# Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

**Objective:**

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

[Q] How to determine if a review is positive or negative?

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

# 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:\ProgramData\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWa
rning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_seria
l")
```

In [2]:

```
# using SQLite Table to read data.
```

```
con = sqlite3.connect('C:/Users/Excel/Desktop/vins/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 50
0000 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 Sco
re != 3 LIMIT 500000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score
 != 3 LIMIT 50000""", con)

# Give reviews with Score>3 a positive rating(1), and reviews with a sc
ore<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 (50000, 10)
```

Out[2]:

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | Helpfulnes |
|---|---|---|---|---|---|---|

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | Helpfulnes |
|---|---|---|---|---|---|---|
| **0** | 1 | B001E4KFG0 | A3SGXH7AUHU8GW | delmartian | 1 | 1 |
| **1** | 2 | B00813GRG4 | A1D87F6ZCVE5NK | dll pa | 0 | 0 |
| **2** | 3 | B000LQOCH0 | ABXLMWJIXXAIN | Natalia Corres "Natalia Corres" | 1 | 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 | COU |
|---|---|---|---|---|---|---|---|
| 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 |
|---|---|---|---|---|---|---|

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

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 | Helpfuln |
|---|---|---|---|---|---|---|

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | Helpfuln |
|---|---|---|---|---|---|---|
| **0** | 78445 | B000HDL1RQ | AR5J8UI46CURR | Geetha Krishnan | 2 | 2 |
| **1** | 138317 | B000HDOPYC | AR5J8UI46CURR | Geetha Krishnan | 2 | 2 |
| **2** | 138277 | B000HDOPYM | AR5J8UI46CURR | Geetha Krishnan | 2 | 2 |
| **3** | 73791 | B000HDOPZG | AR5J8UI46CURR | Geetha Krishnan | 2 | 2 |
| **4** | 155049 | B000PAQ75C | AR5J8UI46CURR | Geetha Krishnan | 2 | 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=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]: (46072, 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]: 92.144

**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 | Helpfuln |
|---|---|---|---|---|---|---|
| 0 | 64422 | B000MIDROQ | A161DK06JJMCYF | J. E. Stephens "Jeanne" | 3 | 1 |
| 1 | 44737 | B001EQ55RW | A2V0I904FH7ABY | Ram | 3 | 2 |

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

In [13]:
```
final["Time"] = pd.to_datetime(final["Time"], unit = "s")
final = final.sort_values(by = "Time")
```

In [14]:
```
#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()
```

```
(46071, 10)
```

Out[14]:
```
1    38479
0     7592
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 [15]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
```

```python
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 was a really good idea and the final product is outstanding. I use
the decals on my car window and everybody asks where i bought the decal
s i made.  Two thumbs up!
==================================================
Speaking as another Texan, I think the first rule for these delicious t
reats is to NOT order them during spring or summer. In fact, your safes
t bet is to ONLY order them in the dead of winter.  LOL!  As long as yo
u do that, be prepared for a truly amazing treat!  This package comes w
ith 12 bite-sized delicacies.  The chocolate is high-quality, the nuts
are crunchy, and the overall taste couldn't be better.  Definitely wort
h the price!
==================================================
Fast, easy and definitely delicious.  Makes a great cup of coffee and v
ery easy to make.  Good purchase.  Will continue to order from here.<br
/>Thanx...
==================================================
Naturally this review is based upon my cat's intake of Petite Cuisine.
She's not a particular picky eater, so I can't say much about that. How
ever, she looks to really enjoy this brand of cat food. I have tried so
me brands of wet food in the past that have made her sick (I know cats
seem to have digestive systems that are prone to upsetting!) Petite Cui
sine did not have any effect there, and she really enjoyed all the flav
ors. I don't feed her wet food often, usually just some tuna fish now a
nd then. So, although this food is expensive if you used it at every me
al, it is priced about the same as tuna, so it fits my needs perfectly.
I'm sure my cat will enjoy the rest of this case, and I can keep the ca
```

```
nned tuna to myself for now :-)
==================================================
```

In [16]:

```
# remove urls from text python: https://stackoverflow.com/a/40823105/40
84039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
This was a really good idea and the final product is outstanding. I use
the decals on my car window and everybody asks where i bought the decal
s i made.  Two thumbs up!
```

In [17]:

```
# 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 was a really good idea and the final product is outstanding. I use
the decals on my car window and everybody asks where i bought the decal
s i made.  Two thumbs up!
==================================================
Speaking as another Texan, I think the first rule for these delicious t
reats is to NOT order them during spring or summer. In fact, your safes
t bet is to ONLY order them in the dead of winter.  LOL!  As long as yo
u do that, be prepared for a truly amazing treat!  This package comes w
ith 12 bite-sized delicacies.  The chocolate is high-quality, the nuts
are crunchy, and the overall taste couldn't be better.  Definitely wort
h the price!
==================================================
Fast, easy and definitely delicious.  Makes a great cup of coffee and v
ery easy to make.  Good purchase.  Will continue to order from here.Tha
nx...
==================================================
Naturally this review is based upon my cat's intake of Petite Cuisine.
She's not a particular picky eater, so I can't say much about that. How
ever, she looks to really enjoy this brand of cat food. I have tried so
me brands of wet food in the past that have made her sick (I know cats
seem to have digestive systems that are prone to upsetting!) Petite Cui
sine did not have any effect there, and she really enjoyed all the flav
ors. I don't feed her wet food often, usually just some tuna fish now a
nd then. So, although this food is expensive if you used it at every me
al, it is priced about the same as tuna, so it fits my needs perfectly.
I'm sure my cat will enjoy the rest of this case, and I can keep the ca
nned tuna to myself for now :-)
```

In [18]:

```python
# 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 [19]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Fast, easy and definitely delicious.  Makes a great cup of coffee and v
ery easy to make.  Good purchase.  Will continue to order from here.<br
/>Thanx...
==================================================
```

In [20]:
```
#remove words with numbers python: https://stackoverflow.com/a/1808237
0/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
This was a really good idea and the final product is outstanding. I use
the decals on my car window and everybody asks where i bought the decal
s i made.  Two thumbs up!
```

In [21]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Fast easy and definitely delicious Makes a great cup of coffee and very
easy to make Good purchase Will continue to order from here br Thanx
```

In [22]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
t'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in
```

```
 the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'o
urs', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', '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', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', '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', "is
n'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 [23]:

```
# 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 stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████████████████████████| 46071/46071 [00:24<00:00, 184
5.69it/s]
```

In [27]:

```
preprocessed_reviews[1500]
```

Out[27]:

```
'fast easy definitely delicious makes great cup coffee easy make good p
urchase continue order thanx'
```

## [3.2] Preprocessing Review Summary

In [28]:

```
## Similartly you can do preprocessing for review summary also.
```

# [4] Featurization

## [4.1] BAG OF WORDS

In [29]:

```
X=preprocessed_reviews
Y=final["Score"]
```

In [30]:

```
from sklearn.model_selection import train_test_split
#time based splitting
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.33,shu
ffle=False)
X_train,X_cv,y_train,y_cv = train_test_split(X_train,y_train,test_size=
0.33,shuffle=False)
```

```
print(np.shape(X_train),y_train.shape)
print(np.shape(X_cv),y_cv.shape)
print(np.shape(X_test),y_test.shape)
```

```
(20680,) (20680,)
(10187,) (10187,)
(15204,) (15204,)
```

In [31]:
```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer=CountVectorizer(min_df=10,max_features=10000,ngram_range=(1,
2))
vectorizer.fit(X_train)

X_train_bow=vectorizer.transform(X_train)
X_cv_bow=vectorizer.transform(X_cv)
X_test_bow=vectorizer.transform(X_test)

print("After BOW vectorizer")
print(np.shape(X_train_bow),y_train.shape)
print(np.shape(X_cv_bow),y_cv.shape)
print(np.shape(X_test_bow),y_test.shape)
```

```
After BOW vectorizer
(20680, 10000) (20680,)
(10187, 10000) (10187,)
(15204, 10000) (15204,)
```

In [32]:
```
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
```

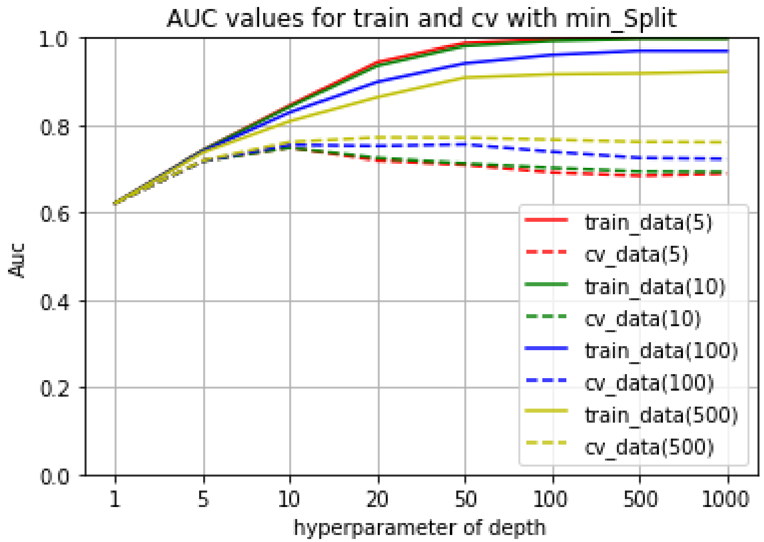
In [34]:
```
depth=[1, 5, 10,20, 50, 100, 500, 1000]
min_split=[5,10,100,500]
parameter={'max_depth':depth,'min_samples_split':min_split}

dt=GridSearchCV(DecisionTreeClassifier(class_weight='balanced'),paramet
er,verbose=1,scoring='roc_auc')
dt.fit(X_train_bow,y_train)

opt_depth,opt_min_split=dt.best_params_.get('max_depth'),dt.best_params
```

```
_.get('min_samples_split')
print("best optimized depth:",opt_depth)
print("best optimized min_split:",opt_min_split)

train_score=dt.cv_results_.get("mean_train_score")
test_score=dt.cv_results_.get("mean_test_score")


plt.plot(np.arange(len(depth)),train_score[::4],'r',label="train_data
(5)")
plt.plot(np.arange(len(depth)),test_score[::4],'r--',label="cv_data(5)"
)

plt.plot(np.arange(len(depth)),train_score[1::4],'g',label="train_data
(10)")
plt.plot(np.arange(len(depth)),test_score[1::4],'g--',label="cv_data(1
0)")

plt.plot(np.arange(len(depth)),train_score[2::4],'b',label="train_data
(100)")
plt.plot(np.arange(len(depth)),test_score[2::4],'b--',label="cv_data(10
0)")

plt.plot(np.arange(len(depth)),train_score[3::4],'y',label="train_data
(500)")
plt.plot(np.arange(len(depth)),test_score[3::4],'y--',label="cv_data(50
0)")

plt.xticks(np.arange(len(depth)),depth)
plt.legend()
plt.xlabel("hyperparameter of depth")
plt.ylabel("Auc")
plt.title("AUC values for train and cv with min_Split")
plt.ylim(0,1)
plt.grid(True)
plt.show()
```

```
Fitting 3 folds for each of 32 candidates, totalling 96 fits
```

```
[Parallel(n_jobs=1)]: Done  96 out of  96 | elapsed:  3.1min finished
```

```
best optimized depth: 20
best optimized min_split: 500
```

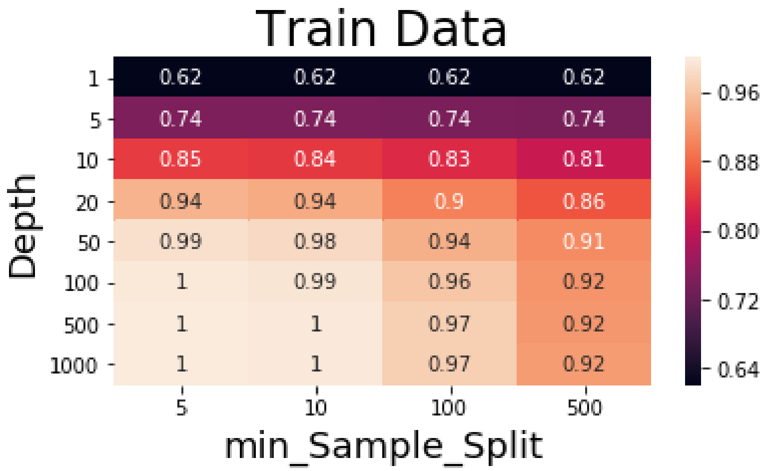


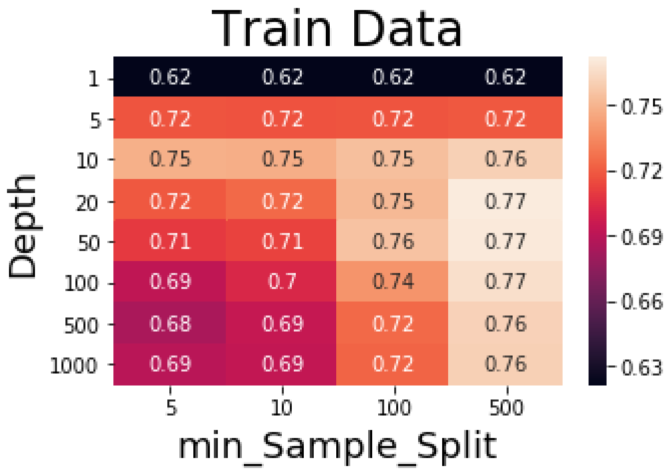

1. we get correct visualization of optimized max_depth.
2. we will go for the seaborn heatmap for min_samples_split.

In [72]:
```
df_heatmap = pd. DataFrame(train_score. reshape(8, 4), index=depth, col
umns=min_split )
fig = plt. figure(figsize=(6, 3))
heatmap = sns. heatmap(df_heatmap, annot=True)
plt. ylabel('Depth' , size=18)
plt. xlabel('min_Sample_Split' , size=18)
plt. title("Train Data", size=24)
plt. show

df_heatmap = pd. DataFrame(test_score. reshape(8, 4), index=depth, colu
mns=min_split )
fig = plt. figure(figsize=(5, 3))
heatmap = sns. heatmap(df_heatmap, annot=True)
plt. ylabel('Depth' , size=18)
plt. xlabel('min_Sample_Split' , size=18)
```

```
plt. title("Train Data", size=24)
plt. show
```

Out[72]: <function matplotlib.pyplot.show(*args, **kw)>

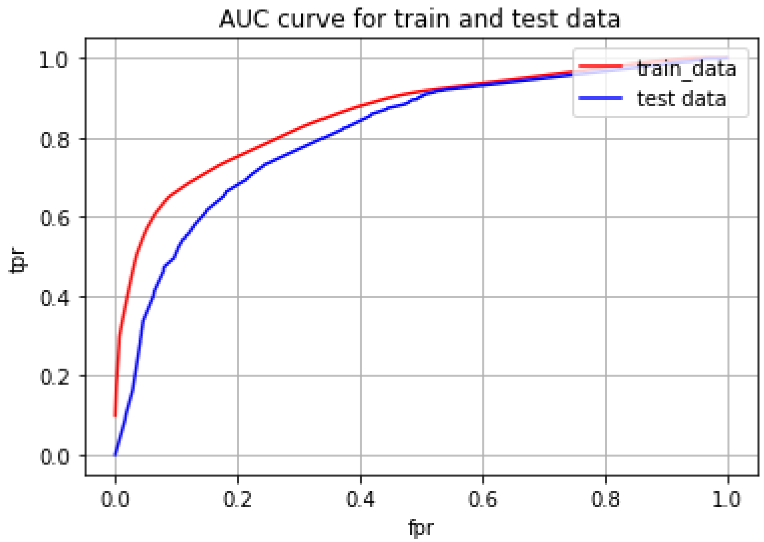



In [53]:
```
from sklearn.metrics import roc_auc_score
dt=DecisionTreeClassifier(class_weight='balanced',max_depth=20,min_samp
les_split=500)
dt.fit(X_train_bow,y_train)
```

```
y_test_proba=dt.predict_proba(X_test_bow)
y_train_proba=dt.predict_proba(X_train_bow)

fpr,tpr,threshold=roc_curve(y_train,y_train_proba[:,1])
fpr1,tpr1,threshold1=roc_curve(y_test,y_test_proba[:,1])
print("AUC value for test data :",roc_auc_score(y_test,y_test_proba[:,1
]))

plt.plot(fpr,tpr,'r',label='train_data')
plt.plot(fpr1,tpr1,'b',label='test data')
plt.legend(loc='upper right')
plt.grid(True)
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('AUC curve for train and test data')
plt.show()
```

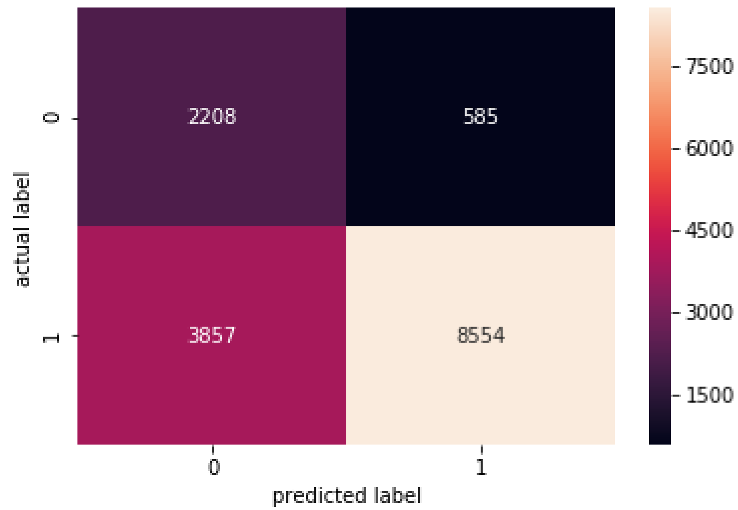
```
AUC value for test data : 0.8110928038929697
```




# Confusion matrix for BOW

In [54]:
```
def confusion_matrix(y_test,test):
    from sklearn.metrics import confusion_matrix
    y_test_predict=dt.predict(test)
    cm_test=confusion_matrix(y_test,y_test_predict)
    import seaborn as sns
    class_label=['0','1']
    df_cm=pd.DataFrame(cm_test,index=class_label,columns=class_label)
    sns.heatmap(df_cm,annot=True,fmt='d')
    plt.xlabel("predicted label")
    plt.ylabel("actual label")
    plt.show()
```

In [55]:
```
confusion_matrix(y_test,X_test_bow)
```




# Top 20 importance features for positive and negative class for BOW

In [73]:
```
# Please write all the code with proper documentation
dt = DecisionTreeClassifier(class_weight= 'balanced',max_depth=20, min_
samples_split=500)
```

```
dt.fit(X_train_bow,y_train)
feat_log = dt.feature_importances_

count_vect = CountVectorizer(min_df=10,max_features=10000,ngram_range=(
1,2))
s = count_vect.fit(X_train)
s = pd.DataFrame(feat_log.T,columns=['+ve'])
s['feature'] = count_vect.get_feature_names()

v = s.sort_values(by = '+ve',kind = 'quicksort',ascending= False)
print("Top 20  important features of positive class", np.array(v['featu
re'][:20]))
print("*"*400)
print("Top 20  important features of negative class",np.array(v.tail(20
)['feature']))
```

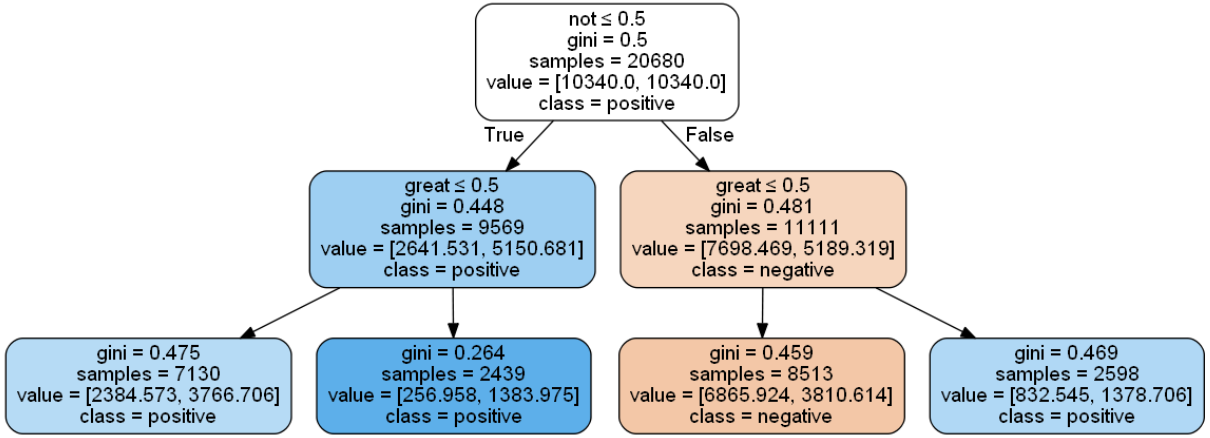
```
Top 20  important features of positive class ['not' 'great' 'best' 'del
icious' 'love' 'perfect' 'good' 'bad' 'loves'
 'excellent' 'disappointed' 'nice' 'favorite' 'wonderful' 'thought'
 'not good' 'worst' 'unfortunately' 'pleased' 'reviews']
***********************************************************************
***********************************************************************
***********************************************************************
***********************************************************************
***********************************************************************
*********************************************
Top 20  important features of negative class ['function' 'fully' 'fruit
s' 'fuller' 'fruits vegetables' 'fruits veggies'
 'fruity' 'fruity taste' 'frustrated' 'frustrating' 'fry' 'frying' 'fud
ge'
 'fuel' 'full' 'full bodied' 'full cup' 'full flavor' 'full flavored'
 'zuke']
```

## Graphviz for BOW

In [80]:
```
# https://github.com/scikit-learn/scikit-learn/issues/9952
from IPython.display import Image
```

```
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier
import pydot
vectorizer=CountVectorizer(ngram_range=(1,2),min_df=10,max_features=10000)
vectorizer.fit(X_train)
feature_name=vectorizer.get_feature_names()

dt=DecisionTreeClassifier(class_weight='balanced',max_depth=2,min_samples_split=500)
dt.fit(X_train_bow,y_train)

dot_data=StringIO()
export_graphviz(dt,out_file=dot_data,feature_names=feature_name,class_names=["negative",'positive'],filled=True,rounded=True,special_characters=True)
graph = pydot.graph_from_dot_data(dot_data.getvalue())[0]
Image(graph.create_png())
```

Out[80]:




# [2] TF-IDF

In [140]:

```
X=preprocessed_reviews
Y=final["Score"]
```

In [141]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.33,shuff
le=False)
X_train,X_cv,y_train,y_cv=train_test_split(X_train,y_train,test_size=0.
33,shuffle=False)
print(np.shape(X_train),y_train.shape)
print(np.shape(X_cv),y_cv.shape)
print(np.shape(X_test),y_test.shape)
```

```
(20680,) (20680,)
(10187,) (10187,)
(15204,) (15204,)
```

In [142]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tf=TfidfVectorizer(min_df=10,max_features=10000,ngram_range=
(1,2))
vectorizer_tf.fit(X_train)

X_train_tf=vectorizer_tf.transform(X_train)
X_cv_tf=vectorizer_tf.transform(X_cv)
X_test_tf=vectorizer_tf.transform(X_test)

print("After TF-IDF vectorizer")
print(np.shape(X_train_tf),y_train.shape)
print(np.shape(X_cv_tf),y_cv.shape)
print(np.shape(X_test_tf),y_test.shape)
```

```
After TF-IDF vectorizer
(20680, 10000) (20680,)
(10187, 10000) (10187,)
(15204, 10000) (15204,)
```

In [96]:

```
depth=[1,5,10,20,50,100,500,1000]
min_split=[5,10,100,500]
parameter={'max_depth':depth,"min_samples_split":min_split}
dt=GridSearchCV(DecisionTreeClassifier(class_weight='balanced'),paramet
er,verbose=1,scoring='roc_auc')
dt.fit(X_train_tf,y_train)
```

```
opt_depth,opt_min_sample_split=dt.best_params_.get("max_depth"),dt.best
_params_.get('min_samples_split')
print("The best optimized depth",opt_depth)
print("the best optimized depth",opt_min_sample_split)

train_score=dt.cv_results_.get("mean_train_score")
cv_score=dt.cv_results_.get('mean_test_score')

plt.plot(np.arange(len(depth)),train_score[::4],'r',label="train_data
(5)")
plt.plot(np.arange(len(depth)),test_score[::4],'r--',label="cv_data(5)"
)

plt.plot(np.arange(len(depth)),train_score[1::4],'g',label="train_data
(10)")
plt.plot(np.arange(len(depth)),test_score[1::4],'g--',label="cv_data(1
0)")

plt.plot(np.arange(len(depth)),train_score[2::4],'b',label="train_data
(100)")
plt.plot(np.arange(len(depth)),test_score[2::4],'b--',label="cv_data(10
0)")

plt.plot(np.arange(len(depth)),train_score[3::4],'y',label="train_data
(500)")
plt.plot(np.arange(len(depth)),test_score[3::4],'y--',label="cv_data(50
0)")

plt.xticks(np.arange(len(depth)),depth)
plt.legend()
plt.xlabel("hyperparameter of depth")
plt.ylabel("Auc")
plt.title("AUC values for train and cv with min_Split")
plt.ylim(0,1)
plt.grid(True)
plt.show
```

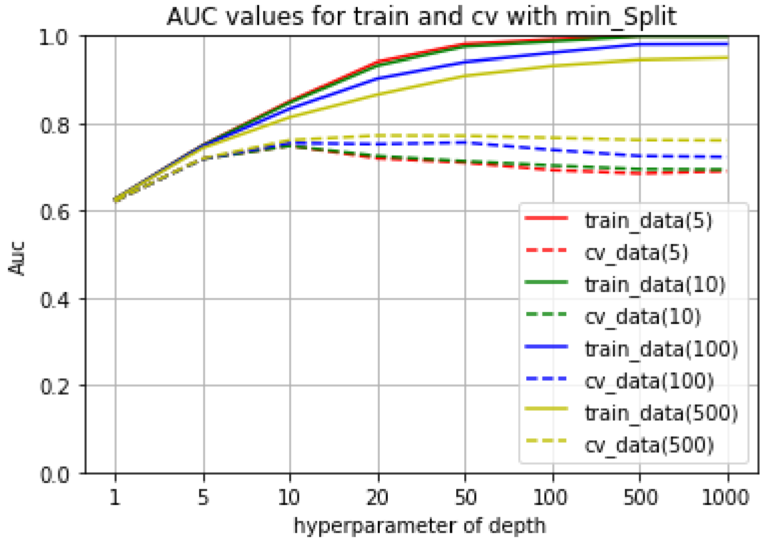
```
Fitting 3 folds for each of 32 candidates, totalling 96 fits
```

```
[Parallel(n_jobs=1)]: Done  96 out of  96 | elapsed:  4.6min finished
```

```
[Parallel(n_jobs=1)]: Done  96 out of  96 | elapsed:  4.6min finished
```

```
The best optimized depth 20
the best optimized depth 500
```

Out[96]: <function matplotlib.pyplot.show(*args, **kw)>

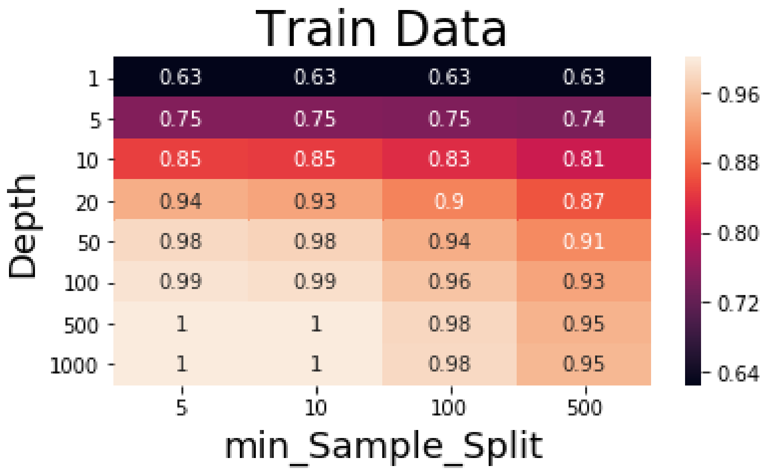


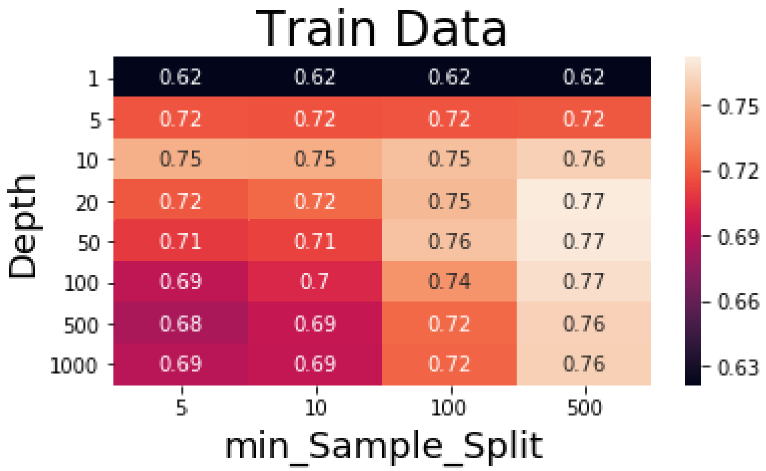

In [98]:

```
df_heatmap = pd. DataFrame(train_score. reshape(8, 4), index=depth, col
umns=min_split )
fig = plt. figure(figsize=(6, 3))
heatmap = sns. heatmap(df_heatmap, annot=True)
plt. ylabel('Depth' , size=18)
plt. xlabel('min_Sample_Split' , size=18)
plt. title("Train Data", size=24)
plt. show

df_heatmap = pd. DataFrame(test_score. reshape(8, 4), index=depth, colu
mns=min_split )
fig = plt. figure(figsize=(6, 3))
heatmap = sns. heatmap(df_heatmap, annot=True)
plt. ylabel('Depth' , size=18)
plt. xlabel('min_Sample_Split' , size=18)
```

```python
plt. title("Train Data", size=24)
plt. show
```

Out[98]: <function matplotlib.pyplot.show(*args, **kw)>

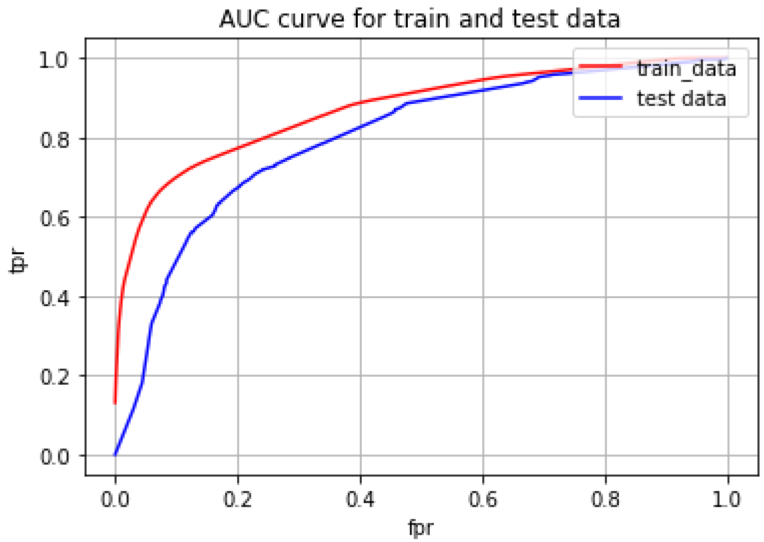



In [143]:
```python
from sklearn.metrics import roc_auc_score
dt=DecisionTreeClassifier(class_weight='balanced',max_depth=20,min_samp
les_split=500)
dt.fit(X_train_tf,y_train)
```

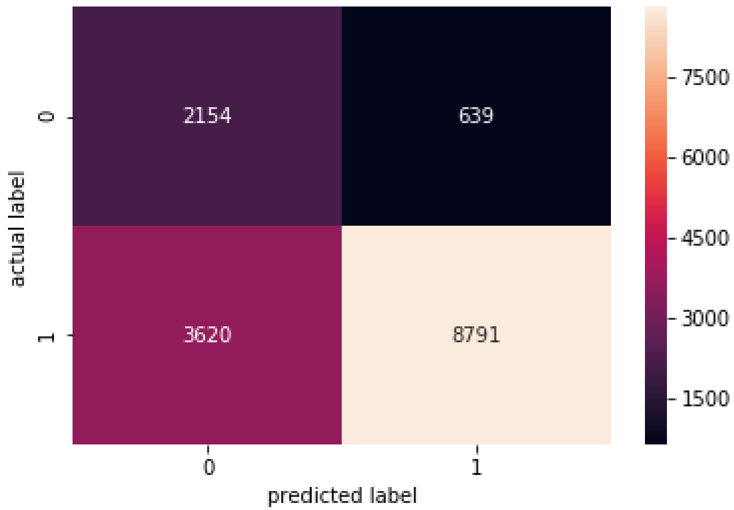
```
y_test_proba=dt.predict_proba(X_test_tf)
y_train_proba=dt.predict_proba(X_train_tf)

fpr,tpr,threshold=roc_curve(y_train,y_train_proba[:,1])
fpr1,tpr1,threshold1=roc_curve(y_test,y_test_proba[:,1])
print("AUC value for test data :",roc_auc_score(y_test,y_test_proba[:,1
]))

plt.plot(fpr,tpr,'r',label='train_data')
plt.plot(fpr1,tpr1,'b',label='test data')
plt.legend(loc='upper right')
plt.grid(True)
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('AUC curve for train and test data')
plt.show()
```

```
AUC value for test data : 0.7985082357816222
```




## Confusion matrix

In [99]:
```
confusion_matrix(y_test,X_test_tf)
```




## Top 20 features importance of positive and negative

In [104]:
```
dt=DecisionTreeClassifier(class_weight='balanced',max_depth=20,min_samp
les_split=500)
dt.fit(X_train_tf,y_train)
feat_log=dt.feature_importances_

vectorizer=TfidfVectorizer(min_df=10,max_features=10000,ngram_range=(1,
2))
s=vectorizer.fit_transform(X_train)
s=pd.DataFrame(feat_log.T,columns=["+ve"])
s['feature']=vectorizer.get_feature_names()
v=s.sort_values(by='+ve',kind='quicksort',ascending=False)
print('top 20 important feature of positive',np.array(v['feature'][:20
]))
print("*"*300)
```

```
print('top 20 important feature of negative',np.array(v.tail(20)['featu
re']))
```

```
top 20 important feature of positive ['not' 'great' 'delicious' 'best'
'love' 'good' 'perfect' 'bad' 'loves'
 'excellent' 'nice' 'wonderful' 'favorite' 'disappointed' 'tasty'
 'thought' 'not good' 'reviews' 'easy' 'terrible']
***********************************************************************
***********************************************************************
***********************************************************************
***********************************************************************
****************
top 20 important feature of negative ['furthermore' 'furniture' 'frustr
ating' 'fur' 'fry' 'frying' 'fudge'
 'fuel' 'full' 'full bodied' 'full cup' 'full flavor' 'full flavored'
 'fuller' 'fully' 'fun' 'function' 'funky' 'funny' 'zuke']
```

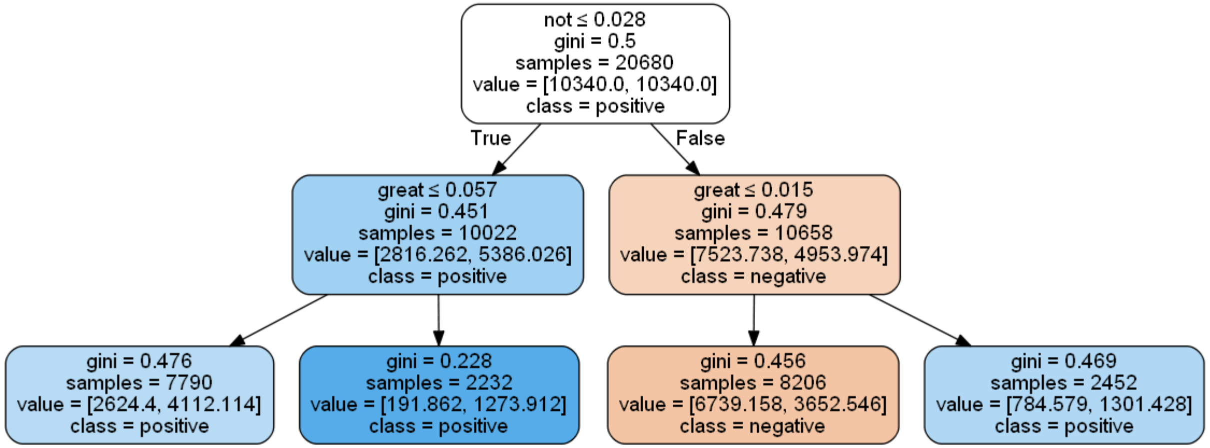
# Graphviz for TF-IDF

In [105]:

```
# https://github.com/scikit-learn/scikit-learn/issues/9952
from IPython.display import Image
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier
import pydot
vectorizer=TfidfVectorizer(ngram_range=(1,2),min_df=10,max_features=100
00)
vectorizer.fit(X_train)
feature_name=vectorizer.get_feature_names()

dt=DecisionTreeClassifier(class_weight='balanced',max_depth=2,min_sampl
es_split=500)
dt.fit(X_train_tf,y_train)

dot_data=StringIO()
export_graphviz(dt,out_file=dot_data,feature_names=feature_name,class_n
ames=["negative",'positive'],filled=True,rounded=True,special_character
```

```
s=True)
graph = pydot.graph_from_dot_data(dot_data.getvalue())[0]
Image(graph.create_png())
```

Out[105]:




## [3] AVG-W2V

In [131]:

```
X=preprocessed_reviews
Y=final['Score']
```

In [132]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.33,shuff
le=False)
X_train,X_cv,y_train,y_cv=train_test_split(X_train,y_train,test_size=0.
33,shuffle=False)

print(np.shape(X_train),y_train.shape)
print(np.shape(X_cv),y_cv.shape)
print(np.shape(X_test),y_test.shape)
```

```
(20680,) (20680,)
(10187,) (10187,)
(15204,) (15204,)
```

In [133]:
```
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [134]:
```
sent_of_train=[]
for sent in X_train:
    sent_of_train.append(sent.split())
```

In [135]:
```
sent_of_cv=[]
for sent in X_cv:
    sent_of_cv.append(sent.split())

sent_of_test=[]
for sent in X_test:
    sent_of_test.append(sent.split())

# Train your own Word2Vec model using your own train text corpus
# min_count = 5 considers only words that occured atleast 5 times
w2v_model=Word2Vec(sent_of_train,min_count=5,size=50, workers=4)

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

In [136]:
```
train_vectors = [];
for sent in sent_of_train:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_vectors.append(sent_vec)

cv_vectors = [];
for sent in sent_of_cv:
```

```
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    cv_vectors.append(sent_vec)




# compute average word2vec for each review for X_test .
test_vectors = [];
for sent in sent_of_test:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    test_vectors.append(sent_vec)
```

In [137]:

```
X_train_wv=train_vectors
X_cv_wv=cv_vectors
X_test_wv=test_vectors
```

In [116]:

```
depth=[1,5,10,20,50,100,500,1000]
min_split=[5,10,100,500]
parameter={'max_depth':depth,"min_samples_split":min_split}
dt=GridSearchCV(DecisionTreeClassifier(class_weight='balanced'),paramet
er,verbose=1,scoring='roc_auc')
dt.fit(X_train_wv,y_train)
```

```
opt_depth,opt_min_sample_split=dt.best_params_.get("max_depth"),dt.best
_params_.get('min_samples_split')
print("The best optimized depth",opt_depth)
print("the best optimized depth",opt_min_sample_split)

train_score=dt.cv_results_.get("mean_train_score")
cv_score=dt.cv_results_.get('mean_test_score')

plt.plot(np.arange(len(depth)),train_score[::4],'r',label="train_data
(5)")
plt.plot(np.arange(len(depth)),test_score[::4],'r--',label="cv_data(5)"
)

plt.plot(np.arange(len(depth)),train_score[1::4],'g',label="train_data
(10)")
plt.plot(np.arange(len(depth)),test_score[1::4],'g--',label="cv_data(1
0)")

plt.plot(np.arange(len(depth)),train_score[2::4],'b',label="train_data
(100)")
plt.plot(np.arange(len(depth)),test_score[2::4],'b--',label="cv_data(10
0)")

plt.plot(np.arange(len(depth)),train_score[3::4],'y',label="train_data
(500)")
plt.plot(np.arange(len(depth)),test_score[3::4],'y--',label="cv_data(50
0)")

plt.xticks(np.arange(len(depth)),depth)
plt.legend()
plt.xlabel("hyperparameter of depth")
plt.ylabel("Auc")
plt.title("AUC values for train and cv with min_Split")
plt.ylim(0,1)
plt.grid(True)
plt.show
```

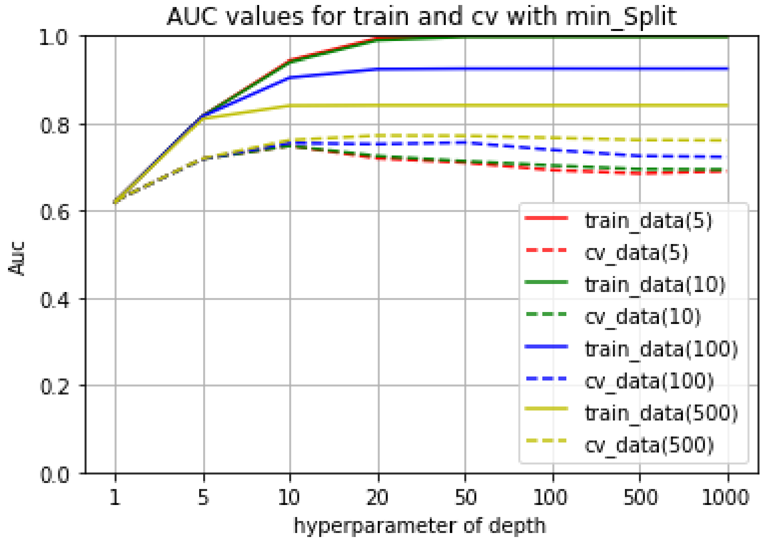
```
Fitting 3 folds for each of 32 candidates, totalling 96 fits
```

```
[Parallel(n_jobs=1)]: Done  96 out of  96 | elapsed:  1.5min finished
```

```
[Parallel(n_jobs=1)]: Done  96 out of  96 | elapsed:  1.5min finished
```

```
The best optimized depth 10
the best optimized depth 500
```

Out[116]: <function matplotlib.pyplot.show(*args, **kw)>

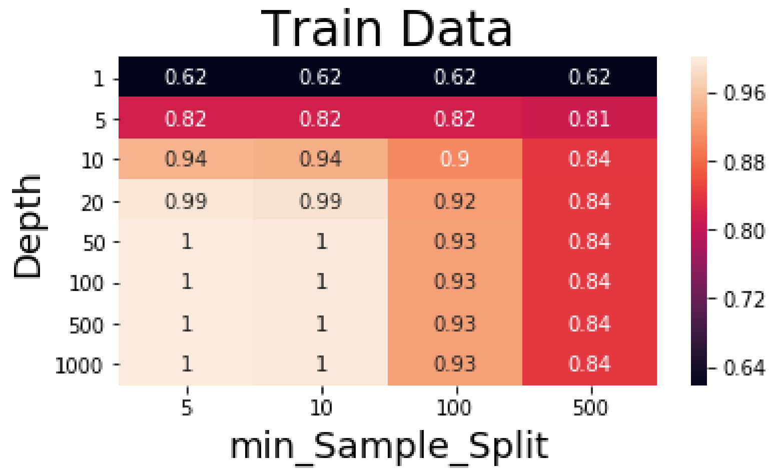


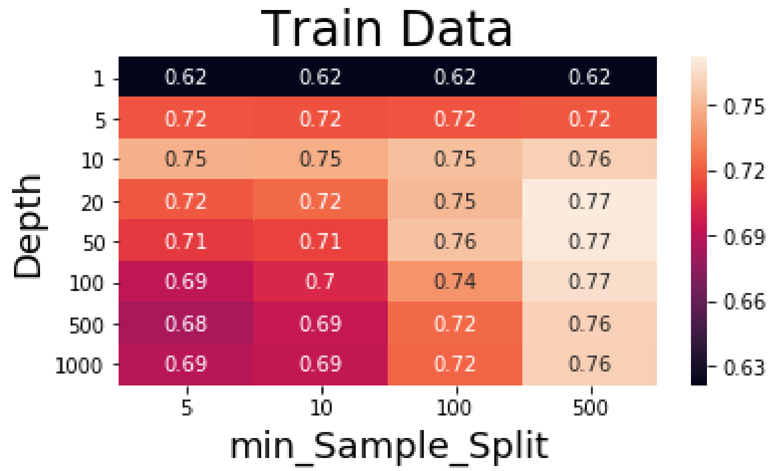

In [117]:

```
df_heatmap = pd. DataFrame(train_score. reshape(8, 4), index=depth, col
umns=min_split )
fig = plt. figure(figsize=(6, 3))
heatmap = sns. heatmap(df_heatmap, annot=True)
plt. ylabel('Depth' , size=18)
plt. xlabel('min_Sample_Split' , size=18)
plt. title("Train Data", size=24)
plt. show

df_heatmap = pd. DataFrame(test_score. reshape(8, 4), index=depth, colu
mns=min_split )
fig = plt. figure(figsize=(6, 3))
heatmap = sns. heatmap(df_heatmap, annot=True)
plt. ylabel('Depth' , size=18)
plt. xlabel('min_Sample_Split' , size=18)
```

```
plt. title("Train Data", size=24)
plt. show
```

Out[117]: <function matplotlib.pyplot.show(*args, **kw)>



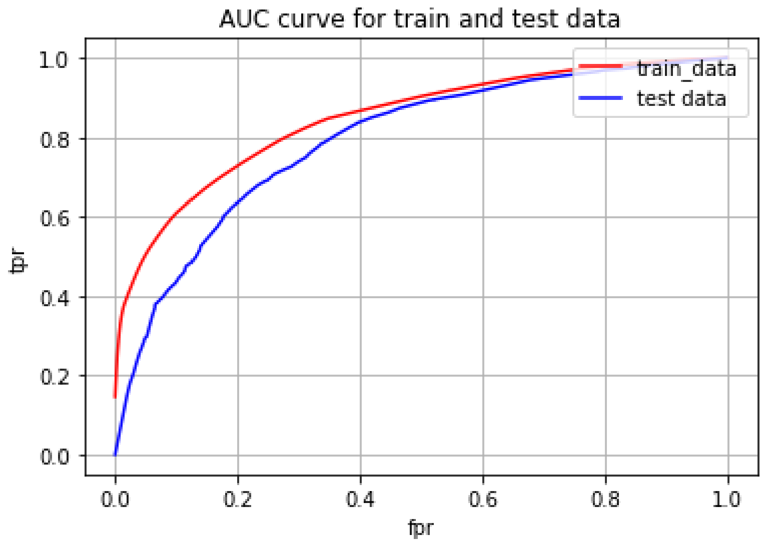




In [138]:

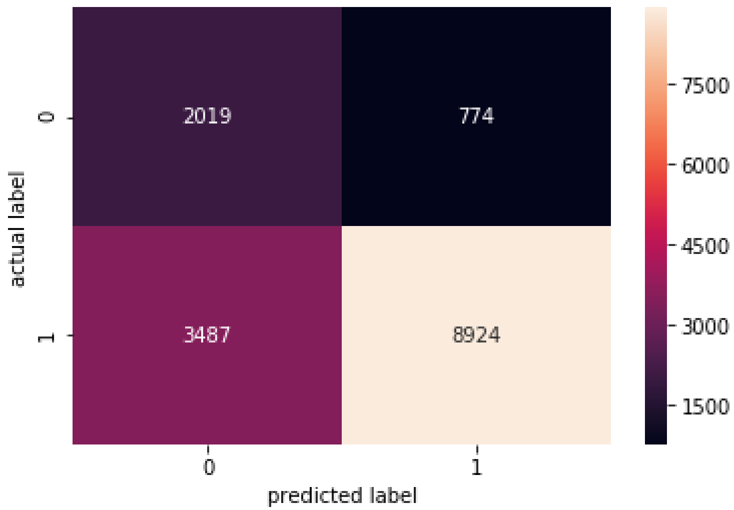
```
from sklearn.metrics import roc_auc_score
dt=DecisionTreeClassifier(class_weight='balanced',max_depth=10,min_samp
les_split=500)
dt.fit(X_train_wv,y_train)
```

```
y_test_proba=dt.predict_proba(X_test_wv)
y_train_proba=dt.predict_proba(X_train_wv)

fpr,tpr,threshold=roc_curve(y_train,y_train_proba[:,1])
fpr1,tpr1,threshold1=roc_curve(y_test,y_test_proba[:,1])
print("AUC value for test data :",roc_auc_score(y_test,y_test_proba[:,1
]))

plt.plot(fpr,tpr,'r',label='train_data')
plt.plot(fpr1,tpr1,'b',label='test data')
plt.legend(loc='upper right')
plt.grid(True)
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('AUC curve for train and test data')
plt.show()
```

```
AUC value for test data : 0.7936668766544397
```




In [139]:
```
confusion_matrix(y_test,X_test_wv)
```




# TF-IDF W2V

In [119]:
```
X=preprocessed_reviews
Y=final["Score"]
```

In [120]:
```
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
3,shuffle=False) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33,shuffle=False)
```

In [121]:
```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_train)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [122]:
```
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
```

```
ll_val = tfidf

tfidf_train_vectors = []; # the tfidf-w2v for each sentence/review is s
tored in this list
row=0;
for sent in tqdm(sent_of_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/r
eview
    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_train_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████| 20680/20680 [05:03<00:00, 6
8.09it/s]
```

In [123]:
```
tfidf_cv_vectors = []; # the tfidf-w2v for each sentence/review is stor
ed in this list
row=0;
for sent in tqdm(sent_of_cv): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/r
eview
    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_cv_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████| 10187/10187 [02:28<00:00, 6
8.53it/s]
```

In [124]:

```
tfidf_test_vectors = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in tqdm(sent_of_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/r
eview
    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_test_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████| 15204/15204 [03:23<00:00, 7
4.55it/s]
```

In [125]:

```
X_train_tw=tfidf_train_vectors
```

```
X_cv_tw=tfidf_cv_vectors
X_test_tw=tfidf_test_vectors
```

In [126]:

```
depth=[1,5,10,20,50,100,500,1000]
min_split=[5,10,100,500]
parameter={'max_depth':depth,"min_samples_split":min_split}
dt=GridSearchCV(DecisionTreeClassifier(class_weight='balanced'),paramet
er,verbose=1,scoring='roc_auc')
dt.fit(X_train_tw,y_train)

opt_depth,opt_min_sample_split=dt.best_params_.get("max_depth"),dt.best
_params_.get('min_samples_split')
print("The best optimized depth",opt_depth)
print("the best optimized depth",opt_min_sample_split)

train_score=dt.cv_results_.get("mean_train_score")
cv_score=dt.cv_results_.get('mean_test_score')

plt.plot(np.arange(len(depth)),train_score[::4],'r',label="train_data
(5)")
plt.plot(np.arange(len(depth)),test_score[::4],'r--',label="cv_data(5)"
)

plt.plot(np.arange(len(depth)),train_score[1::4],'g',label="train_data
(10)")
plt.plot(np.arange(len(depth)),test_score[1::4],'g--',label="cv_data(1
0)")

plt.plot(np.arange(len(depth)),train_score[2::4],'b',label="train_data
(100)")
plt.plot(np.arange(len(depth)),test_score[2::4],'b--',label="cv_data(10
0)")

plt.plot(np.arange(len(depth)),train_score[3::4],'y',label="train_data
(500)")
plt.plot(np.arange(len(depth)),test_score[3::4],'y--',label="cv_data(50
0)")

plt.xticks(np.arange(len(depth)),depth)
```

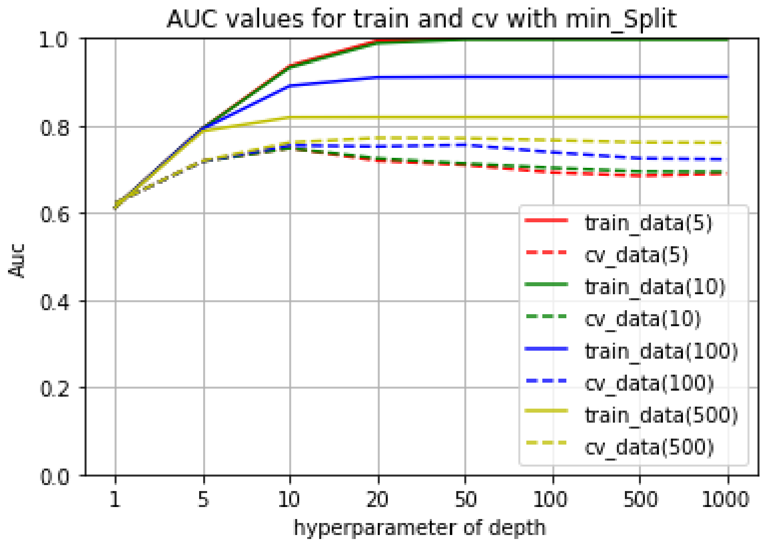
```
plt.legend()
plt.xlabel("hyperparameter of depth")
plt.ylabel("Auc")
plt.title("AUC values for train and cv with min_Split")
plt.ylim(0,1)
plt.grid(True)
plt.show
```

```
Fitting 3 folds for each of 32 candidates, totalling 96 fits
```

```
[Parallel(n_jobs=1)]: Done  96 out of  96 | elapsed:  1.4min finished
```

```
The best optimized depth 20
the best optimized depth 500
```

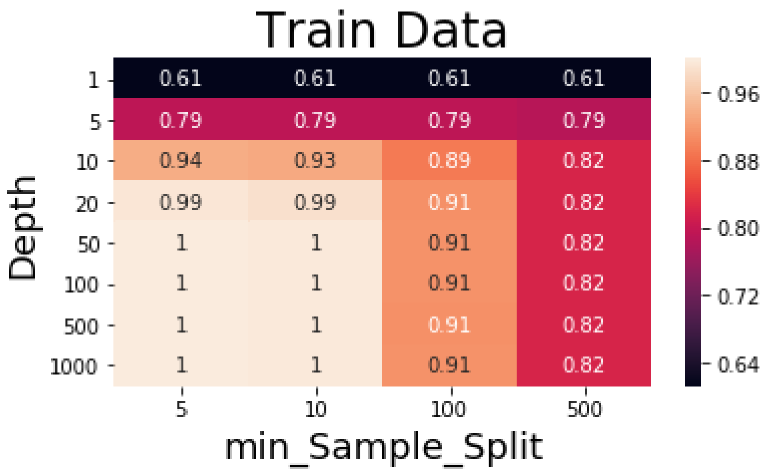
Out[126]: <function matplotlib.pyplot.show(*args, **kw)>

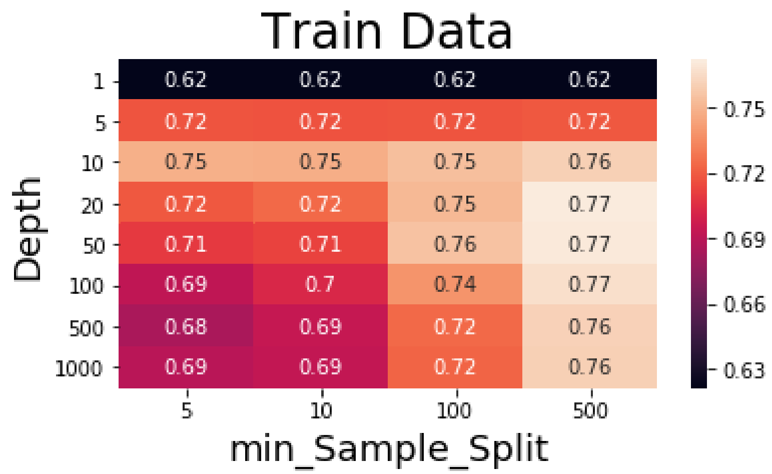



In [127]:
```
df_heatmap = pd. DataFrame(train_score. reshape(8, 4), index=depth, col
umns=min_split )
fig = plt. figure(figsize=(6, 3))
heatmap = sns. heatmap(df_heatmap, annot=True)
plt. ylabel('Depth' , size=18)
plt. xlabel('min_Sample_Split' , size=18)
```

```
plt. title("Train Data", size=24)
plt. show

df_heatmap = pd. DataFrame(test_score. reshape(8, 4), index=depth, colu
mns=min_split )
fig = plt. figure(figsize=(6, 3))
heatmap = sns. heatmap(df_heatmap, annot=True)
plt. ylabel('Depth' , size=18)
plt. xlabel('min_Sample_Split' , size=18)
plt. title("Train Data", size=24)
plt. show
```

Out[127]: <function matplotlib.pyplot.show(*args, **kw)>

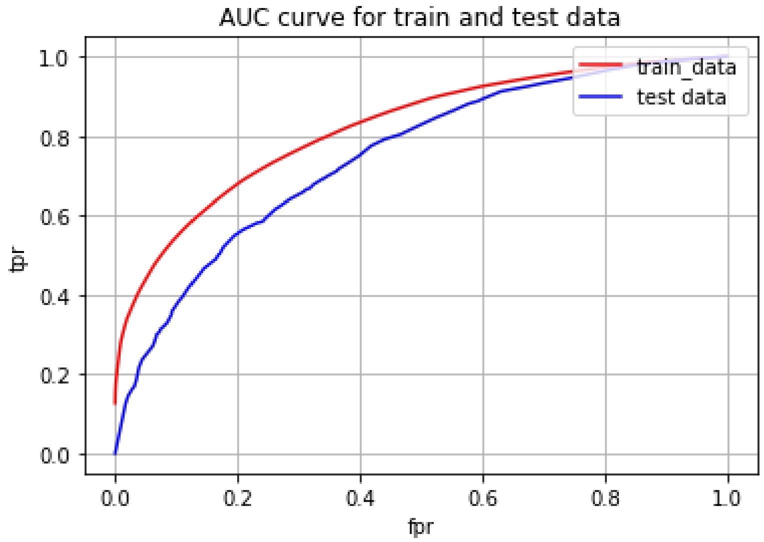




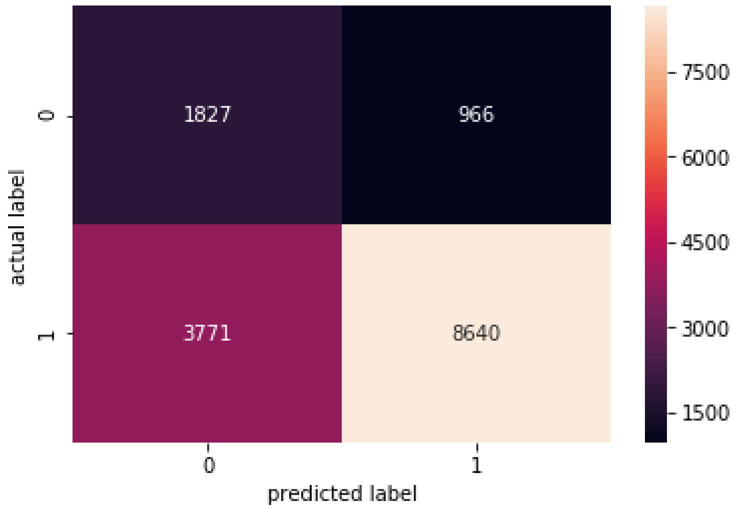

In [130]:

```
from sklearn.metrics import roc_auc_score
dt=DecisionTreeClassifier(class_weight='balanced',max_depth=20,min_samp
les_split=500)
dt.fit(X_train_tw,y_train)

y_test_proba=dt.predict_proba(X_test_tw)
y_train_proba=dt.predict_proba(X_train_tw)

fpr,tpr,threshold=roc_curve(y_train,y_train_proba[:,1])
fpr1,tpr1,threshold1=roc_curve(y_test,y_test_proba[:,1])
print("AUC value for test data :",roc_auc_score(y_test,y_test_proba[:,1
]))

plt.plot(fpr,tpr,'r',label='train_data')
plt.plot(fpr1,tpr1,'b',label='test data')
plt.legend(loc='upper right')
plt.grid(True)
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('AUC curve for train and test data')
plt.show()
```

```
AUC value for test data : 0.7494858703672981
```




In [129]:
```
confusion_matrix(y_test,X_test_tw)
```




## Conclusion

In [149]:
```
from tabulate import tabulate
print(tabulate ([['BOW(20)', 500, 81],['TF-IDF(20)',500,79],['AVG-W2V(10)',500,79] , ['TFIDF-W2V(20)',500,74]],    headers=['Vectorizer(opt_depth)', 'best_min_sample_split','AUC_test']))
```

```
Vectorizer(opt_depth)      best_min_sample_split    AUC_test
-----------------------  -----------------------  ----------
BOW(20)                                      500          81
TF-IDF(20)                                   500          79
AVG-W2V(10)                                  500          79
TFIDF-W2V(20)                                500          74
```

1. cost of computation is fast.
2. The best model is Bag of words.
3. we can improve the model by taking more datapoints.

# [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 paramter 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-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

# Applying Decision Trees

## [5.1] Applying Decision Trees on BOW, SET 1

In [0]:
```
# Please write all the code with proper documentation
```

### [5.1.1] Top 20 important features from SET 1

In [0]:
```
# Please write all the code with proper documentation
```

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

In [0]:
```
# Please write all the code with proper documentation
```

## [5.2] Applying Decision Trees on TFIDF, SET 2

In [0]:
```
# Please write all the code with proper documentation
```

### [5.2.1] Top 20 important features from SET 2

In [0]:
```
# Please write all the code with proper documentation
```

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

In [0]:
```
# Please write all the code with proper documentation
```

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

In [0]:
```
# Please write all the code with proper documentation
```

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

In [0]:
```
# Please write all the code with proper documentation
```

# [6] Conclusions

In [0]:
```
# Please compare all your models using Prettytable library
```