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia	1	1	1	1219017600	"Delight" says it all

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
--	----	-----------	--------	-------------	----------------------	------------------------	-------	------	---------

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
```

```
FROM REVIEWS
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANII WAFE
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANII WAFE
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANII WAFE
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANII WAFE
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANII WAFE

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 delete 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=False, kind='quicksort', na_position='last')
```

In [9]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[9]:

```
(364173, 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]:

69.25890143662969

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 calculations

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	HelpfulnessDenominator	Score	Time	Summary
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son at College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside

In [12]:

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [13]:

```
#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()
```

(364171, 10)

Out[13]:

```
1    307061
0     57110
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 observed 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)

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)
```

this witty little book makes my son laugh at loud. i recite it in the car as we're driving along and he always can sing the refrain. he's learned about whales, India, drooping roses: i love all the new words this book introduces and the silliness of it all. this is a classic book i am willing to bet my son will STILL be able to recite from memory when he is in college

=====

I was really looking forward to these pods based on the reviews. Starbucks is good, but I prefer bolder taste.... imagine my surprise when I ordered 2 boxes - both were expired! One expired back in 2005 for gosh sakes. I admit that Amazon agreed to credit me for cost plus part of shipping, but geez, 2 years expired!!! I'm hoping to find local San Diego area shoppe that carries pods so that I can try something different than starbucks.

=====

Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing I do not think belongs in it is Canola oil. Canola or rapeseed is not something a dog would ever find in nature and if it did find rapeseed in nature and eat it, it would poison them. Today's Food industries have convinced the masses that Canola oil is a safe and even better oil than olive or virgin coconut, facts though say otherwise. Until the late 70's it was poisonous until they figured out a way to fix that. I still like it but it could be better.

=====

Can't do sugar. Have tried scores of SF Syrups. NONE of them can touch the excellence of this product.

Thick, delicious. Perfect. 3 ingredients: Water, Maltitol, Natural Maple Flavor. PERIOD. No chemicals. No garbage.

Have numerous friends & family members hooked on this stuff. My husband & son, who do NOT like "sugar free" prefer this over major label regular syrup.

I use this as my SWEETENER in baking: cheesecakes, white brownies, muffins, pumpkin pies, etc... Unbelievably delicious...

Can you tell I like it? :)

=====

In [15]:

```
# 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_1500 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

this witty little book makes my son laugh at loud. i recite it in the car as we're driving along and he always can sing the refrain. he's learned about whales, India, drooping roses: i love all the new words this book introduces and the silliness of it all. this is a classic book i am willing to bet my son will STILL be able to recite from memory when he is in college

In [16]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-element
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)

```

this witty little book makes my son laugh at loud. i recite it in the car as we're driving along and he always can sing the refrain. he's learned about whales, India, drooping roses: i love all the new words this book introduces and the silliness of it all. this is a classic book i am willing to bet my son will STILL be able to recite from memory when he is in college

I was really looking forward to these pods based on the reviews. Starbucks is good, but I prefer bolder taste.... imagine my surprise when I ordered 2 boxes - both were expired! One expired back in 2005 for gosh sakes. I admit that Amazon agreed to credit me for cost plus part of shipping, but geez, 2 years expired!!! I'm hoping to find local San Diego area shoppe that carries pods so that I can try something different than starbucks.

Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing I do not think belongs in it is Canola oil. Canola or rapeseed is not something a dog would ever find in nature and if it did find rapeseed in nature and eat it, it would poison them. Today's Food industries have convinced the masses that Canola oil is a safe and even better oil than olive or virgin coconut, facts though say otherwise. Until the late 70's it was poisonous until they figured out a way to fix that. I still like it but it could be better.

Can't do sugar. Have tried scores of SF Syrups. NONE of them can touch the excellence of this product. Thick, delicious. Perfect. 3 ingredients: Water, Maltitol, Natural Maple Flavor. PERIOD. No chemicals. No garbage. Have numerous friends & family members hooked on this stuff. My husband & son, who do NOT like "sugar free" prefer this over major label regular syrup. I use this as my SWEETENER in baking: cheesecakes, white brownies, muffins, pumpkin pies, etc... Unbelievably delicious...Can you tell I like it? :)

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_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)

```

Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing I do not think belongs in it is Canola oil. Canola or rapeseed is not something a dog would ever fi

I do not think belongs in it is Canola oil. Canola or rapeseed is not something a dog would ever find in nature and if it did find rapeseed in nature and eat it, it would poison them. Today is Food industries have convinced the masses that Canola oil is a safe and even better oil than olive or virgin coconut, facts though say otherwise. Until the late 70 is it was poisonous until they figured out a way to fix that. I still like it but it could be better.

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)
```

this witty little book makes my son laugh at loud. i recite it in the car as we're driving along and he always can sing the refrain. he's learned about whales, India, drooping roses: i love all the new words this book introduces and the silliness of it all. this is a classic book i am willing to bet my son will STILL be able to recite from memory when he is in college

In [20]:

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

Great ingredients although chicken should have been 1st rather than chicken broth the only thing I do not think belongs in it is Canola oil Canola or rapeseed is not something a dog would ever find in nature and if it did find rapeseed in nature and eat it it would poison them Today is Food industries have convinced the masses that Canola oil is a safe and even better oil than olive or virgin coconut facts though say otherwise Until the late 70 is it was poisonous until they figured out a way to fix that I still like it but it could be better

In [21]:

```
# 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 been removed in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", \
    "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
    'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', \
    'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
    'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
    'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', \
    'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', \
    'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', \
    'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
    've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', \
    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', \
    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
    'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:

```
# Combining all the above students
from tqdm import tqdm
```



```

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

```

100%|██████████| 364171/364171 [02:37<00:00, 2314.81it/s]

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

'great ingredients although chicken rather chicken broth thing not think belongs canola oil canola rapeseed not someting dog would ever find nature find rapeseed nature eat would poison today food industries convinced masses canola oil safe even better oil olive virgin coconut facts though say otherwise late poisonous figured way fix still like could better'

[3.2] Preprocessing Review Summary

In [24]:

```

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

```

100%|██████████| 364171/364171 [01:42<00:00, 3542.91it/s]

In [24]:

```
final['Cleaned_text'] = preprocessed_reviews
```

In [25]:

```

### Sort data according to time series
final.sort_values('Time', inplace=True)

```

In [26]:

```

### Taking 100k samples
final_100k = final.sample(n=100000)

```

In [27]:

```

x = final_100k['Cleaned_text']
x.size

```

Out[27]:

100000

In [28]:

```
y = final_100k['Score']
y.size
```

Out[28]:

100000

In [29]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=42)
```

[4] Featurization

[4.1] BAG OF WORDS

In [30]:

```
#BoW
count_vect = CountVectorizer() #in scikit-learn
x_train_bow = count_vect.fit_transform(x_train)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

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

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaaaaaaaaa', 'aaaaaaaaaaaaaa', 'aaaaaaaah',
'aaaaaaaahhhhhh', 'aaaaaaand', 'aaaaaaaawwwwwwwww', 'aaaah']
=====
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (30000, 51338)
the number of unique words  51338
```

In [31]:

```
from sklearn import preprocessing
x_train_bow = preprocessing.normalize(x_train_bow)
x_test_bow = preprocessing.normalize(x_test_bow)
```

[4.2] Bi-Grams and n-Grams.

In [31]:

```
#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/modules/generated/sklearn.feature\_extraction.text.CountVectorizer.html

# 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(x_train)
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_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (70000, 5000)
```

the number of unique words including both unigrams and bigrams 5000

[4.3] TF-IDF

In [32]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
x_train_tfidf = tf_idf_vect.fit_transform(x_train)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print('='*50)

x_test_tfidf = tf_idf_vect.transform(x_test)
print("the type of count vectorizer ",type(x_test_tfidf))
print("the shape of out text TFIDF vectorizer ",x_test_tfidf.get_shape())
print("the number of unique words including both unigrams and bigrams ", x_test_tfidf.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['aa', 'ab', 'abandoned', 'abdominal', 'ability',
'able', 'able add', 'able buy', 'able drink', 'able eat']
=====
the type of count vectorizer <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer (30000, 40840)
the number of unique words including both unigrams and bigrams 40840
```

In [33]:

```
from sklearn import preprocessing
x_train_tfidf = preprocessing.normalize(x_train_tfidf)
x_test_tfidf = preprocessing.normalize(x_test_tfidf)
```

[4.4] Word2Vec

In [36]:

```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentence=[]
for sentence in x:
    list_of_sentence.append(sentence.split())
```

In [37]:

```
# 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 variable 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_sentence,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.bin', binary=True)
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w2v ")

```

```

[('fantastic', 0.842923104763031), ('awesome', 0.8419500589370728), ('good', 0.8406391739845276), ('excellent', 0.8268475532531738), ('wonderful', 0.7978573441505432), ('terrific', 0.7962212562561035), ('perfect', 0.7497711777687073), ('fabulous', 0.7309261560440063), ('nice', 0.7059241533279419), ('decent', 0.6821746230125427)]
=====
[('greatest', 0.7703800797462463), ('nastiest', 0.7154430150985718), ('best', 0.68451988697052), ('disgusting', 0.6830753684043884), ('smoothest', 0.6693339347839355), ('terrible', 0.6668342351913452), ('awful', 0.6540132164955139), ('horrible', 0.6431991457939148), ('tastiest', 0.6185607314109802), ('horrid', 0.6162493824958801)]

```

In [38]:

```

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

```

```

number of words that occurred minimum 5 times 18851
sample words ['dissappointing', 'much', 'wouldnt', 'waste', 'another', 'second', 'talking', 'except', 'say', 'im', 'tossing', 'product', 'lacked', 'taste', 'whatsoever', 'want', 'nothing', 'stick', 'water', 'least', 'flavor', 'good', 'lots', 'great', 'coffee', 'especially', 'wake', 'enjoy', 'give', 'plus', 'apricot', 'kernels', 'lower', 'blood', 'pressure', 'healthy', 'snack', 'swanson', 'fine', 'treat', 'cancer', 'need', 'power', 'bitter', 'compared', 'gum', 'lot', 'appreciate', 'flavors', 'runs']

```

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

[4.4.1.1] Avg W2v

In [39]:

```

# 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_sentence): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you use google's w2v
    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%|██████████| 100000/100000 [04:19<00:00, 385.86it/s]

100000
50

In [40]:

```

x_train_avgw2v,x_test_avgw2v,y_train,y_test = train_test_split(sent_vectors,y,test_size=0.3,random_state=42)

```

In [41]:

```
from sklearn import preprocessing
x_train_avgw2v = preprocessing.normalize(x_train_avgw2v)
x_test_avgw2v = preprocessing.normalize(x_test_avgw2v)
```

[4.4.1.2] TFIDF weighted W2v

In [42]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer(min_df=10, max_features=500)
tf_idf_matrix = model.fit_transform(x)
# 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 [43]:

```
# 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 list
row=0;
for sent in tqdm(list_of_sentence): # 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 corpus
            # sent.count(word) = tf value 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
```

100%|██████████| 100000/100000 [04:53<00:00, 340.99it/s]

In [44]:

```
x_train_tfidfw2v, x_test_tfidfw2v, y_train, y_test = train_test_split(tfidf_sent_vectors, y, test_size=0.3, random_state=42)
```

In [45]:

```
from sklearn import preprocessing
x_train_tfidfw2v = preprocessing.normalize(x_train_tfidfw2v)
x_test_tfidfw2v = preprocessing.normalize(x_test_tfidfw2v)
```

[5] Assignment 8: Decision Trees

1. Apply Decision Trees on these feature sets

- **SET 1:** Review text, preprocessed one converted into vectors using (BOW)
- **SET 2:** Review text, preprocessed one converted into vectors using (TFIDF)
- **SET 3:** Review text, preprocessed one converted into vectors using (AVG W2v)
- **SET 4:** Review text, preprocessed one converted into vectors using (TFIDF W2v)

2. The hyper parameter tuning (best `depth` in range [1, 5, 10, 50, 100, 500, 100], and the best `min_samples_split` in range

[5, 10, 100, 500])

- Find the best hyper parameter which will give the maximum [AUC](#) value
- Find the best hyper paramter using k-fold cross validation or simple cross validation data
- Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. Graphviz

- Visualize your decision tree with Graphviz. It helps you to understand how a decision is being made, given a new vector.
- Since feature names are not obtained from word2vec related models, visualize only BOW & TFIDF decision trees using Graphviz
- Make sure to print the words in each node of the decision tree instead of printing its index.
- Just for visualization purpose, limit max_depth to 2 or 3 and either embed the generated images of graphviz in your notebook, or directly upload them as .png files.

4. Feature importance

- Find the top 20 important features from both feature sets [Set 1](#) and [Set 2](#) using `feature_importances_` method of [Decision Tree Classifier](#) and print their corresponding feature names

5. Feature engineering

- To increase the performance of your model, you can also experiment with with feature engineering like :
 - Taking length of reviews as another feature.
 - Considering some features from review summary as well.

6. Representation of results

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
- Along with plotting ROC curve, you need to print the [confusion matrix](#) with predicted and original labels of test data points. Please visualize your confusion matrices using [seaborn heatmaps](#).

7. Conclusion

- [You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link](#)

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-leakage, 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](#).

Applying Decision Trees

[5.1] Applying Decision Trees on BOW, SET 1

In [46]:

```
## find hyperparameter using cross validation score and plot AUC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
max_depth_values = [1, 5, 10, 50, 100, 500, 1000]

#empty lists that stores cv scores and training_scores
cv_scores = []
training_scores = []

#perform k fold cross validation
for d in max_depth_values:
```

```

clf = DecisionTreeClassifier(max_depth=d)
clf.fit(x_train_bow, y_train)
y_train_pred = clf.predict_proba(x_train_bow)[:,-1]
y_test_pred = clf.predict_proba(x_test_bow)[:,-1]

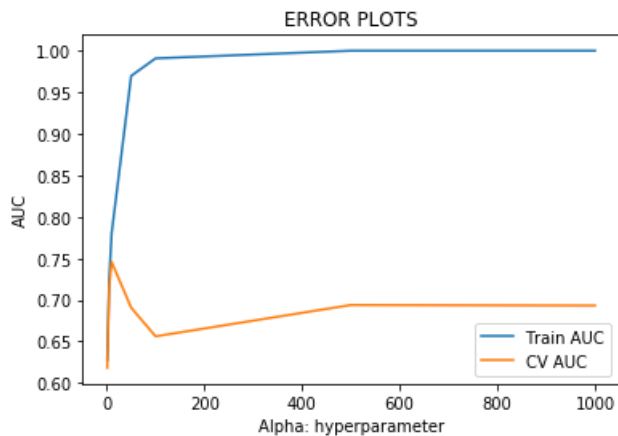
training_scores.append(roc_auc_score(y_train,y_train_pred))
cv_scores.append(roc_auc_score(y_test, y_test_pred))

```

```

#plot cross-validated score, training score vs alpha
plt.plot(max_depth_values, training_scores, label='Train AUC')
plt.plot(max_depth_values, cv_scores, label='CV AUC')
plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

```



Observation: By observing the curve, getting high AUC at max_depth 50.

In [47]:

```

from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=10)

```

Observation: By Observing curve, getting high AUC at max_depth_value 10.

In [49]:

```

### Using GridsearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from scipy.stats import randint as sp_randint
clf = DecisionTreeClassifier()

param_grid = {'max_depth':[1, 5, 10, 50, 100, 500, 1000], 'min_samples_split':[5, 10, 100, 500]}
#params we need to try on classifier
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(clf,param_grid,cv=tscv,verbose=1,scoring='roc_auc')
gsv.fit(x_train_bow,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))

```

Fitting 10 folds for each of 28 candidates, totalling 280 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
 [Parallel(n_jobs=1)]: Done 280 out of 280 | elapsed: 61.1min finished

Best HyperParameter: {'max_depth': 50, 'min_samples_split': 500}
 Best Accuracy: 79.79%

In []:

```

### ROC Curve using false positive rate versus true positive rate

```

In [50]:

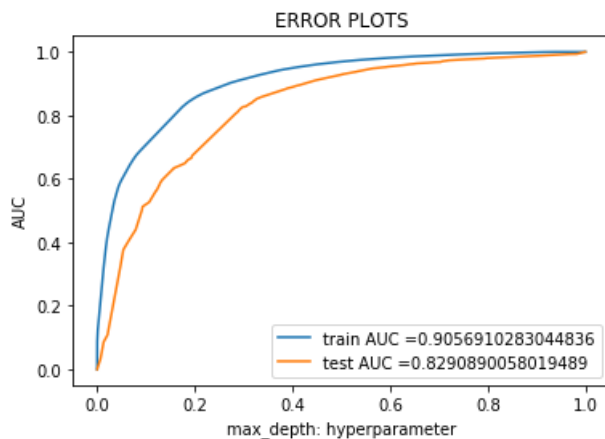
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(max_depth=50,min_samples_split=500)
clf.fit(x_train_bow, y_train)
y_pred = clf.predict(x_test_bow)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(x_train_bow)[:,-1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf.predict_proba(x_test_bow)[:,-1])

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("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, clf.predict(x_train_bow)))
print("Test confusion matrix")
print(confusion_matrix(y_test, clf.predict(x_test_bow)))
```



=====

Train confusion matrix

```
[[ 6025  4931]
 [ 2311 56733]]
```

Test confusion matrix

```
[[ 2093  2676]
 [ 1345 23886]]
```

In [52]:

```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[52]:

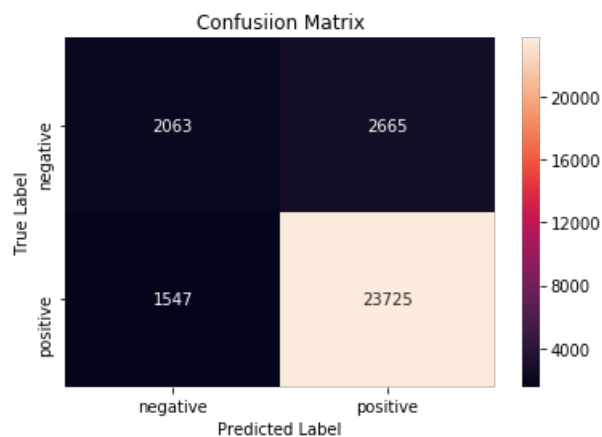
```
array([[ 2063,  2665],
       [ 1547, 23725]])
```

In [53]:

```
# plot confusion matrix to describe the performance of classifier.
```



```
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



[5.1.1] Top 20 important features from SET 1

In [54]:

```
# Calculate feature importances from decision trees
importances = clf.feature_importances_

# Sort feature importances in descending order
indices = np.argsort(importances)[:,-1][:20]

# Rearrange feature names so they match the sorted feature importances
names = count_vect.get_feature_names()

sns.set(rc={'figure.figsize': (6.7, 4.27)})

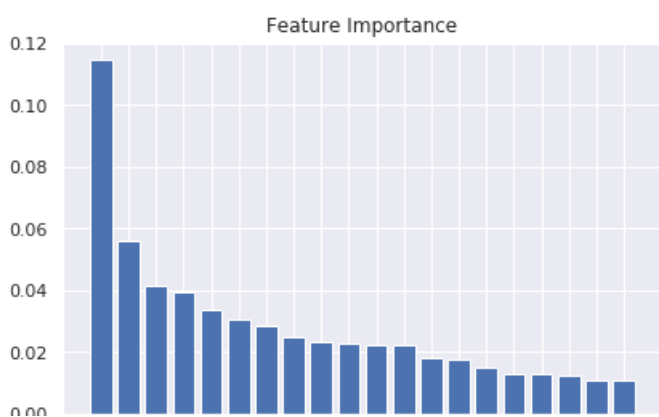
# Create plot
plt.figure()

# Create plot title
plt.title("Feature Importance")

# Add bars
plt.bar(range(20), importances[indices])

# Add feature names as x-axis labels
names = np.array(names)
plt.xticks(range(20), names, rotation=90)

# Show plot
plt.show()
# uni_gram.get_feature_names()
```



0.00

ability
able
able buy
able find
able get
able order
absolute
absolute favorite
absolutely
absolutely delicious
absolutely love
absolutely loves
absolutely no
absorb
acai
acceptable
access
according
account
accustomed

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

In [51]:

```
from sklearn.tree import export_graphviz
from sklearn import tree
from IPython.display import Image

clf = clf.fit(x_train_bow, y_train)
#tree.export_graphviz(clf, out_file='bow_tree.png', max_depth=2)
```

In [68]:

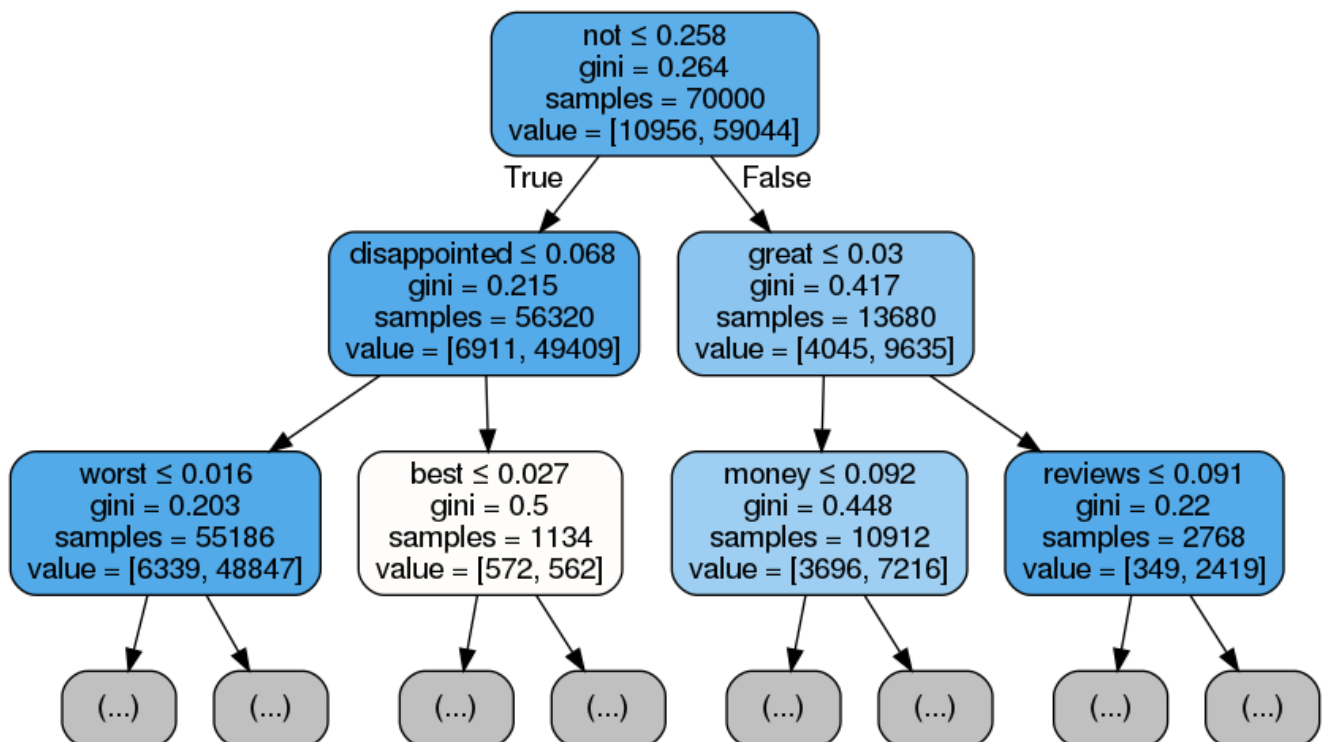
```
feature_names = count_vect.get_feature_names()
```

In [69]:

```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()

export_graphviz(clf, out_file=dot_data, feature_names=feature_names,
                filled=True, rounded=True,
                special_characters=True, max_depth=2)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out [69]:



[5.2] Applying Decision Trees on TFIDF, SET 2

In [56]:

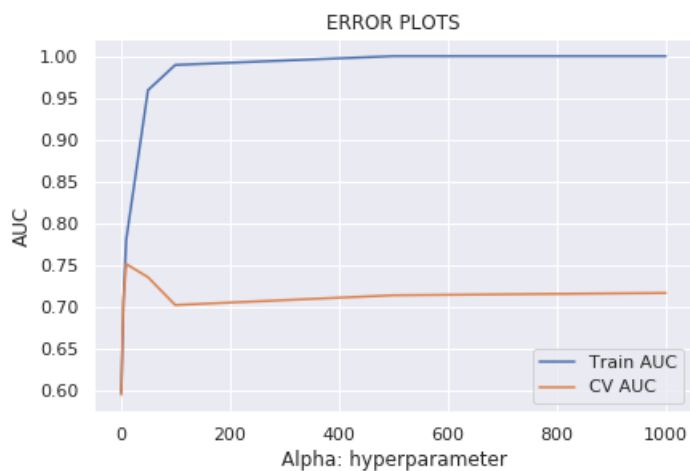
```
## find hyperparameter using cross validation score and plot AUC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
max_depth_values = [1, 5, 10, 50, 100, 500, 1000]

#empty lists that stores cv scores and training_scores
cv_scores = []
training_scores = []

#perform k fold cross validation
for d in max_depth_values:
    clf = DecisionTreeClassifier(max_depth=d)
    clf.fit(x_train_tfidf, y_train)
    y_train_pred = clf.predict_proba(x_train_tfidf)[:,1]
    y_test_pred = clf.predict_proba(x_test_tfidf)[:,1]

    training_scores.append(roc_auc_score(y_train,y_train_pred))
    cv_scores.append(roc_auc_score(y_test, y_test_pred))

#plot cross-validated score, training score vs alpha
plt.plot(max_depth_values, training_scores, label='Train AUC')
plt.plot(max_depth_values, cv_scores, label='CV AUC')
plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



Observation: By Observing curve, getting high AUC at max_depth_value 50.

In [57]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from scipy.stats import randint as sp_randint
clf = DecisionTreeClassifier()

param_grid = {'max_depth':[1, 5, 10, 50, 100, 500, 1000], 'min_samples_split':[5, 10, 100, 500]}
#params we need to try on classifier
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(clf,param_grid,cv=tscv,verbose=1,scoring='roc_auc')
gsv.fit(x_train_tfidf,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

Fitting 10 folds for each of 28 candidates, totalling 280 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 280 out of 280 | elapsed: 80.5min finished

Best HyperParameter: {'max_depth': 50, 'min_samples_split': 500}
Best Accuracy: 79.97%

In [35]:

```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(max_depth=50,min_samples_split=500)
clf.fit(x_train_tfidf,y_train)
y_pred = clf.predict(x_test_tfidf)
```

In []:

```
## ROC Curve
```

In [61]:

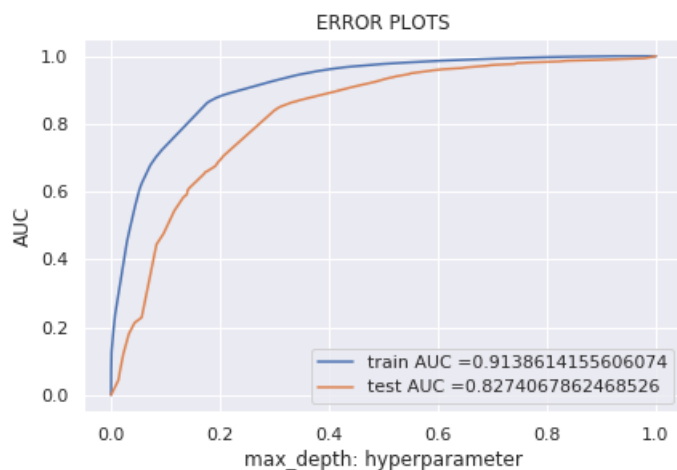
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
clf = DecisionTreeClassifier(max_depth=50,min_samples_split=500)
clf.fit(x_train_tfidf, y_train)
y_pred = clf.predict(x_test_tfidf)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(x_train_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf.predict_proba(x_test_tfidf)[:,1])

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("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, clf.predict(x_train_tfidf)))
print("Test confusion matrix")
print(confusion_matrix(y_test, clf.predict(x_test_tfidf)))
```



=====

Train confusion matrix

```
[[ 6328  4621]
 [ 2005 57046]]
```

Test confusion matrix

```
[[ 2139  2589]
 [ 1334 23938]]
```

In [62]:

```
In [62]:
```

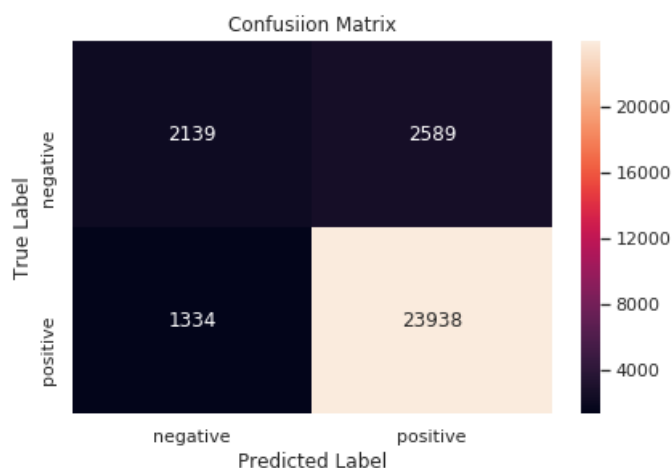
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

```
Out[62]:
```

```
array([[ 2139,  2589],
       [ 1334, 23938]])
```

```
In [63]:
```

```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



[5.2.1] Top 20 important features from SET 2

```
In [64]:
```

```
# Calculate feature importances from decision trees
importances = clf.feature_importances_

# Sort feature importances in descending order
indices = np.argsort(importances)[::-1][:20]

# Rearrange feature names so they match the sorted feature importances
names = tf_idf_vect.get_feature_names()

sns.set(rc={'figure.figsize': (6.7, 4.27)})

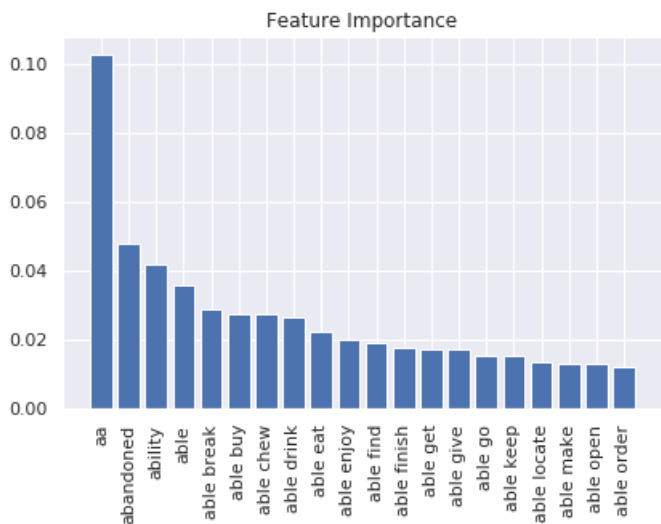
# Create plot
plt.figure()

# Create plot title
plt.title("Feature Importance")

# Add bars
plt.bar(range(20), importances[indices])

# Add feature names as x-axis labels
names = np.array(names)
plt.xticks(range(20), names, rotation=90)

# Show plot
plt.show()
# uni_gram.get_feature_names()
```



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

In [70]:

```
from sklearn.tree import export_graphviz
from sklearn import tree
from IPython.display import Image

clf = clf.fit(x_train_tfidf, y_train)
#tree.export_graphviz(clf,out_file='bow_tree.png',max_depth=2)
```

In [71]:

```
feature_names = tf_idf_vect.get_feature_names()
```

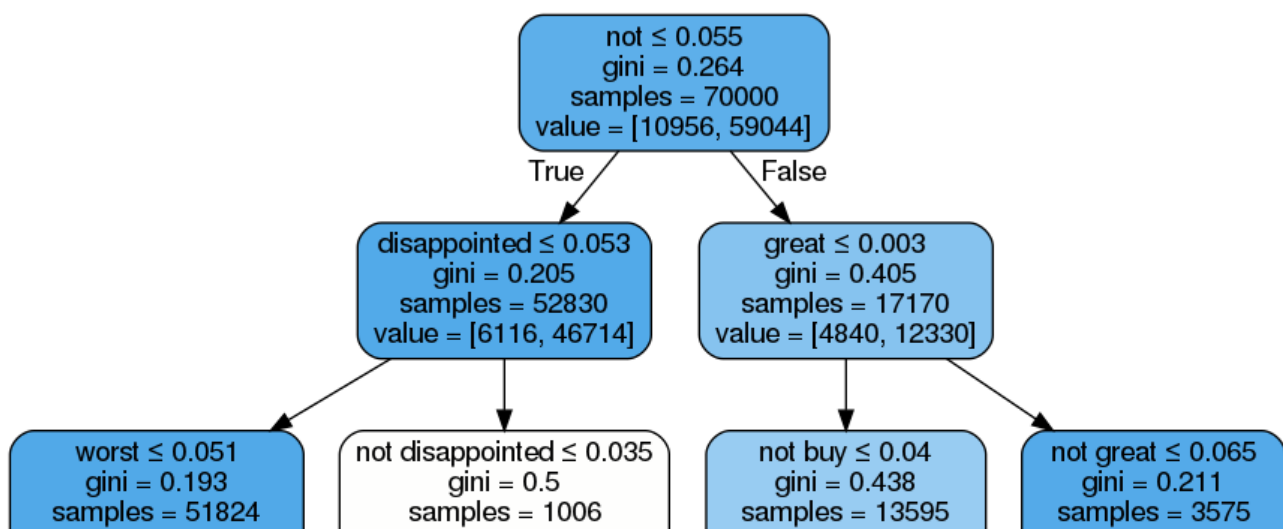
In [72]:

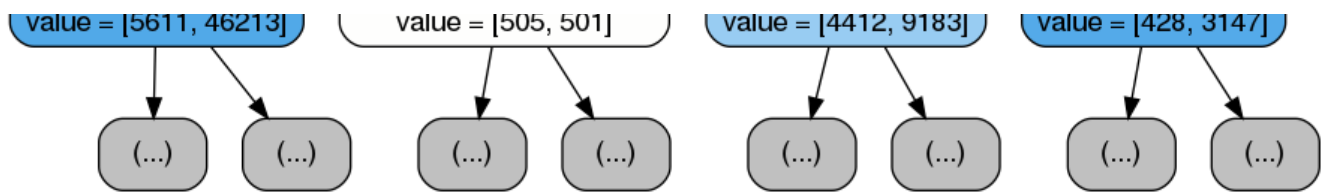
```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus

dot_data = StringIO()

export_graphviz(clf, out_file=dot_data,feature_names=feature_names,
                filled=True, rounded=True,
                special_characters=True,max_depth=2)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[72]:





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

In [66]:

```

## find hyperparameter using cross validation score and plot AUC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
max_depth_values = [1, 5, 10, 50, 100, 500, 1000]

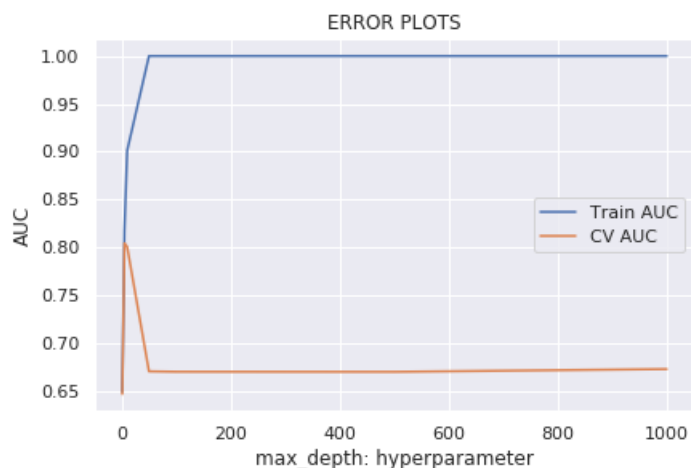
#empty lists that stores cv scores and training_scores
cv_scores = []
training_scores = []

#perform k fold cross validation
for d in max_depth_values:
    clf = DecisionTreeClassifier(max_depth=d)
    clf.fit(x_train_avgw2v, y_train)
    y_train_pred = clf.predict_proba(x_train_avgw2v)[:,1]
    y_test_pred = clf.predict_proba(x_test_avgw2v)[:,1]

    training_scores.append(roc_auc_score(y_train,y_train_pred))
    cv_scores.append(roc_auc_score(y_test, y_test_pred))

#plot cross-validated score, training score vs alpha
plt.plot(max_depth_values, training_scores, label='Train AUC')
plt.plot(max_depth_values, cv_scores, label='CV AUC')
plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

```



Observation: By Observing curve, getting value high AUC at max_depth_value 10.

In [67]:

```

from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()

param_grid = {'max_depth':[1, 5, 10, 50, 100, 500, 1000], 'min_samples_split':[5, 10, 100, 500]}
#params we need to try on classifier
gsv = GridSearchCV(clf,param_grid,cv=tscv,verbose=1,scoring='roc_auc')
gsv.fit(x_train_avgw2v,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))

```

Fitting 10 folds for each of 28 candidates, totalling 280 fits

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.  
[Parallel(n_jobs=1)]: Done 280 out of 280 | elapsed: 14.3min finished
```

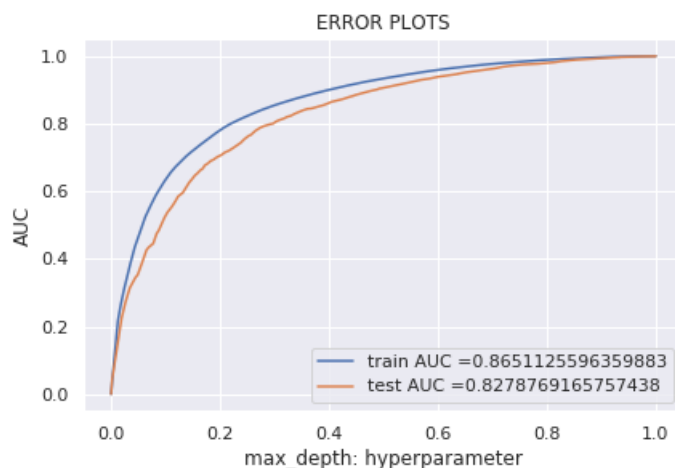
Best HyperParameter: {'max_depth': 10, 'min_samples_split': 500}
Best Accuracy: 81.29%

In []:

```
## ROC Curve
```

In [69]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve  
from sklearn.metrics import roc_curve, auc  
clf = DecisionTreeClassifier(max_depth=10,min_samples_split=500)  
clf.fit(x_train_avgw2v, y_train)  
y_pred = clf.predict(x_test_avgw2v)  
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive  
# class  
# not the predicted outputs  
  
train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(x_train_avgw2v)[:,-1])  
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf.predict_proba(x_test_avgw2v)[:,-1])  
  
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("max_depth: hyperparameter")  
plt.ylabel("AUC")  
plt.title("ERROR PLOTS")  
plt.show()  
  
print("="*100)  
  
from sklearn.metrics import confusion_matrix  
print("Train confusion matrix")  
print(confusion_matrix(y_train, clf.predict(x_train_avgw2v)))  
print("Test confusion matrix")  
print(confusion_matrix(y_test, clf.predict(x_test_avgw2v)))
```



=====

Train confusion matrix

```
[[ 3957  6992]  
 [ 1972 57079]]
```

Test confusion matrix

```
[[ 1466  3262]  
 [ 1018 24254]]
```


In [70]:

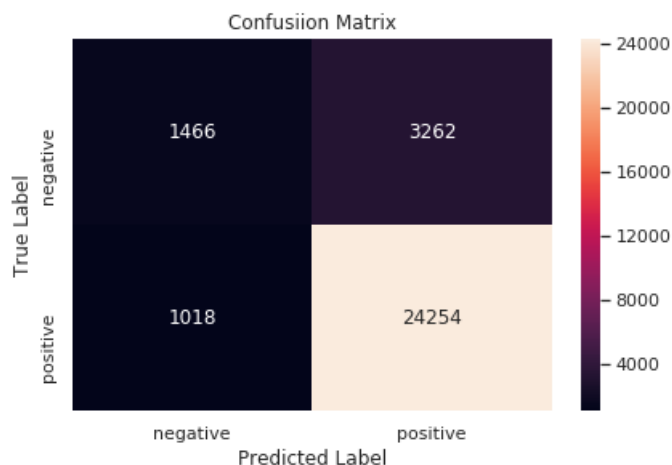
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[70]:

```
array([[ 1466,  3262],
       [ 1018, 24254]])
```

In [71]:

```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



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

In [72]:

```
## find hyperparameter using cross validation score and plot AUC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
max_depth_values = [1, 5, 10, 50, 100, 500, 1000]

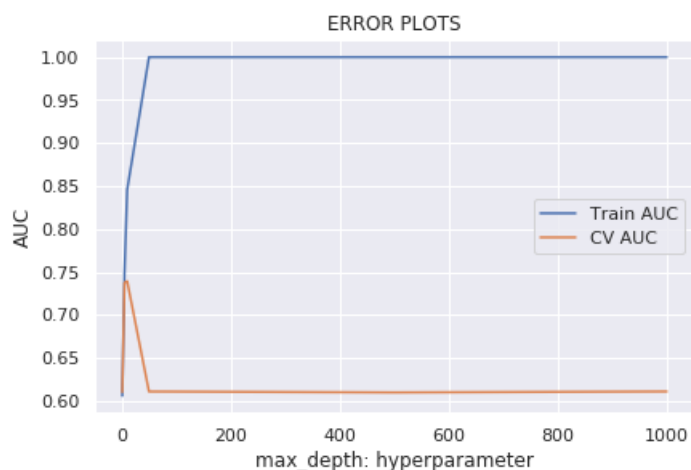
#empty lists that stores cv scores and training_scores
cv_scores = []
training_scores = []

#perform k fold cross validation
for d in max_depth_values:
    clf = DecisionTreeClassifier(max_depth=d)
    clf.fit(x_train_tfidf2v, y_train)
    y_train_pred = clf.predict_proba(x_train_tfidf2v)[:,1]
    y_test_pred = clf.predict_proba(x_test_tfidf2v)[:,1]

    training_scores.append(roc_auc_score(y_train,y_train_pred))
    cv_scores.append(roc_auc_score(y_test, y_test_pred))

#plot cross-validated score, training score vs alpha
plt.plot(max_depth_values, training_scores, label='Train AUC')
plt.plot(max_depth_values, cv_scores, label='CV AUC')
plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
```

```
plt.plot(100, 1.00)
plt.title("ERROR PLOTS")
plt.show()
```



Observation: By Observing curve, getting high AUC at max_depth_value 10.

In [73]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()

param_grid = {'max_depth':[1, 5, 10, 50, 100, 500, 1000], 'min_samples_split':[5, 10, 100, 500]}
#params we need to try on classifier
gsv = GridSearchCV(clf,param_grid,cv=tscv,verbose=1,scoring='roc_auc')
gsv.fit(x_train_tfidf2v,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

Fitting 10 folds for each of 28 candidates, totalling 280 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 280 out of 280 | elapsed: 14.7min finished

Best HyperParameter: {'max_depth': 10, 'min_samples_split': 500}
Best Accuracy: 73.69%

In []:

```
## ROC Curve
```

In [75]:

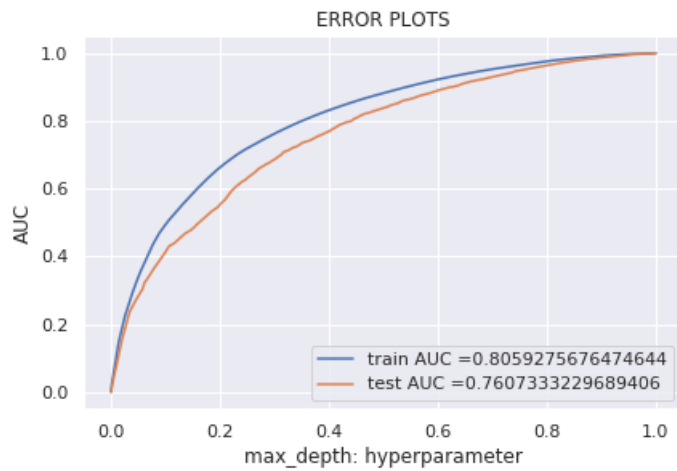
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
clf = DecisionTreeClassifier(max_depth=10,min_samples_split=500)
clf.fit(x_train_tfidf2v, y_train)
y_pred = clf.predict(x_test_tfidf2v)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(x_train_tfidf2v)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, clf.predict_proba(x_test_tfidf2v)[:,1])

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("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, clf.predict(x_train_tfidf2v)))
print("Test confusion matrix")
print(confusion_matrix(y_test, clf.predict(x_test_tfidf2v)))
```



```
=====

Train confusion matrix
[[ 1930  9019]
 [ 1144 57907]]
Test confusion matrix
[[  674  4054]
 [  546 24726]]
```

In [77]:

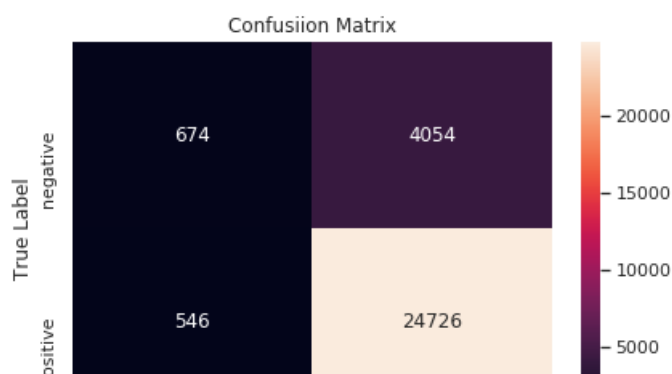
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[77]:

```
array([[ 674,  4054],
       [  546, 24726]])
```

In [78]:

```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```





[6] Conclusions

In [3]:

```
#### from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Hyperparameter", "AUC"]

x.add_row(["BOW", "50", "82.4%"])
x.add_row(["TFIDF", "50", "82.74%"])
x.add_row(["AvgW2V", "10", "82.78%"])
x.add_row(["TFIDF W2V", "10", "76%"])

print(x)
```

```
+-----+-----+-----+
| Vectorizer | Hyperparameter | AUC |
+-----+-----+-----+
| BOW        | 50              | 82.4% |
| TFIDF      | 50              | 82.74% |
| AvgW2V     | 10              | 82.78% |
| TFIDF W2V  | 10              | 76%   |
+-----+-----+-----+
```

In []:

- 1) With AvgW2V getting high accuracy.
- 2) Decision Tree taking more computation time than naive bais and logistic regression algorithm.

In []: