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.

[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
```

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 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 5000""", 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 (5000, 10)
```

Out[2]:

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

In [3]:

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

In [4]:

```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[4]:

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

In [5]:

```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)

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

In [6]:

```
display['COUNT(*)'].sum()
```

Out[6]:

```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

In [7]:

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

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenon
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenon
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

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

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

In [8]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=Tr
ue, 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]:

```
(4986, 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]:

```
99.72
```

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	HelpfulnessDenom
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	

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

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

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

```
(4986, 10)
```

Out[13]:
```
1    4178
0     808
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

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

In [14]:

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

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)
```

```
Why is this $[...] when the same product is available for $[...] here?<
br />http://www.amazon.com/VICTOR-FLY-MAGNET-BAIT-REFILL/dp/B00004RBDY<
br /><br />The Victor M380 and M502 traps are unreal, of course -- tota
l fly genocide. Pretty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious t
hese chips are.  The best thing was that there were a lot of "brown" ch
ips in the bsg (my favorite), so I bought some more through amazon and
shared with family and friends.  I am a little disappointed that there
are not, so far, very many brown chips in these bags, but the flavor is
still very good.  I like them better than the yogurt and green onion fl
avor because they do not seem to be as salty, and the onion flavor is b
etter.  If you haven't eaten Kettle chips before, I recommend that you
try a bag before buying bulk.  They are thicker and crunchier than Lays
but just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what the
y were ordering; the other wants crispy cookies.  Hey, I'm sorry; but t
hese reviews do nobody any good beyond reminding us to look  before ord
ering.<br /><br />These are chocolate-oatmeal cookies.  If you don't li
ke that combination, don't order this type of cookie.  I find the combo
quite nice, really.  The oatmeal sort of "calms" the rich chocolate fla
vor and gives the cookie sort of a coconut-type consistency.  Now let's
also remember that tastes differ; so, I've given my opinion.<br /><br /
>Then, these are soft, chewy cookies -- as advertised.  They are not "c
rispy" cookies, or the blurb would say "crispy," rather than "chewy."
I happen to like raw cookie dough; however, I don't see where these tas
te like raw cookie dough.  Both are soft, however, so is this the confu
sion?  And, yes, they stick together.  Soft cookies tend to do that.  T
hey aren't individually wrapped, which would add to the cost.  Oh yeah,
chocolate chip cookies tend to be somewhat sweet.<br /><br />So, if you
want something hard and crisp, I suggest Nabiso's Ginger Snaps.  If you
want a cookie that's soft, chewy and tastes like a combination of choco
late and oatmeal, give these a try.  I'm here to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.<br />Thi
s k cup is great coffee.  dcaf is very good as well
==================================================
```

In [15]:

```python
# 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)
```

```
Why is this $[...] when the same product is available for $[...] here?<
br /> /><br />The Victor M380 and M502 traps are unreal, of course -- t
otal fly genocide. Pretty stinky, but only right nearby.
```

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)
```

```
Why is this $[...] when the same product is available for $[...] here?
/>The Victor M380 and M502 traps are unreal, of course -- total fly gen
ocide. Pretty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious t
hese chips are.  The best thing was that there were a lot of "brown" ch
ips in the bsg (my favorite), so I bought some more through amazon and
shared with family and friends.  I am a little disappointed that there
```

```
are not, so far, very many brown chips in these bags, but the flavor is
still very good.  I like them better than the yogurt and green onion fl
avor because they do not seem to be as salty, and the onion flavor is b
etter.  If you haven't eaten Kettle chips before, I recommend that you
try a bag before buying bulk.  They are thicker and crunchier than Lays
but just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what the
y were ordering; the other wants crispy cookies.  Hey, I'm sorry; but t
hese reviews do nobody any good beyond reminding us to look  before ord
ering.These are chocolate-oatmeal cookies.  If you don't like that comb
ination, don't order this type of cookie.  I find the combo quite nice,
really.  The oatmeal sort of "calms" the rich chocolate flavor and give
s the cookie sort of a coconut-type consistency.  Now let's also rememb
er that tastes differ; so, I've given my opinion.Then, these are soft,
chewy cookies -- as advertised.  They are not "crispy" cookies, or the
blurb would say "crispy," rather than "chewy."  I happen to like raw co
okie dough; however, I don't see where these taste like raw cookie doug
h.  Both are soft, however, so is this the confusion?  And, yes, they s
tick together.  Soft cookies tend to do that.  They aren't individually
wrapped, which would add to the cost.  Oh yeah, chocolate chip cookies
tend to be somewhat sweet.So, if you want something hard and crisp, I s
uggest Nabiso's Ginger Snaps.  If you want a cookie that's soft, chewy
and tastes like a combination of chocolate and oatmeal, give these a tr
y.  I'm here to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.This k cu
p is great coffee.  dcaf is very good as well
```

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)
```

```
Wow.  So far, two two-star reviews.  One obviously had no idea what the
y were ordering; the other wants crispy cookies.  Hey, I am sorry; but
these reviews do nobody any good beyond reminding us to look  before or
dering.<br /><br />These are chocolate-oatmeal cookies.  If you do not
like that combination, do not order this type of cookie.  I find the co
mbo quite nice, really.  The oatmeal sort of "calms" the rich chocolate
flavor and gives the cookie sort of a coconut-type consistency.  Now le
t is also remember that tastes differ; so, I have given my opinion.<br
/><br />Then, these are soft, chewy cookies -- as advertised.  They are
not "crispy" cookies, or the blurb would say "crispy," rather than "che
wy."  I happen to like raw cookie dough; however, I do not see where th
ese taste like raw cookie dough.  Both are soft, however, so is this th
e confusion?  And, yes, they stick together.  Soft cookies tend to do t
hat.  They are not individually wrapped, which would add to the cost.
Oh yeah, chocolate chip cookies tend to be somewhat sweet.<br /><br />S
o, if you want something hard and crisp, I suggest Nabiso is Ginger Sna
ps.  If you want a cookie that is soft, chewy and tastes like a combina
tion of chocolate and oatmeal, give these a try.  I am here to place my
second order.
==================================================
```

In [19]:

```
#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)
```

```
Why is this $[...] when the same product is available for $[...] here?<
br /> /><br />The Victor  and  traps are unreal, of course -- total fly
genocide. Pretty stinky, but only right nearby.
```

In [20]:

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

```
Wow So far two two star reviews One obviously had no idea what they wer
e ordering the other wants crispy cookies Hey I am sorry but these revi
ews do nobody any good beyond reminding us to look before ordering br b
r These are chocolate oatmeal cookies If you do not like that combinati
on do not order this type of cookie I find the combo quite nice really
The oatmeal sort of calms the rich chocolate flavor and gives the cooki
e sort of a coconut type consistency Now let is also remember that tast
es differ so I have given my opinion br br Then these are soft chewy co
okies as advertised They are not crispy cookies or the blurb would say
crispy rather than chewy I happen to like raw cookie dough however I do
not see where these taste like raw cookie dough Both are soft however s
o is this the confusion And yes they stick together Soft cookies tend t
o do that They are not individually wrapped which would add to the cost
Oh yeah chocolate chip cookies tend to be somewhat sweet br br So if yo
u want something hard and crisp I suggest Nabiso is Ginger Snaps If you
want a cookie that is soft chewy and tastes like a combination of choco
late and oatmeal give these a try I am here to place my second order
```

In [21]:

```
# 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 [22]:

```
# 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%|██████████████████████████████████████████████████████████████████
██████████| 4986/4986 [00:01<00:00, 3137.37it/s]
```

In [23]:
```
preprocessed_reviews[1500]
```

Out[23]:
```
'wow far two two star reviews one obviously no idea ordering wants cris
py cookies hey sorry reviews nobody good beyond reminding us look order
ing chocolate oatmeal cookies not like combination not order type cooki
e find combo quite nice really oatmeal sort calms rich chocolate flavor
gives cookie sort coconut type consistency let also remember tastes dif
fer given opinion soft chewy cookies advertised not crispy cookies blur
b would say crispy rather chewy happen like raw cookie dough however no
t see taste like raw cookie dough soft however confusion yes stick toge
ther soft cookies tend not individually wrapped would add cost oh yeah
chocolate chip cookies tend somewhat sweet want something hard crisp su
ggest nabiso ginger snaps want cookie soft chewy tastes like combinatio
n chocolate oatmeal give try place second order'
```

[3.2] Preprocessing Review Summary

In [6]:
```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [25]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(preprocessed_reviews)
```

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

```
some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abb
ott', 'abby', 'abdominal', 'abiding', 'ability']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

[4.2] Bi-Grams and n-Grams.

In [26]:

```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-gra
ms
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.
org/stable/modules/generated/sklearn.feature_extraction.text.CountVecto
rizer.html

# you can choose these numebrs min_df=10, max_features=5000, of your ch
oice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features
=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_s
hape())
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  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.3] TF-IDF

In [27]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.ge
t_feature_names()[0:10])
print('='*50)

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

```
some sample features(unique words in the corpus) ['ability', 'able', 'a
ble find', 'able get', 'absolute', 'absolutely', 'absolutely deliciou
s', 'absolutely love', 'absolutely no', 'according']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.4] Word2Vec

In [28]:

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

In [42]:

```
# 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 val
ues
# To use this code-snippet, download "GoogleNews-vectors-negative300.bi
n"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edi
t
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17
SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors
-negative300.bin', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_trai
n_w2v = True, to train your own w2v ")
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wond
erful', 0.9946032166481018), ('excellent', 0.9944332838058472), ('espec
ially', 0.9941144585609436), ('baked', 0.9940600395202637), ('salted',
0.994047224521637), ('alternative', 0.9937226176261902), ('tasty', 0.99
36816692352295), ('healthy', 0.9936649799346924)]
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('p
opcorn', 0.9992750883102417), ('de', 0.9992610216140747), ('miss', 0.99
92451071739197), ('melitta', 0.999218761920929), ('choice', 0.999210238
4567261), ('american', 0.9991837739944458), ('beef', 0.999178051948547
4), ('finish', 0.9991567134857178)]
```

In [36]:

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

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'st
inky', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'receiv
ed', 'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'ins
tead', 'removed', 'easily', 'daughter', 'designed', 'printed', 'use',
'car', 'windows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fu
n', 'everywhere', 'like', 'tv', 'computer', 'really', 'good', 'idea',
'final', 'outstanding', 'window', 'everybody', 'asks', 'bought', 'mad
e']
```

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

[4.4.1.1] Avg W2v

In [38]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in
 this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u 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/re
view
```

```
    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%|██████████████████████████████████████████████████████████████████
██████████| 4986/4986 [00:03<00:00, 1330.47it/s]
```

```
4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [39]:

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

In [41]:

```
# 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 ce
ll_val = tfidf

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

```
100%|██████████████████████████████████████████████████████████████████
███████████| 4986/4986 [00:20<00:00, 245.63it/s]
```

[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 10 features of positive class and top 10 features of negative class for 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

In [2]:
```
# after preprocessing
df = pd.read_pickle("files/preprocessed.pkl")
df.shape
```

Out[2]:
```
(364171, 12)
```

In [3]:
```
df.head()
```

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulness

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulness
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	
138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	
138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1	
138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3	

In [4]:

```
from sklearn.model_selection import train_test_split
from sklearn.grid_search import GridSearchCV
from sklearn.datasets import *
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score , f1_score , confusion_matri
x
from collections import Counter
from sklearn.metrics import accuracy_score, roc_auc_score , roc_curve
from sklearn.model_selection import train_test_split
```

```
/home/prasad/anaconda3/lib/python3.6/site-packages/sklearn/cross_valida
tion.py:41: DeprecationWarning: This module was deprecated in version
0.18 in favor of the model_selection module into which all the refactor
ed classes and functions are moved. Also note that the interface of the
new CV iterators are different from that of this module. This module wi
ll be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
/home/prasad/anaconda3/lib/python3.6/site-packages/sklearn/grid_search.
py:42: DeprecationWarning: This module was deprecated in version 0.18 i
n favor of the model_selection module into which all the refactored cla
sses and functions are moved. This module will be removed in 0.20.
  DeprecationWarning)
```

In [5]:

```
# take 50k sample data randomly
sample_data = df.sample(100000)
sample_data.shape
```

Out[5]:

```
(100000, 12)
```

In [6]:

```
# sorted the data using time based
sorted_data = sample_data.sort_values('Time', axis=0, inplace=False)
sorted_data.shape
```

Out[6]:

```
(100000, 12)
```

In [7]:

```
sorted_data['Score'].value_counts()
```

Out[7]:

```
1    84419
0    15581
Name: Score, dtype: int64
```

In [8]:

```
X = np.array(sorted_data['CleanedText'])
y = np.array(sorted_data['Score'])
print(X.shape)
print(y.shape)
```

```
(100000,)
(100000,)
```

In [9]:

```
# Simple cross validation
# split the data sent into train and test
train , test , train_y , test_y = train_test_split(X, y, test_size = 0.
3, random_state=None)

# split the train data set into cross validation train and cross valida
tion test
train, cv , train_y, cv_y = train_test_split(train, train_y, test_size=
0.3, random_state=None)

print("train data = ", train.shape)
print("cros validation = ", cv.shape)
print("test data = ", test.shape)
```

```
train data =  (49000,)
cros validation =  (21000,)
test data =  (30000,)
```

[5.1] Applying Decision Trees on BOW, SET 1

In [10]:

```
# Please write all the code with proper documentation
#BoW
count_vect = CountVectorizer(min_df=20) #in scikit-learn
count_vect.fit(train)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

bow_train = count_vect.fit_transform(train)
bow_cv = count_vect.transform(cv)
```

```
bow_test = count_vect.transform(test)
print("============Train Data=========")
print("the type of count vectorizer ",type(bow_train))
print("the shape of out text BOW vectorizer ",bow_train.get_shape())
print("the number of unique words ", bow_train.get_shape()[1])
print("============Cross validation Data=========")
print("the type of count vectorizer ",type(bow_cv))
print("the shape of out text BOW vectorizer ",bow_cv.get_shape())
print("the number of unique words ", bow_cv.get_shape()[1])
print("============Test Data=========")
print("the type of count vectorizer ",type(bow_test))
print("the shape of out text BOW vectorizer ",bow_test.get_shape())
print("the number of unique words ", bow_test.get_shape()[1])
```

```
some feature names  ['ability', 'able', 'absolute', 'absolutely', 'abso
rb', 'absorbed', 'absorbs', 'abundance', 'acai', 'accent']
==================================================
============Train Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (49000, 5975)
the number of unique words  5975
============Cross validation Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (21000, 5975)
the number of unique words  5975
============Test Data=========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (30000, 5975)
the number of unique words  5975
```

In [11]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [12]:

```
depth = [1, 5, 10, 50, 100, 500, 1000]
samples = [5, 10, 100, 500]
bow_train_auc = []
bow_cv_auc = []
for i in depth:
    for j in samples:
        DTC = DecisionTreeClassifier(criterion='gini',max_depth=i, min_
samples_split=j )
        DTC.fit(bow_train, train_y)
        # train data
        y_prob_train = DTC.predict_proba(bow_train)[:,1]
        y_pred = np.where(y_prob_train > 0.5, 1, 0)
        auc_roc_train = roc_auc_score(train_y , y_prob_train)
        print('\nTrain AUC for min_sample = %s and max_depth = %s is %
0.2f%%' % (str(j),str(i),(auc_roc_train * float(100))))
        bow_train_auc.append(auc_roc_train)
        # CV
        y_prob_cv = DTC.predict_proba(bow_cv)[:,1]
        y_pred = np.where(y_prob_cv > 0.5, 1, 0)
        auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
        print('\nCV AUC for min_sample = %s and max_depth = %s is %0.2
f%%' % (str(j),str(i),(auc_roc_cv * float(100))))
        bow_cv_auc.append(auc_roc_cv)
        print("="*50)
```

```
Train AUC for min_sample = 5 and max_depth = 1 is 60.69%

CV AUC for min_sample = 5 and max_depth = 1 is 60.20%
==================================================

Train AUC for min_sample = 10 and max_depth = 1 is 60.69%

CV AUC for min_sample = 10 and max_depth = 1 is 60.20%
==================================================

Train AUC for min_sample = 100 and max_depth = 1 is 60.69%

CV AUC for min_sample = 100 and max_depth = 1 is 60.20%
==================================================

Train AUC for min_sample = 500 and max_depth = 1 is 60.69%

CV AUC for min_sample = 500 and max_depth = 1 is 60.20%
==================================================

Train AUC for min_sample = 5 and max_depth = 5 is 71.30%
```

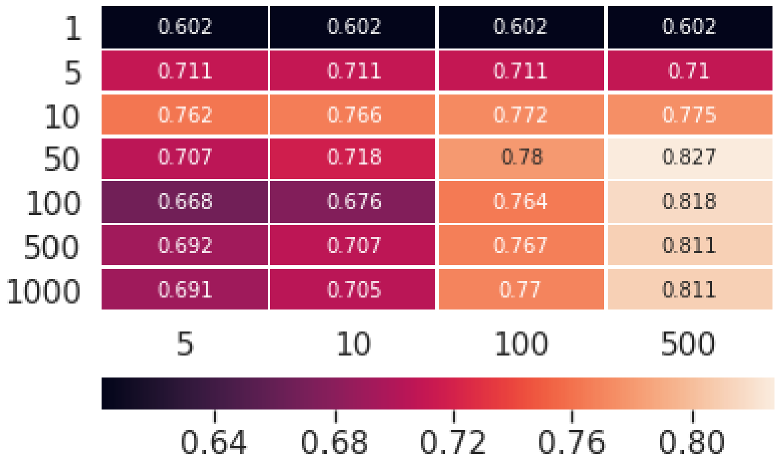
```
CV AUC for min_sample = 5 and max_depth = 5 is 71.06%
==================================================

Train AUC for min_sample = 10 and max_depth = 5 is 71.30%

CV AUC for min_sample = 10 and max_depth = 5 is 71.06%
==================================================

Train AUC for min_sample = 100 and max_depth = 5 is 71.25%

CV AUC for min_sample = 100 and max_depth = 5 is 71.08%
==================================================

Train AUC for min_sample = 500 and max_depth = 5 is 71.17%

CV AUC for min_sample = 500 and max_depth = 5 is 71.00%
==================================================

Train AUC for min_sample = 5 and max_depth = 10 is 79.30%

CV AUC for min_sample = 5 and max_depth = 10 is 76.22%
==================================================

Train AUC for min_sample = 10 and max_depth = 10 is 79.23%

CV AUC for min_sample = 10 and max_depth = 10 is 76.61%
==================================================

Train AUC for min_sample = 100 and max_depth = 10 is 78.58%

CV AUC for min_sample = 100 and max_depth = 10 is 77.23%
==================================================

Train AUC for min_sample = 500 and max_depth = 10 is 78.10%

CV AUC for min_sample = 500 and max_depth = 10 is 77.51%
==================================================

Train AUC for min_sample = 5 and max_depth = 50 is 97.20%
```

```
CV AUC for min_sample = 5 and max_depth = 50 is 70.66%
==================================================

Train AUC for min_sample = 10 and max_depth = 50 is 96.79%

CV AUC for min_sample = 10 and max_depth = 50 is 71.77%
==================================================

Train AUC for min_sample = 100 and max_depth = 50 is 94.54%

CV AUC for min_sample = 100 and max_depth = 50 is 78.02%
==================================================

Train AUC for min_sample = 500 and max_depth = 50 is 90.04%

CV AUC for min_sample = 500 and max_depth = 50 is 82.73%
==================================================

Train AUC for min_sample = 5 and max_depth = 100 is 99.32%

CV AUC for min_sample = 5 and max_depth = 100 is 66.80%
==================================================

Train AUC for min_sample = 10 and max_depth = 100 is 99.06%

CV AUC for min_sample = 10 and max_depth = 100 is 67.62%
==================================================

Train AUC for min_sample = 100 and max_depth = 100 is 97.04%

CV AUC for min_sample = 100 and max_depth = 100 is 76.43%
==================================================

Train AUC for min_sample = 500 and max_depth = 100 is 92.80%

CV AUC for min_sample = 500 and max_depth = 100 is 81.75%
==================================================
Train AUC for min_sample = 5 and max_depth = 500 is 99.93%

CV AUC for min_sample = 5 and max_depth = 500 is 69.20%
==================================================

Train AUC for min_sample = 10 and max_depth = 500 is 99.69%

CV AUC for min_sample = 10 and max_depth = 500 is 70.69%
==================================================

Train AUC for min_sample = 100 and max_depth = 500 is 97.92%

CV AUC for min_sample = 100 and max_depth = 500 is 76.69%
==================================================

Train AUC for min_sample = 500 and max_depth = 500 is 93.93%

CV AUC for min_sample = 500 and max_depth = 500 is 81.10%
==================================================

Train AUC for min_sample = 5 and max_depth = 1000 is 99.93%

CV AUC for min_sample = 5 and max_depth = 1000 is 69.13%
==================================================

Train AUC for min_sample = 10 and max_depth = 1000 is 99.71%

CV AUC for min_sample = 10 and max_depth = 1000 is 70.51%
==================================================

Train AUC for min_sample = 100 and max_depth = 1000 is 97.92%

CV AUC for min_sample = 100 and max_depth = 1000 is 77.00%
==================================================

Train AUC for min_sample = 500 and max_depth = 1000 is 93.96%

CV AUC for min_sample = 500 and max_depth = 1000 is 81.11%
==================================================
```

In [13]:
```
import mglearn
```

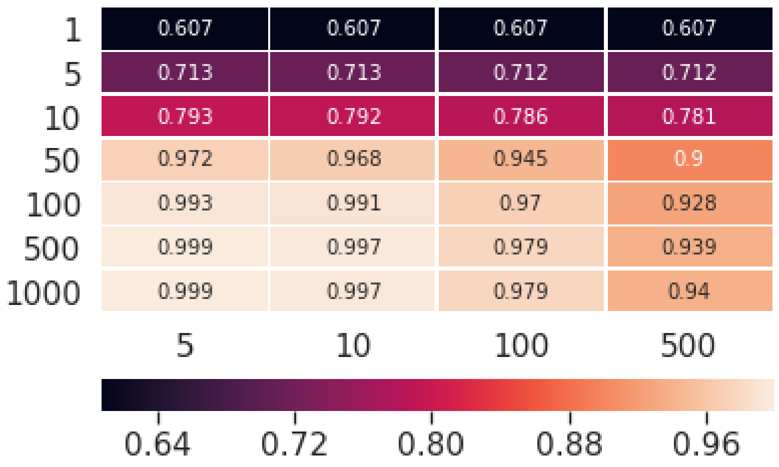
In [38]:
```
# auc on cv
cv_scores = np.array(bow_cv_auc).reshape(len(depth),len(samples))
df_cm = pd.DataFrame(cv_scores, depth, samples)
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, annot_kws={"size": 10}, fmt='.3g',linewi
dths=.5, cbar_kws={"orientation": "horizontal"})
```

Out[38]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3a956792b0>
```



In [51]:
```
# train auc
train_scores = np.array(bow_train_auc).reshape(len(depth),len(samples))
df_cm = pd.DataFrame(train_scores, depth, samples)
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, annot_kws={"size": 10}, fmt='.3g',linewi
dths=.5, cbar_kws={"orientation": "horizontal"})
```

Out[51]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3aa6387438>
```

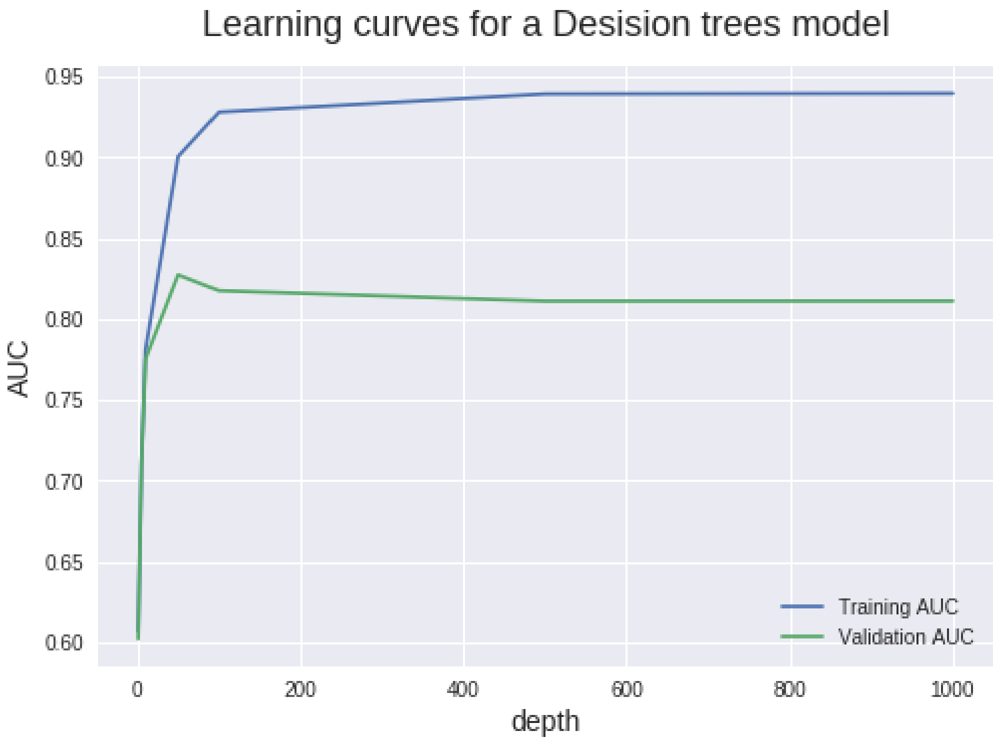


In [55]:

```
depth = [1, 5, 10, 50, 100, 500, 1000]
# https://www.dataquest.io/blog/learning-curves-machine-learning/

import matplotlib.pyplot as plt
%matplotlib inline

plt.style.use('seaborn')

plt.plot(depth,train_scores[:,-1],label = 'Training AUC')
plt.plot(depth,cv_scores[:,-1], label = 'Validation AUC')

plt.ylabel('AUC', fontsize = 14)
plt.xlabel('depth', fontsize = 14)
plt.title('Learning curves for a Desision trees model', fontsize = 18,
y = 1.03)
plt.legend()
```

Out[55]:

```
<matplotlib.legend.Legend at 0x7f3aa60a55f8>
```



In [15]:

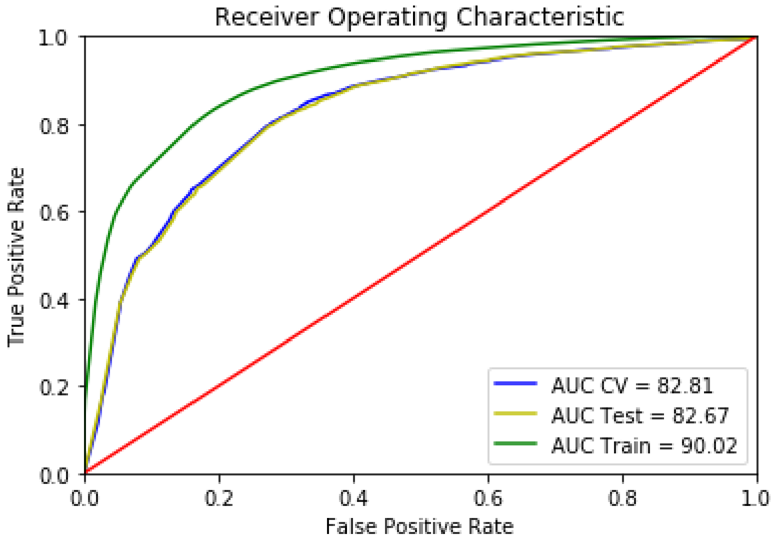
```
i = 50
j = 500

DTC = DecisionTreeClassifier(criterion='gini' , max_depth=i, min_samples_split=j)
DTC.fit(bow_train, train_y)
# train data
y_prob_train = DTC.predict_proba(bow_train)[:,1]
fprt, tprt, throsholdt = roc_curve(train_y, y_prob_train)
y_pred_train = np.where(y_prob_train > 0.5, 1, 0)
auc_roc_train = roc_auc_score(train_y , y_prob_train)
print('\nTrain AUC for min_sample = %s and depth = %s is %0.2f%%' % (str(j),str(i),(auc_roc_train * float(100))))
# CV
y_prob_cv = DTC.predict_proba(bow_cv)[:,1]
fprc, tprc, throsholdc = roc_curve(cv_y, y_prob_cv)
```

```
y_pred_cv = np.where(y_prob_cv > 0.5, 1, 0)
auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
print('\nCV AUC for min_sample = %s and depth = %s is %0.2f%%' % (str(j
),str(i),(auc_roc_cv * float(100))))
# Test
y_prob_test = DTC.predict_proba(bow_test)[:,1]
fprts, tprts, throsholdts = roc_curve(test_y, y_prob_test)
y_pred_test = np.where(y_prob_test > 0.5, 1, 0)
auc_roc_test = roc_auc_score(test_y , y_prob_test)
print('\nTest AUC for min_sample = %s and depth = %s is %0.2f%%' % (str
(j),str(i),(auc_roc_test * float(100))))
```

```
Train AUC for min_sample = 500 and depth = 50 is 90.02%

CV AUC for min_sample = 500 and depth = 50 is 82.81%

Test AUC for min_sample = 500 and depth = 50 is 82.67%
```

In [16]:

```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in
-python

import matplotlib.pyplot as plt
plt.clf()
plt.title('Receiver Operating Characteristic')
plt.plot(fprc, tprc, 'b' , label ='AUC CV = %0.2f' % (auc_roc_cv * floa
t(100)))
plt.plot(fprts, tprts, 'y' , label ='AUC Test = %0.2f' % (auc_roc_test
* float(100)))
plt.plot(fprt, tprt, 'g', label='AUC Train = %0.2f' % (auc_roc_train *
float(100)))
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

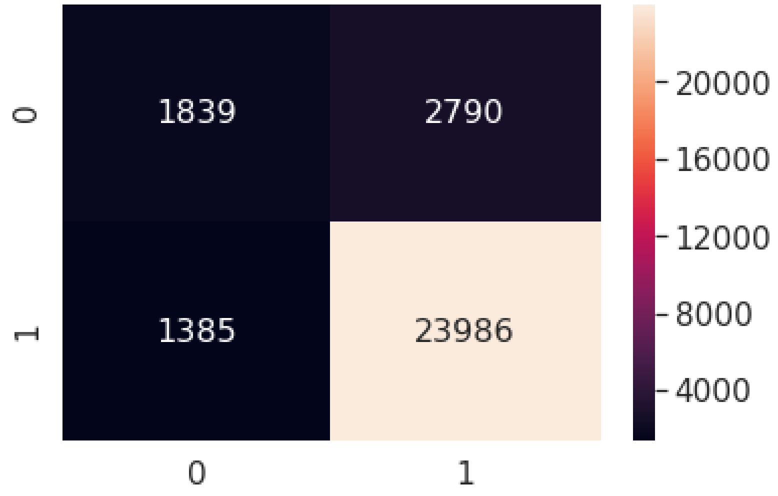


In [17]:
```
print("F1-Score on test set: %0.2f"%(f1_score(test_y, y_pred_test)))
```

```
F1-Score on test set: 0.92
```

In [18]:
```
df_cm = pd.DataFrame(confusion_matrix(test_y, y_pred_test), range(2), range(2))
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3aafe7c080>
```



[5.1.1] Top 10 important features of from SET 1

In [19]:

```
# Please write all the code with proper documentation
feture_imp = DTC.feature_importances_
feature_names = count_vect.get_feature_names()
features = dict(zip(feture_imp,feature_names))
```

In [20]:

```
features_df = pd.DataFrame.from_dict(features, orient='index')
```

In [21]:

```
sorted_features = features_df.sort_index(axis=0,ascending=False )
```

In [75]:

```
# top 10 important features
sorted_features.head(10)
```

Out[75]:

	0
0.095068	not
0.060579	great

	0
0.047388	worst
0.039606	disappointed
0.029607	horrible
0.028482	best
0.026107	waste
0.025539	delicious
0.024585	money
0.020553	love

[5.1.2] Top 10 unimportant features from SET 1

In [76]:

```
# Please write all the code with proper documentation
# top 10 unimportant features
sorted_features.tail(10)
```

Out[76]:

	0
0.000195	rise
0.000195	justify
0.000194	edible
0.000192	skinny
0.000132	flavor
0.000132	italy
0.000131	outrageous
0.000130	thirst
0.000050	results

	0
0.000000	zuke

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

In [77]:

```
# Please write all the code with proper documentation
i = 3
j = 2

DTC = DecisionTreeClassifier(criterion='gini' , max_depth=i, min_samples_split=j)
DTC.fit(bow_train, train_y)
# train data
y_prob_train = DTC.predict_proba(bow_train)[:,1]
fprt, tprt, throsholdt = roc_curve(train_y, y_prob_train)
y_pred_train = np.where(y_prob_train > 0.5, 1, 0)
auc_roc_train = roc_auc_score(train_y , y_prob_train)
print('\nTrain AUC for min_sample = %s and depth = %s is %0.2f%%' % (str(j),str(i),(auc_roc_train * float(100))))
# CV
y_prob_cv = DTC.predict_proba(bow_cv)[:,1]
fprc, tprc, throsholdc = roc_curve(cv_y, y_prob_cv)
y_pred_cv = np.where(y_prob_cv > 0.5, 1, 0)
auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
print('\nCV AUC for min_sample = %s and depth = %s is %0.2f%%' % (str(j),str(i),(auc_roc_cv * float(100))))
# Test
y_prob_test = DTC.predict_proba(bow_test)[:,1]
fprts, tprts, throsholdts = roc_curve(test_y, y_prob_test)
y_pred_test = np.where(y_prob_test > 0.5, 1, 0)
auc_roc_test = roc_auc_score(test_y , y_prob_test)
print('\nTest AUC for min_sample = %s and depth = %s is %0.2f%%' % (str(j),str(i),(auc_roc_test * float(100))))
```

```
Train AUC for min_sample = 2 and depth = 3 is 65.07%

CV AUC for min_sample = 2 and depth = 3 is 65.44%
```

```
Test AUC for min_sample = 2 and depth = 3 is 64.88%
```

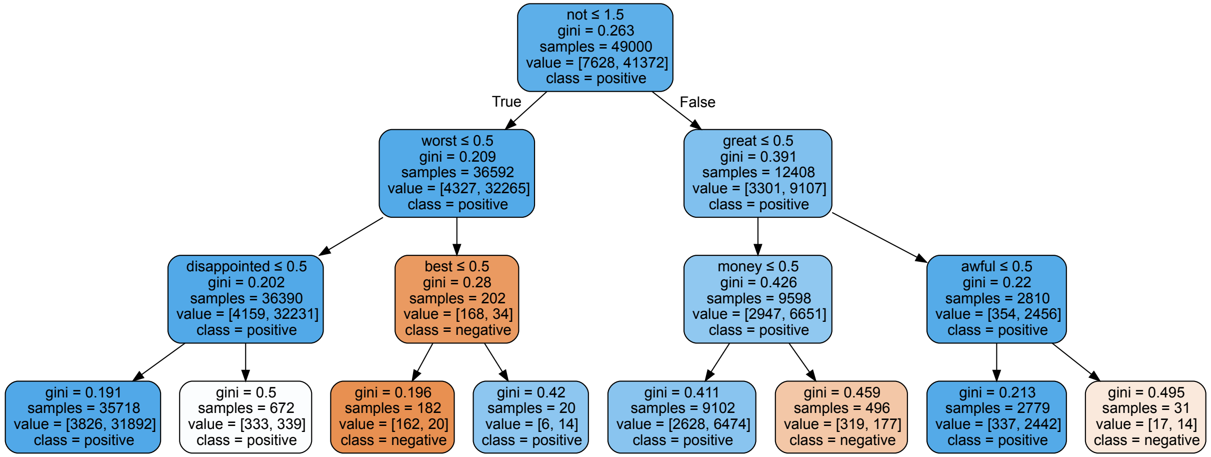
In [80]:

```
import graphviz
from sklearn import tree
```

In [87]:

```
dot_data = tree.export_graphviz(DTC , out_file=None, feature_names =cou
nt_vect.get_feature_names() , class_names=['negative', 'positive'], fil
led=True, rounded=True, special_characters=True)
graph = graphviz.Source(dot_data)
graph
```

Out[87]:



In [94]:

```
graph.view()
```

Out[94]:

```
'bow_graph.gv.pdf'
```

[5.2] Applying Decision Trees on TFIDF, SET 2

In [56]:

```
# Please write all the code with proper documentation

model = TfidfVectorizer(min_df=20, ngram_range=(1,2))
#tf_idf_matrix = model.fit_transform(train)
```

```
print("================Train Data===========")
tf_idf_train = model.fit_transform(train)
print("the type of count vectorizer ",type(tf_idf_train))
print("the shape of out text TFIDF vectorizer ",tf_idf_train.get_shape
())
print("the number of unique words including both unigrams and bigrams "
,tf_idf_train.get_shape()[1])
print("================CV Data===========")
tf_idf_cv = model.transform(cv)
print("the type of count vectorizer ",type(tf_idf_cv))
print("the shape of out text TFIDF vectorizer ",tf_idf_cv.get_shape())
print("the number of unique words including both unigrams and bigrams "
,tf_idf_cv.get_shape()[1])
print("================Test Data===========")
tf_idf_test = model.transform(test)
print("the type of count vectorizer ",type(tf_idf_test))
print("the shape of out text TFIDF vectorizer ",tf_idf_test.get_shape
())
print("the number of unique words including both unigrams and bigrams "
, tf_idf_test.get_shape()[1])

# 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_)))
```

```
================Train Data===========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (49000, 13730)
the number of unique words including both unigrams and bigrams  13730
================CV Data===========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (21000, 13730)
the number of unique words including both unigrams and bigrams  13730
================Test Data===========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (30000, 13730)
the number of unique words including both unigrams and bigrams  13730
```

In [57]:

```
depth = [1, 5, 10, 50, 100, 500, 1000]
samples = [5, 10, 100, 500]
tfidf_train_auc = []
tfidf_cv_auc = []
for i in depth:
    for j in samples:
        DTC = DecisionTreeClassifier(criterion='gini',max_depth=i, min_
samples_split=j )
        DTC.fit(tf_idf_train, train_y)
        # train data
        y_prob_train = DTC.predict_proba(tf_idf_train)[:,1]
        y_pred = np.where(y_prob_train > 0.5, 1, 0)
        auc_roc_train = roc_auc_score(train_y , y_prob_train)
        print('\nTrain AUC for min_sample = %s and max_depth = %s is %
0.2f%%' % (str(j),str(i),(auc_roc_train * float(100))))
        tfidf_train_auc.append(auc_roc_train)
        # CV
        y_prob_cv = DTC.predict_proba(tf_idf_cv)[:,1]
        y_pred = np.where(y_prob_cv > 0.5, 1, 0)
        auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
        print('\nCV AUC for min_sample = %s and max_depth = %s is %0.2
f%%' % (str(j),str(i),(auc_roc_cv * float(100))))
        tfidf_cv_auc.append(auc_roc_cv)
        print("="*50)
```

```
Train AUC for min_sample = 5 and max_depth = 1 is 60.97%

CV AUC for min_sample = 5 and max_depth = 1 is 60.23%
==================================================

Train AUC for min_sample = 10 and max_depth = 1 is 60.97%

CV AUC for min_sample = 10 and max_depth = 1 is 60.23%
==================================================

Train AUC for min_sample = 100 and max_depth = 1 is 60.97%

CV AUC for min_sample = 100 and max_depth = 1 is 60.23%
==================================================

Train AUC for min_sample = 500 and max_depth = 1 is 60.97
Train AUC for min_sample = 500 and max_depth = 1 is 60.97%

CV AUC for min_sample = 500 and max_depth = 1 is 60.23%
==================================================

Train AUC for min_sample = 5 and max_depth = 5 is 67.71%

CV AUC for min_sample = 5 and max_depth = 5 is 66.22%
==================================================

Train AUC for min_sample = 10 and max_depth = 5 is 67.71%

CV AUC for min_sample = 10 and max_depth = 5 is 66.22%
==================================================

Train AUC for min_sample = 100 and max_depth = 5 is 67.67%

CV AUC for min_sample = 100 and max_depth = 5 is 66.33%
==================================================

Train AUC for min_sample = 500 and max_depth = 5 is 67.62%

CV AUC for min_sample = 500 and max_depth = 5 is 66.32%
==================================================

Train AUC for min_sample = 5 and max_depth = 10 is 76.33%

CV AUC for min_sample = 5 and max_depth = 10 is 72.44%
==================================================

Train AUC for min_sample = 10 and max_depth = 10 is 76.22%

CV AUC for min_sample = 10 and max_depth = 10 is 72.75%
==================================================

Train AUC for min_sample = 100 and max_depth = 10 is 75.81%

CV AUC for min_sample = 100 and max_depth = 10 is 73.73%
==================================================
Train AUC for min_sample = 500 and max_depth = 10 is 75.43%

CV AUC for min_sample = 500 and max_depth = 10 is 73.89%
==================================================

Train AUC for min_sample = 5 and max_depth = 50 is 95.67%

CV AUC for min_sample = 5 and max_depth = 50 is 72.05%
==================================================

Train AUC for min_sample = 10 and max_depth = 50 is 95.30%

CV AUC for min_sample = 10 and max_depth = 50 is 74.09%
==================================================

Train AUC for min_sample = 100 and max_depth = 50 is 93.54%

CV AUC for min_sample = 100 and max_depth = 50 is 80.28%
==================================================

Train AUC for min_sample = 500 and max_depth = 50 is 91.06%

CV AUC for min_sample = 500 and max_depth = 50 is 82.81%
==================================================

Train AUC for min_sample = 5 and max_depth = 100 is 98.64%

CV AUC for min_sample = 5 and max_depth = 100 is 69.09%
==================================================

Train AUC for min_sample = 10 and max_depth = 100 is 98.52%

CV AUC for min_sample = 10 and max_depth = 100 is 70.28%
==================================================

Train AUC for min_sample = 100 and max_depth = 100 is 96.92%

CV AUC for min_sample = 100 and max_depth = 100 is 77.28%
==================================================
Train AUC for min_sample = 500 and max_depth = 100 is 94.98%

CV AUC for min_sample = 500 and max_depth = 100 is 81.04%
==================================================

Train AUC for min_sample = 5 and max_depth = 500 is 99.97%

CV AUC for min_sample = 5 and max_depth = 500 is 70.68%
==================================================

Train AUC for min_sample = 10 and max_depth = 500 is 99.88%

CV AUC for min_sample = 10 and max_depth = 500 is 71.88%
==================================================

Train AUC for min_sample = 100 and max_depth = 500 is 98.79%

CV AUC for min_sample = 100 and max_depth = 500 is 76.20%
==================================================

Train AUC for min_sample = 500 and max_depth = 500 is 97.28%

CV AUC for min_sample = 500 and max_depth = 500 is 78.31%
==================================================

Train AUC for min_sample = 5 and max_depth = 1000 is 99.97%

CV AUC for min_sample = 5 and max_depth = 1000 is 71.10%
==================================================

Train AUC for min_sample = 10 and max_depth = 1000 is 99.88%

CV AUC for min_sample = 10 and max_depth = 1000 is 72.42%
==================================================

Train AUC for min_sample = 100 and max_depth = 1000 is 98.82%

CV AUC for min_sample = 100 and max_depth = 1000 is 76.43%
```

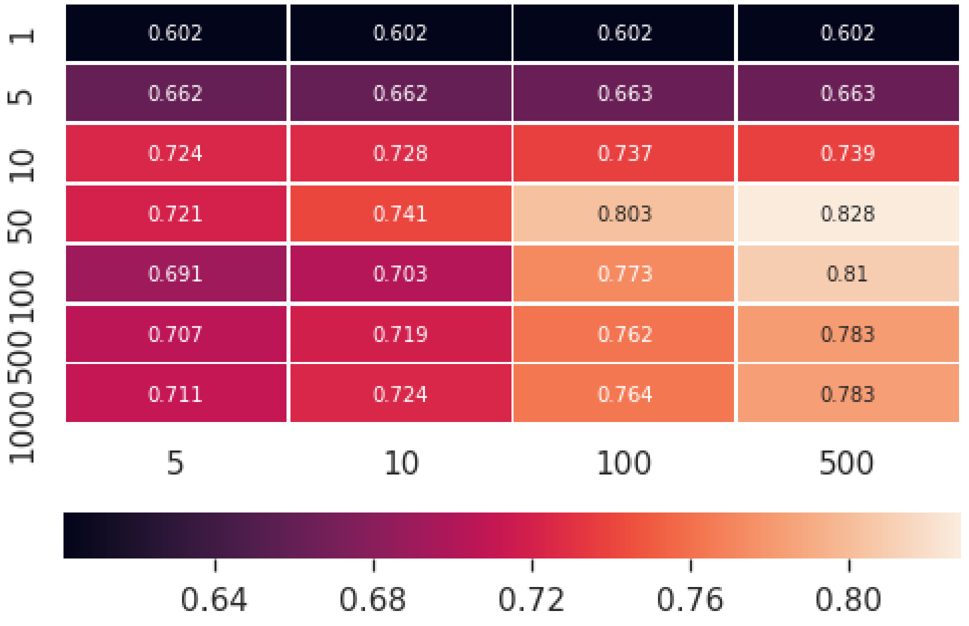
```
==================================================

Train AUC for min_sample = 500 and max_depth = 1000 is 97.23%

CV AUC for min_sample = 500 and max_depth = 1000 is 78.34%
==================================================
```

In [60]:

```
# cross validation auc
cv_scores = np.array(tfidf_cv_auc).reshape(len(depth),len(samples))
df_cm = pd.DataFrame(cv_scores, depth, samples)
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, annot_kws={"size": 10}, fmt='.3g',linewi
dths=.5, cbar_kws={"orientation": "horizontal"})
```

Out[60]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3ab4360160>
```

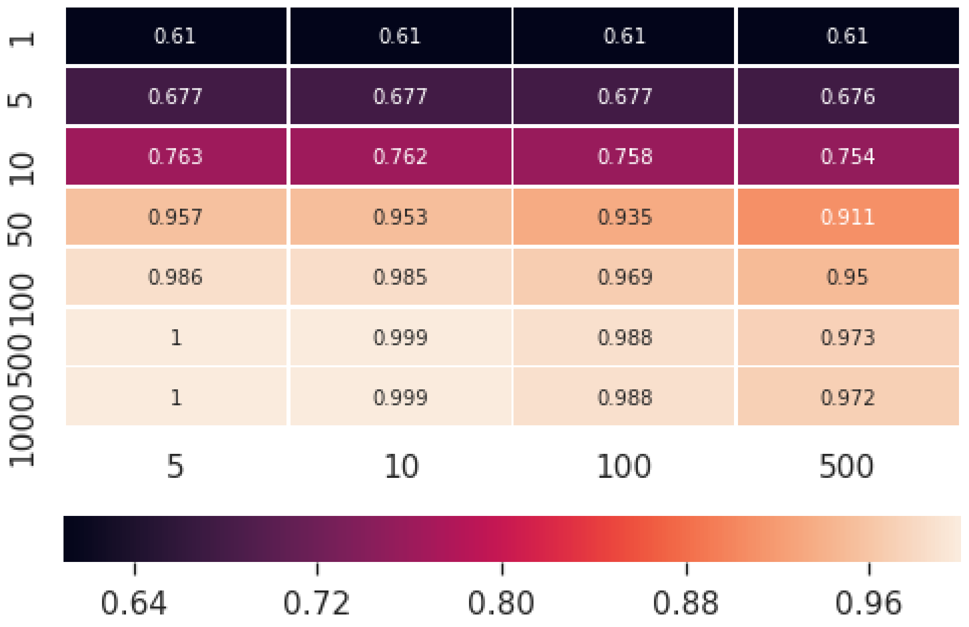


In [61]:

```
# train auc
train_scores = np.array(tfidf_train_auc).reshape(len(depth),len(samples
))
df_cm = pd.DataFrame(train_scores, depth, samples)
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, annot_kws={"size": 10}, fmt='.3g',linewi
dths=.5, cbar_kws={"orientation": "horizontal"})
```

Out[61]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3ab45b3e80>
```



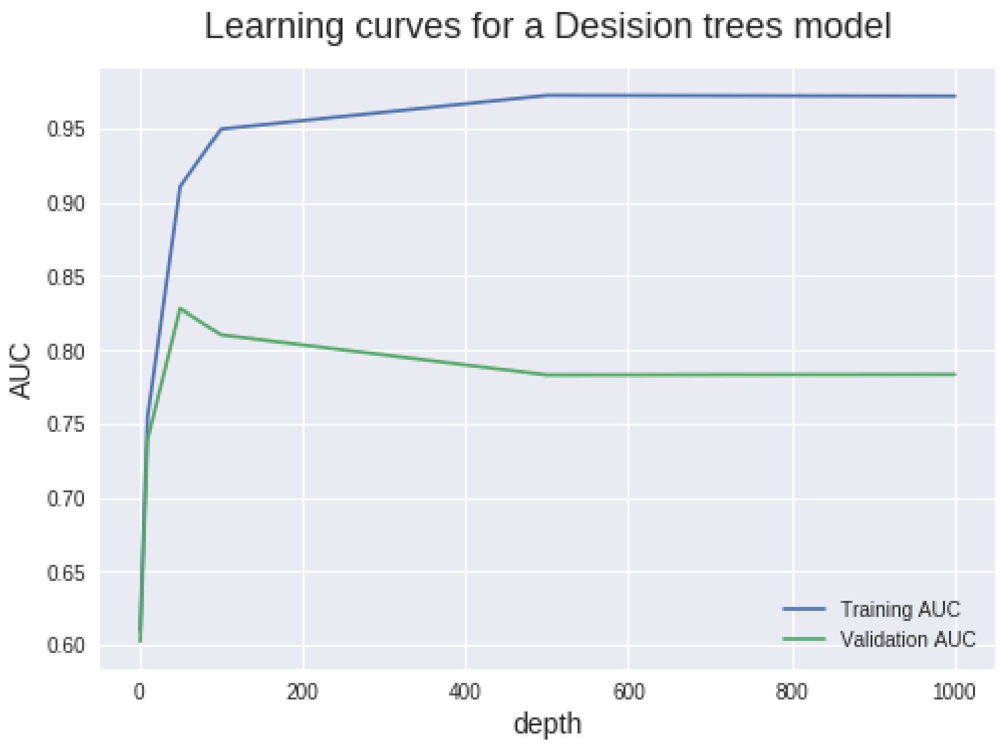
	5	10	100	500
1	0.61	0.61	0.61	0.61
5	0.677	0.677	0.677	0.676
10	0.763	0.762	0.758	0.754
50	0.957	0.953	0.935	0.911
100	0.986	0.985	0.969	0.95
500	1	0.999	0.988	0.973
1000	1	0.999	0.988	0.972

In [62]:

```
depth = [1, 5, 10, 50, 100, 500, 1000]
# https://www.dataquest.io/blog/learning-curves-machine-learning/

import matplotlib.pyplot as plt
%matplotlib inline

plt.style.use('seaborn')

plt.plot(depth,train_scores[:,-1],label = 'Training AUC')
plt.plot(depth,cv_scores[:,-1], label = 'Validation AUC')
plt.ylabel('AUC', fontsize = 14)
plt.xlabel('depth', fontsize = 14)
plt.title('Learning curves for a Desision trees model', fontsize = 18,
y = 1.03)
plt.legend()
```

Out[62]:

```
<matplotlib.legend.Legend at 0x7f3ab49074a8>
```



In [158]:

```
# Please write all the code with proper documentation
i = 50
j = 500

DTC = DecisionTreeClassifier(criterion='gini' , max_depth=i, min_sample
s_split=j)
DTC.fit(tf_idf_train, train_y)
# train data
```

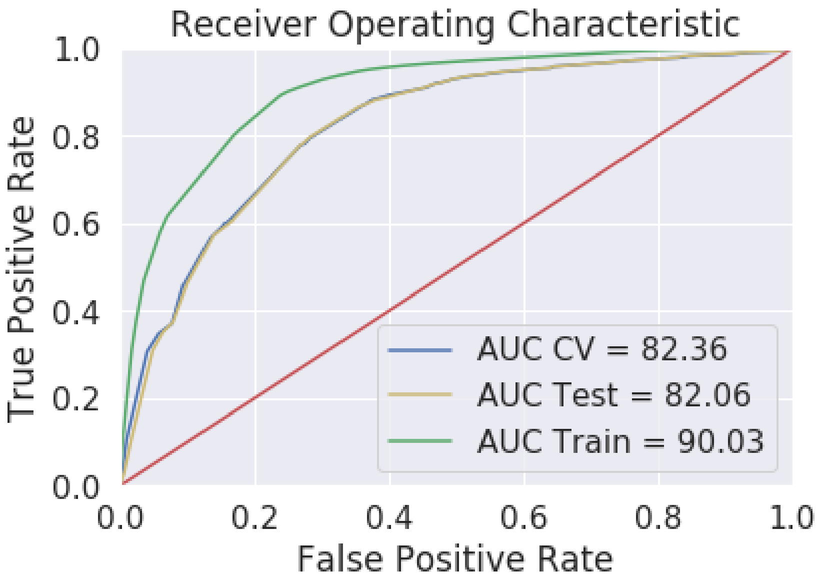
```
y_prob_train = DTC.predict_proba(tf_idf_train)[:,1]
fprt, tprt, thrsholdt = roc_curve(train_y, y_prob_train)
y_pred_train = np.where(y_prob_train > 0.5, 1, 0)
auc_roc_train = roc_auc_score(train_y , y_prob_train)
print('\nTrain AUC for min_sample = %s and depth = %s is %0.2f%%' % (st
r(j),str(i),(auc_roc_train * float(100))))
# CV
y_prob_cv = DTC.predict_proba(tf_idf_cv)[:,1]
fprc, tprc, thrsholdc = roc_curve(cv_y, y_prob_cv)
y_pred_cv = np.where(y_prob_cv > 0.5, 1, 0)
auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
print('\nCV AUC for min_sample = %s and depth = %s is %0.2f%%' % (str(j
),str(i),(auc_roc_cv * float(100))))
# Test
y_prob_test = DTC.predict_proba(tf_idf_test)[:,1]
fprts, tprts, thrsholdts = roc_curve(test_y, y_prob_test)
y_pred_test = np.where(y_prob_test > 0.5, 1, 0)
auc_roc_test = roc_auc_score(test_y , y_prob_test)
print('\nTest AUC for min_sample = %s and depth = %s is %0.2f%%' % (str
(j),str(i),(auc_roc_test * float(100))))
```

```
Train AUC for min_sample = 500 and depth = 50 is 90.03%

CV AUC for min_sample = 500 and depth = 50 is 82.36%

Test AUC for min_sample = 500 and depth = 50 is 82.06%
```

In [159]:

```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in
-python

import matplotlib.pyplot as plt
plt.clf()
plt.title('Receiver Operating Characteristic')
plt.plot(fprc, tprc, 'b' , label ='AUC CV = %0.2f' % (auc_roc_cv * floa
t(100)))
plt.plot(fprts, tprts, 'y' , label ='AUC Test = %0.2f' % (auc_roc_test
* float(100)))
plt.plot(fprt, tprt, 'g', label='AUC Train = %0.2f' % (auc_roc_train *
float(100)))
```

```
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [161]:

```
print("F1-Score on test set: %0.2f"%(f1_score(test_y, y_pred_test)))
```

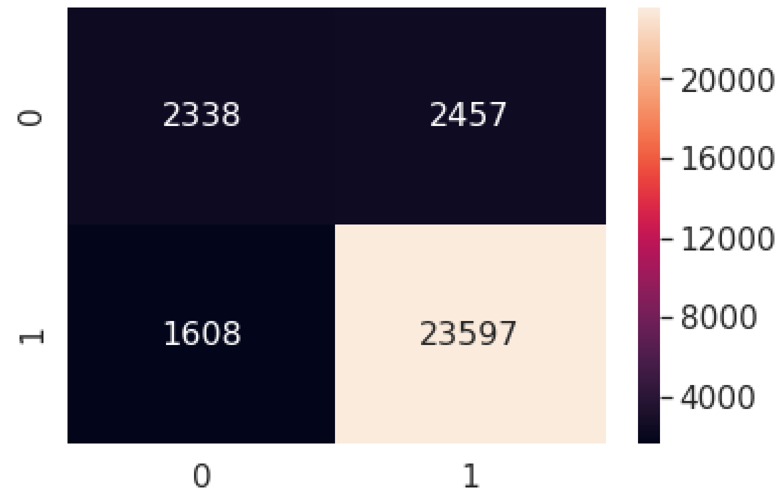
```
F1-Score on test set: 0.92
```

In [162]:

```
df_cm = pd.DataFrame(confusion_matrix(test_y, y_pred_test), range(2), r
ange(2))
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

Out[162]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fce87c36240>
```



[5.2.1] Top 10 important features from SET 2

In [164]:
```
# Please write all the code with proper documentation
feture_imp = DTC.feature_importances_
feature_names = model.get_feature_names()
features = dict(zip(feture_imp,feature_names))
```

In [165]:
```
features_df = pd.DataFrame.from_dict(features, orient='index')
```

In [166]:
```
sorted_features = features_df.sort_index(axis=0,ascending=False )
```

In [167]:
```
# top 10 important features
sorted_features.head(10)
```

Out[167]:

	0
0.089370	not
0.045175	great

	0
0.041375	disappointed
0.038242	worst
0.034612	not buy
0.030820	waste money
0.025641	money
0.025234	horrible
0.019566	refund
0.018403	awful

[5.2.2] Top 10 unimportant features from SET 2

In [168]:

```
# Please write all the code with proper documentation
# top 10 unimportant features
sorted_features.tail(10)
```

Out[168]:

	0
0.000346	big
0.000345	one first
0.000345	taste food
0.000345	expensive not
0.000344	quantity
0.000343	taste flavor
0.000343	say not
0.000342	dehydrated
0.000341	requirements

	0
0.000000	zuke

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

In [169]:

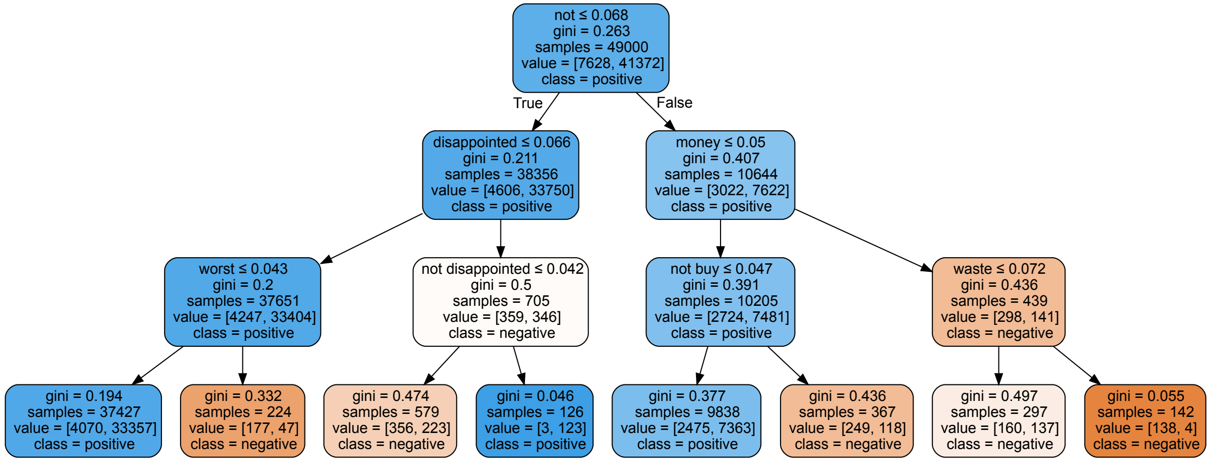
```
# Please write all the code with proper documentation
i = 3
j = 2

DTC = DecisionTreeClassifier(criterion='gini' , max_depth=i, min_sample
s_split=j)
DTC.fit(tf_idf_train, train_y)
# train data
y_prob_train = DTC.predict_proba(tf_idf_train)[:,1]
fprt, tprt, throsholdt = roc_curve(train_y, y_prob_train)
y_pred_train = np.where(y_prob_train > 0.5, 1, 0)
auc_roc_train = roc_auc_score(train_y , y_prob_train)
print('\nTrain AUC for min_sample = %s and depth = %s is %0.2f%%' % (st
r(j),str(i),(auc_roc_train * float(100))))
# CV
y_prob_cv = DTC.predict_proba(tf_idf_cv)[:,1]
fprc, tprc, throsholdc = roc_curve(cv_y, y_prob_cv)
y_pred_cv = np.where(y_prob_cv > 0.5, 1, 0)
auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
print('\nCV AUC for min_sample = %s and depth = %s is %0.2f%%' % (str(j
),str(i),(auc_roc_cv * float(100))))
# Test
y_prob_test = DTC.predict_proba(tf_idf_test)[:,1]
fprts, tprts, throsholdts = roc_curve(test_y, y_prob_test)
y_pred_test = np.where(y_prob_test > 0.5, 1, 0)
auc_roc_test = roc_auc_score(test_y , y_prob_test)
print('\nTest AUC for min_sample = %s and depth = %s is %0.2f%%' % (str
(j),str(i),(auc_roc_test * float(100))))
```

```
Train AUC for min_sample = 2 and depth = 3 is 64.91%

CV AUC for min_sample = 2 and depth = 3 is 65.01%
```

```
Test AUC for min_sample = 2 and depth = 3 is 65.08%
```

In [171]:
```
dot_data = tree.export_graphviz(DTC , out_file=None, feature_names =mod
el.get_feature_names() , class_names=['negative', 'positive'], filled=T
rue, rounded=True, special_characters=True)
graph = graphviz.Source(dot_data)
graph
```

Out[171]:



In [172]:
```
graph.view()
```

Out[172]:
```
'Source.gv.pdf'
```

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

In [63]:
```
# Train your own Word2Vec model using your own text corpus
######## Train Set #########
i=0
list_of_train_sentance=[]
for sentance in train:
    list_of_train_sentance.append(sentance.split())
######## CV Set ###########
```

```
i=0
list_of_cv_sentance=[]
for sentance in cv:
    list_of_cv_sentance.append(sentance.split())
######## Test Set #########
i=0
list_of_test_sentance=[]
for sentance in test:
    list_of_test_sentance.append(sentance.split())
print("Length of Train = ", len(list_of_train_sentance))
print("Length of CV = ", len(list_of_cv_sentance))
print("Length of Test = ", len(list_of_test_sentance))
```

```
Length of Train =  49000
Length of CV =  21000
Length of Test =  30000
```

In [64]:

```
w2v_model=Word2Vec(list_of_train_sentance,min_count=15,size=100, worker
s=4)
print(w2v_model.wv.most_similar('great'))
print('='*50)
print(w2v_model.wv.most_similar('worst'))
```

```
[('awesome', 0.7750544548034668), ('excellent', 0.7643860578536987),
('fantastic', 0.762516975402832), ('good', 0.7511047720909119), ('wonde
rful', 0.7507315874099731), ('fabulous', 0.70582115650177), ('terrifi
c', 0.6960405707359314), ('amazing', 0.6841104626655579), ('perfect',
0.6794580817222595), ('decent', 0.5999100208282471)]
==================================================
[('greatest', 0.8142168521881104), ('best', 0.7570872902870178), ('tast
iest', 0.728318452835083), ('smoothest', 0.681471586227417), ('disgusti
ng', 0.5995469093322754), ('honestly', 0.5933093428611755), ('closest',
0.5928040742874146), ('superior', 0.5815048217773438), ('horrible', 0.5
754640698432922), ('freshest', 0.5535082817077637)]
```

In [65]:

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

```
number of words that occured minimum 5 times  7519
sample words  ['fresh', 'nuts', 'excellent', 'service', 'shipped', 'fas
t', 'made', 'hazelnut', 'cake', 'hit', 'party', 'definitely', 'use', 'f
ruits', 'vegetables', 'skin', 'edible', 'negative', 'washed', 'produc
e', 'much', 'shorter', 'shelf', 'life', 'hate', 'bananas', 'not', 'bi
g', 'fan', 'banana', 'nut', 'bread', 'either', 'expect', 'like', 'wron
g', 'loved', 'good', 'cold', 'warmed', 'seconds', 'prefer', 'warm', 'fa
vorite', 'part', 'fact', 'microwaved', 'plastic', 'bag', 'came']
```

In [66]:

```
######## Train data ########
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_train = []; # the avg-w2v for each sentence/review is stor
ed in this list
for sent in tqdm(list_of_train_sentance): # for each review/sentence
    sent_vec = np.zeros(100) # as word vectors are of zero length 50, y
ou 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/re
view
    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_train.append(sent_vec)
print(len(sent_vectors_train))
print(len(sent_vectors_train[0]))
```

```
100%|██████████| 49000/49000 [02:08<00:00, 380.99it/s]
```

```
49000
100
```

In [67]:

```
######## CV data ########
# average Word2Vec
# compute average word2vec for each review.
```

```
sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_cv_sentance): # for each review/sentence
    sent_vec = np.zeros(100) # 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_cv.append(sent_vec)
print(len(sent_vectors_cv))
print(len(sent_vectors_cv[0]))
```

```
100%|██████████| 21000/21000 [00:56<00:00, 374.22it/s]
```

```
21000
100
```

In [68]:

```
######## Test data ########
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_test_sentance): # for each review/sentence
    sent_vec = np.zeros(100) # 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_test.append(sent_vec)
print(len(sent_vectors_test))
print(len(sent_vectors_test[0]))
```

```
100%|██████████| 30000/30000 [01:26<00:00, 347.61it/s]
```

```
30000
100
```

In [69]:

```
# save the datasets as numpy array
w2v_train = np.array(sent_vectors_train)
w2v_cv = np.array(sent_vectors_cv)
w2v_test = np.array(sent_vectors_test)
```

In [70]:

```
depth = [1, 5, 10, 50, 100, 500, 1000]
samples = [5, 10, 100, 500]
w2v_train_auc = []
w2v_cv_auc = []
for i in depth:
    for j in samples:
        DTC = DecisionTreeClassifier(criterion='gini',max_depth=i, min_
samples_split=j )
        DTC.fit(w2v_train, train_y)
        # train data
        y_prob_train = DTC.predict_proba(w2v_train)[:,1]
        y_pred = np.where(y_prob_train > 0.5, 1, 0)
        auc_roc_train = roc_auc_score(train_y , y_prob_train)
        print('\nTrain AUC for min_sample = %s and max_depth = %s is %
0.2f%%' % (str(j),str(i),(auc_roc_train * float(100))))
        w2v_train_auc.append(auc_roc_train)
        # CV
        y_prob_cv = DTC.predict_proba(w2v_cv)[:,1]
        y_pred = np.where(y_prob_cv > 0.5, 1, 0)
        auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
        print('\nCV AUC for min_sample = %s and max_depth = %s is %0.2
f%%' % (str(j),str(i),(auc_roc_cv * float(100))))
```

```
        w2v_cv_auc.append(auc_roc_cv)
        print("="*50)
```

```
Train AUC for min_sample = 5 and max_depth = 1 is 65.07%

CV AUC for min_sample = 5 and max_depth = 1 is 64.50%
==================================================

Train AUC for min_sample = 10 and max_depth = 1 is 65.07%

CV AUC for min_sample = 10 and max_depth = 1 is 64.50%
==================================================

Train AUC for min_sample = 100 and max_depth = 1 is 65.07%

CV AUC for min_sample = 100 and max_depth = 1 is 64.50%
==================================================

Train AUC for min_sample = 500 and max_depth = 1 is 65.07%

CV AUC for min_sample = 500 and max_depth = 1 is 64.50%
==================================================

Train AUC for min_sample = 5 and max_depth = 5 is 82.38%

CV AUC for min_sample = 5 and max_depth = 5 is 80.20%
==================================================

Train AUC for min_sample = 10 and max_depth = 5 is 82.38%

CV AUC for min_sample = 10 and max_depth = 5 is 80.20%
==================================================

Train AUC for min_sample = 100 and max_depth = 5 is 82.38%

CV AUC for min_sample = 100 and max_depth = 5 is 80.20%
==================================================

Train AUC for min_sample = 500 and max_depth = 5 is 82.38%
```

```
CV AUC for min_sample = 500 and max_depth = 5 is 80.20%
==================================================

Train AUC for min_sample = 5 and max_depth = 10 is 91.74%

CV AUC for min_sample = 5 and max_depth = 10 is 78.43%
==================================================

Train AUC for min_sample = 10 and max_depth = 10 is 91.61%

CV AUC for min_sample = 10 and max_depth = 10 is 79.12%
==================================================

Train AUC for min_sample = 100 and max_depth = 10 is 89.95%

CV AUC for min_sample = 100 and max_depth = 10 is 81.56%
==================================================

Train AUC for min_sample = 500 and max_depth = 10 is 86.87%

CV AUC for min_sample = 500 and max_depth = 10 is 81.69%
==================================================

Train AUC for min_sample = 5 and max_depth = 50 is 99.97%

CV AUC for min_sample = 5 and max_depth = 50 is 67.36%
==================================================

Train AUC for min_sample = 10 and max_depth = 50 is 99.73%

CV AUC for min_sample = 10 and max_depth = 50 is 69.27%
==================================================

Train AUC for min_sample = 100 and max_depth = 50 is 94.58%

CV AUC for min_sample = 100 and max_depth = 50 is 78.89%
==================================================

Train AUC for min_sample = 500 and max_depth = 50 is 88.23%
```

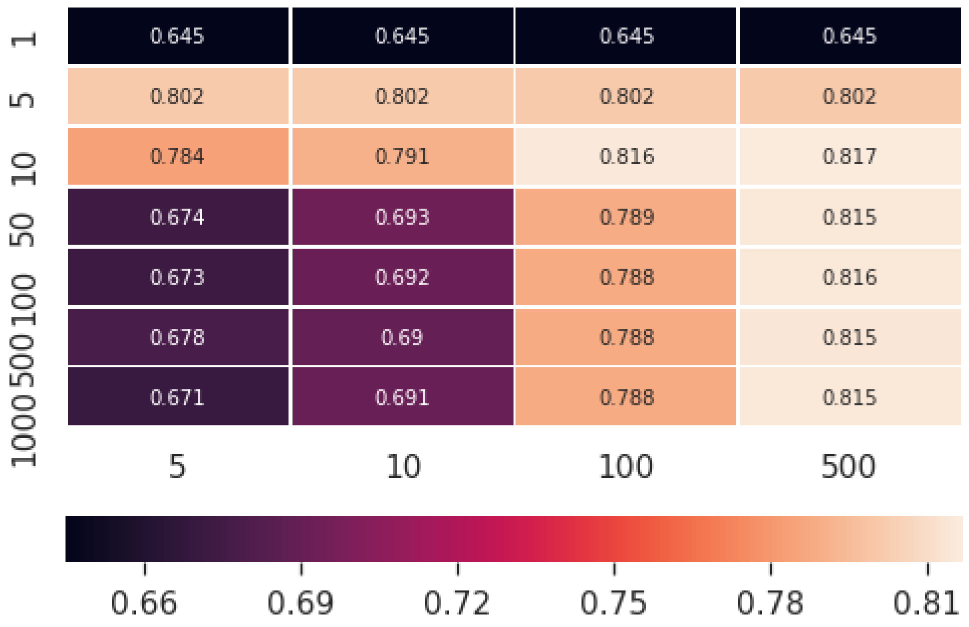
```
CV AUC for min_sample = 500 and max_depth = 50 is 81.53%
==================================================

Train AUC for min_sample = 5 and max_depth = 100 is 99.97%

CV AUC for min_sample = 5 and max_depth = 100 is 67.30%
==================================================

Train AUC for min_sample = 10 and max_depth = 100 is 99.73%

CV AUC for min_sample = 10 and max_depth = 100 is 69.19%
==================================================

Train AUC for min_sample = 100 and max_depth = 100 is 94.60%

CV AUC for min_sample = 100 and max_depth = 100 is 78.81%
==================================================

Train AUC for min_sample = 500 and max_depth = 100 is 88.24%

CV AUC for min_sample = 500 and max_depth = 100 is 81.56%
==================================================

Train AUC for min_sample = 5 and max_depth = 500 is 99.97%

CV AUC for min_sample = 5 and max_depth = 500 is 67.81%
==================================================

Train AUC for min_sample = 10 and max_depth = 500 is 99.73%

CV AUC for min_sample = 10 and max_depth = 500 is 68.99%
==================================================

Train AUC for min_sample = 100 and max_depth = 500 is 94.60%

CV AUC for min_sample = 100 and max_depth = 500 is 78.84%
==================================================
Train AUC for min_sample = 500 and max_depth = 500 is 88.25%

CV AUC for min_sample = 500 and max_depth = 500 is 81.48%
==================================================

Train AUC for min_sample = 5 and max_depth = 1000 is 99.97%

CV AUC for min_sample = 5 and max_depth = 1000 is 67.11%
==================================================

Train AUC for min_sample = 10 and max_depth = 1000 is 99.73%

CV AUC for min_sample = 10 and max_depth = 1000 is 69.11%
==================================================

Train AUC for min_sample = 100 and max_depth = 1000 is 94.61%

CV AUC for min_sample = 100 and max_depth = 1000 is 78.84%
==================================================

Train AUC for min_sample = 500 and max_depth = 1000 is 88.24%

CV AUC for min_sample = 500 and max_depth = 1000 is 81.51%
==================================================
```

In [77]:

```
# cv auc
cv_scores = np.array(w2v_cv_auc).reshape(len(depth),len(samples))
df_cm = pd.DataFrame(cv_scores, depth, samples)
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, annot_kws={"size": 10}, fmt='.3g',linewi
dths=.5 , cbar_kws={"orientation": "horizontal"})
```

Out[77]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3a7d9acb38>
```

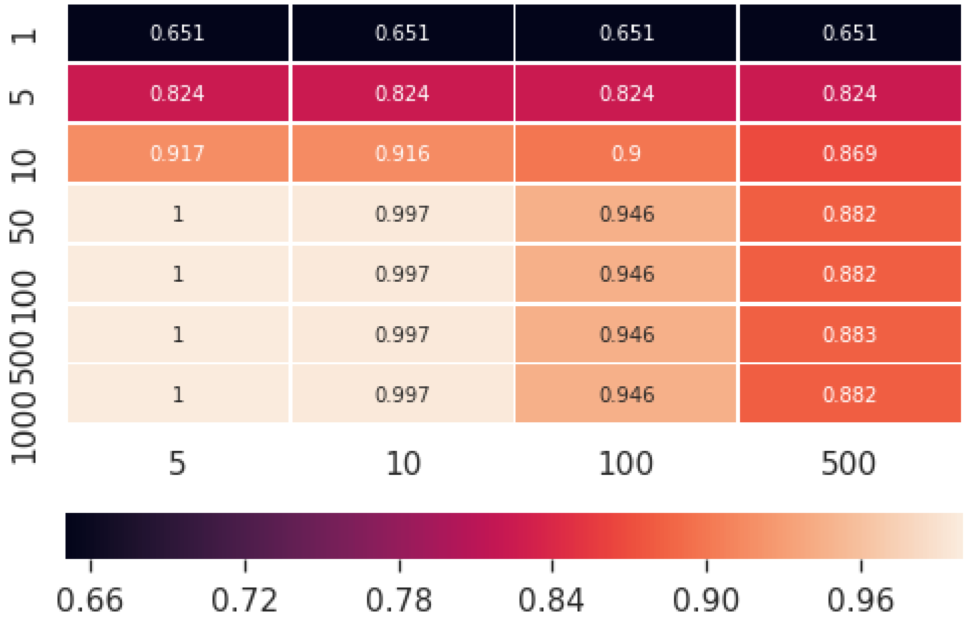


In [80]:

```
# train auc
train_scores = np.array(w2v_train_auc).reshape(len(depth),len(samples))
df_cm = pd.DataFrame(train_scores, depth, samples)
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, annot_kws={"size": 10}, fmt='.3g',linewi
dths=.5 , cbar_kws={"orientation": "horizontal"})
```

Out[80]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3a7d8215f8>
```

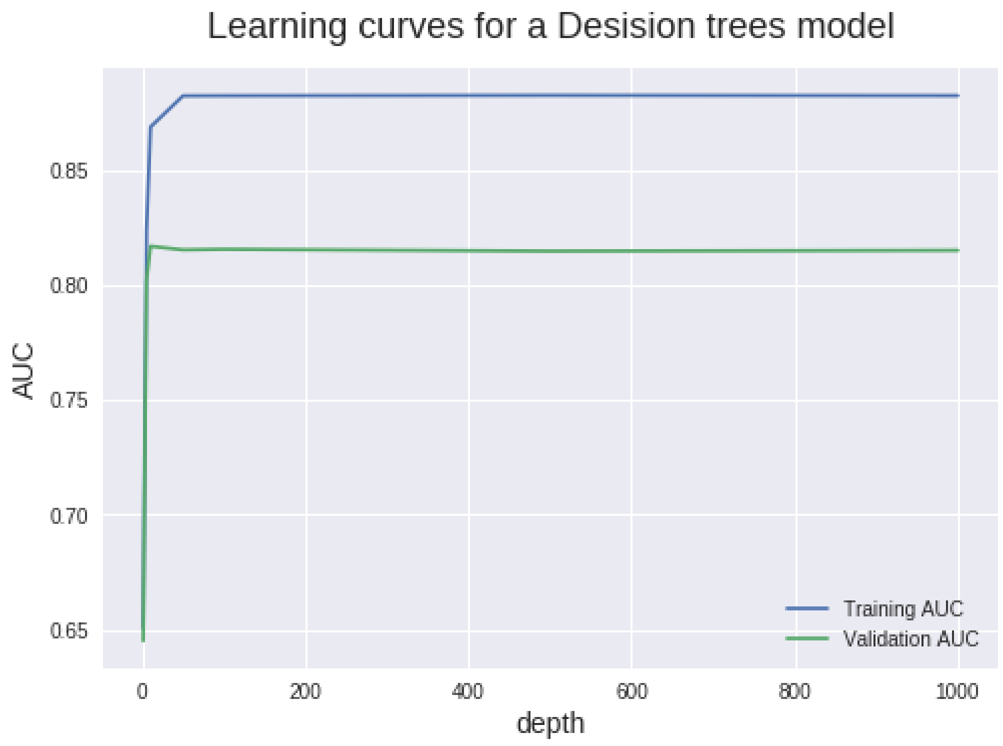


In [81]:

```
depth = [1, 5, 10, 50, 100, 500, 1000]
# https://www.dataquest.io/blog/learning-curves-machine-learning/

import matplotlib.pyplot as plt
%matplotlib inline

plt.style.use('seaborn')

plt.plot(depth,train_scores[:,-1],label = 'Training AUC')
plt.plot(depth,cv_scores[:,-1], label = 'Validation AUC')

plt.ylabel('AUC', fontsize = 14)
plt.xlabel('depth', fontsize = 14)
plt.title('Learning curves for a Desision trees model', fontsize = 18,
y = 1.03)
plt.legend()
```

Out[81]:

```
<matplotlib.legend.Legend at 0x7f3a7d70e978>
```



In [183]:

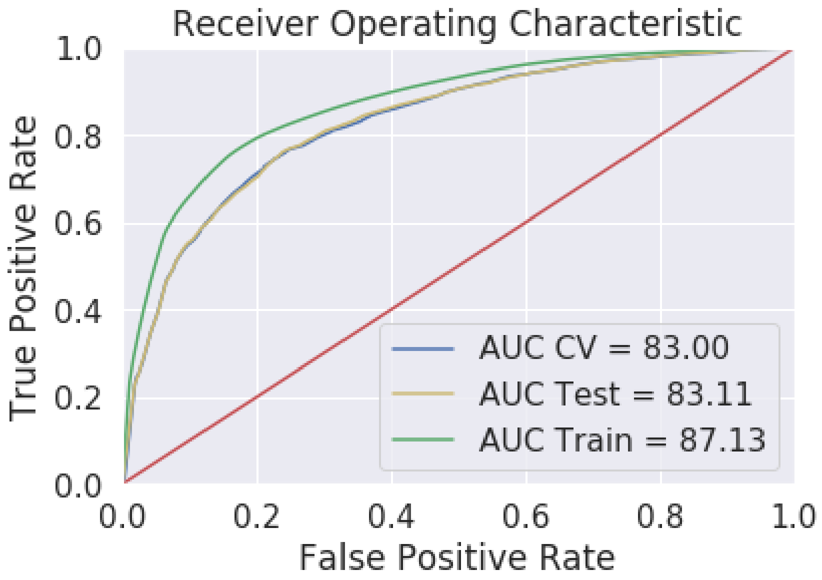
```
# Please write all the code with proper documentation
i = 10
j = 500

DTC = DecisionTreeClassifier(criterion='gini' , max_depth=i, min_sample
s_split=j)
DTC.fit(w2v_train, train_y)
# train data
y_prob_train = DTC.predict_proba(w2v_train)[:,1]
fprt, tprt, throsholdt = roc_curve(train_y, y_prob_train)
y_pred_train = np.where(y_prob_train > 0.5, 1, 0)
auc_roc_train = roc_auc_score(train_y , y_prob_train)
print('\nTrain AUC for min_sample = %s and depth = %s is %0.2f%%' % (st
r(j),str(i),(auc_roc_train * float(100))))
# CV
y_prob_cv = DTC.predict_proba(w2v_cv)[:,1]
```

```
fprc, tprc, thrsholdc = roc_curve(cv_y, y_prob_cv)
y_pred_cv = np.where(y_prob_cv > 0.5, 1, 0)
auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
print('\nCV AUC for min_sample = %s and depth = %s is %0.2f%%' % (str(j
),str(i),(auc_roc_cv * float(100))))
# Test
y_prob_test = DTC.predict_proba(w2v_test)[:,1]
fprts, tprts, thrsholdts = roc_curve(test_y, y_prob_test)
y_pred_test = np.where(y_prob_test > 0.5, 1, 0)
auc_roc_test = roc_auc_score(test_y , y_prob_test)
print('\nTest AUC for min_sample = %s and depth = %s is %0.2f%%' % (str
(j),str(i),(auc_roc_test * float(100))))
```

```
Train AUC for min_sample = 500 and depth = 10 is 87.13%

CV AUC for min_sample = 500 and depth = 10 is 83.00%

Test AUC for min_sample = 500 and depth = 10 is 83.11%
```

In [184]:

```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in
-python

import matplotlib.pyplot as plt
plt.clf()
plt.title('Receiver Operating Characteristic')
plt.plot(fprc, tprc, 'b' , label ='AUC CV = %0.2f' % (auc_roc_cv * floa
t(100)))
plt.plot(fprts, tprts, 'y' , label ='AUC Test = %0.2f' % (auc_roc_test
* float(100)))
plt.plot(fprt, tprt, 'g', label='AUC Train = %0.2f' % (auc_roc_train *
float(100)))
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

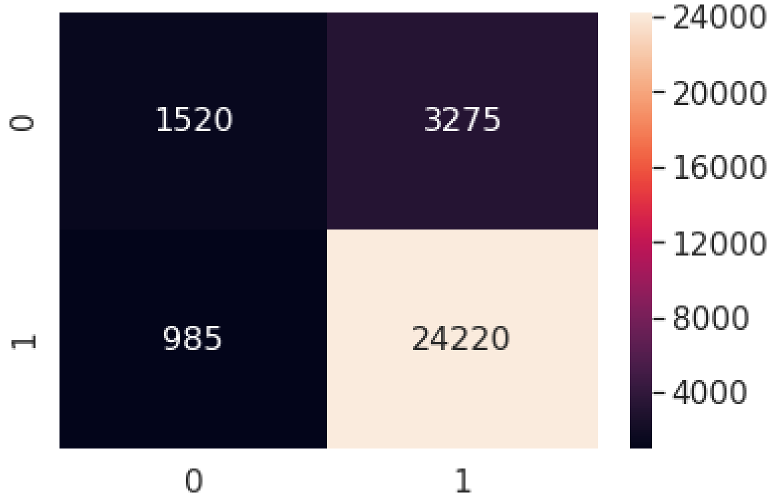


```
In [185]: print("F1-Score on test set: %0.2f"%(f1_score(test_y, y_pred_test)))
```

```
F1-Score on test set: 0.92
```

```
In [186]: df_cm = pd.DataFrame(confusion_matrix(test_y, y_pred_test), range(2), r
          ange(2))
          sns.set(font_scale=1.4)
          sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
Out[186]: <matplotlib.axes._subplots.AxesSubplot at 0x7fce6c0f2160>
```



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

In [187]:

```
# Please write all the code with proper documentation

model = TfidfVectorizer()
#tf_idf_matrix = model.fit_transform(train)

print("================Train Data===========")
final_tf_idf_train = model.fit_transform(train)
print("the type of count vectorizer ",type(final_tf_idf_train))
print("the shape of out text TFIDF vectorizer ",final_tf_idf_train.get_
shape())
print("the number of unique words including both unigrams and bigrams "
, final_tf_idf_train.get_shape()[1])
print("================CV Data===========")
final_tf_idf_cv = model.transform(cv)
print("the type of count vectorizer ",type(final_tf_idf_cv))
print("the shape of out text TFIDF vectorizer ",final_tf_idf_cv.get_sha
pe())
print("the number of unique words including both unigrams and bigrams "
, final_tf_idf_cv.get_shape()[1])
```

```
print("================Test Data===========")
final_tf_idf_test = model.transform(test)
print("the type of count vectorizer ",type(final_tf_idf_test))
print("the shape of out text TFIDF vectorizer ",final_tf_idf_test.get_s
hape())
print("the number of unique words including both unigrams and bigrams "
, final_tf_idf_test.get_shape()[1])

# 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_)))
```

```
================Train Data===========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (49000, 42866)
the number of unique words including both unigrams and bigrams  42866
================CV Data===========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (21000, 42866)
the number of unique words including both unigrams and bigrams  42866
================Test Data===========
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (30000, 42866)
the number of unique words including both unigrams and bigrams  42866
```

In [188]:

```
######## Train #######
# 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 ce
ll_val = tfidf

train_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review
 is stored in this list
row=0;
for sent in tqdm(list_of_train_sentance): # for each review/sentence
    sent_vec = np.zeros(100) # 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
    train_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 49000/49000 [50:42<00:00, 20.65it/s]
```

In [189]:

```
######## CV #######
# 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 ce
ll_val = tfidf

cv_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is
 stored in this list
row=0;
for sent in tqdm(list_of_cv_sentance): # for each review/sentence
    sent_vec = np.zeros(100) # 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
    cv_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 21000/21000 [15:39<00:00, 22.34it/s]
```

In [190]:

```
######## Train #######
# 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 ce
ll_val = tfidf

test_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review
 is stored in this list
row=0;
for sent in tqdm(list_of_test_sentance): # for each review/sentence
    sent_vec = np.zeros(100) # 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
    test_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 30000/30000 [24:17<00:00, 20.58it/s]
```

In [191]:

```
# save the datasets as numpy array
tfidf_w2v_train = np.array(train_tfidf_sent_vectors)
```

```
tfidf_w2v_cv = np.array(cv_tfidf_sent_vectors)
tfidf_w2v_test = np.array(test_tfidf_sent_vectors)
```

In [193]:
```
depth = [1, 5, 10, 50, 100, 500, 1000]
samples = [5, 10, 100, 500]
tfidf_w2v_train_auc = []
tfidf_w2v_cv_auc = []
for i in depth:
    for j in samples:
        DTC = DecisionTreeClassifier(criterion='gini',max_depth=i, min_
samples_split=j )
        DTC.fit(tfidf_w2v_train, train_y)
        # train data
        y_prob_train = DTC.predict_proba(tfidf_w2v_train)[:,1]
        y_pred = np.where(y_prob_train > 0.5, 1, 0)
        auc_roc_train = roc_auc_score(train_y , y_prob_train)
        print('\nTrain AUC for min_sample = %s and max_depth = %s is %
0.2f%%' % (str(j),str(i),(auc_roc_train * float(100))))
        tfidf_w2v_train_auc.append(auc_roc_train)
        # CV
        y_prob_cv = DTC.predict_proba(tfidf_w2v_cv)[:,1]
        y_pred = np.where(y_prob_cv > 0.5, 1, 0)
        auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
        print('\nCV AUC for min_sample = %s and max_depth = %s is %0.2
f%%' % (str(j),str(i),(auc_roc_cv * float(100))))
        tfidf_w2v_cv_auc.append(auc_roc_cv)
        print("="*50)
```

```
Train AUC for min_sample = 5 and max_depth = 1 is 66.20%

CV AUC for min_sample = 5 and max_depth = 1 is 66.25%
==================================================

Train AUC for min_sample = 10 and max_depth = 1 is 66.20%

CV AUC for min_sample = 10 and max_depth = 1 is 66.25%
==================================================

Train AUC for min_sample = 100 and max_depth = 1 is 66.20%
```

```
CV AUC for min_sample = 100 and max_depth = 1 is 66.25%
==================================================

Train AUC for min_sample = 500 and max_depth = 1 is 66.20%

CV AUC for min_sample = 500 and max_depth = 1 is 66.25%
==================================================

Train AUC for min_sample = 5 and max_depth = 5 is 80.38%

CV AUC for min_sample = 5 and max_depth = 5 is 79.14%
==================================================

Train AUC for min_sample = 10 and max_depth = 5 is 80.38%

CV AUC for min_sample = 10 and max_depth = 5 is 79.14%
==================================================

Train AUC for min_sample = 100 and max_depth = 5 is 80.38%

CV AUC for min_sample = 100 and max_depth = 5 is 79.14%
==================================================

Train AUC for min_sample = 500 and max_depth = 5 is 80.38%

CV AUC for min_sample = 500 and max_depth = 5 is 79.14%
==================================================

Train AUC for min_sample = 5 and max_depth = 10 is 89.89%

CV AUC for min_sample = 5 and max_depth = 10 is 77.01%
==================================================

Train AUC for min_sample = 10 and max_depth = 10 is 89.73%

CV AUC for min_sample = 10 and max_depth = 10 is 77.45%
==================================================
Train AUC for min_sample = 100 and max_depth = 10 is 87.93%

CV AUC for min_sample = 100 and max_depth = 10 is 79.73%
==================================================

Train AUC for min_sample = 500 and max_depth = 10 is 84.57%

CV AUC for min_sample = 500 and max_depth = 10 is 80.43%
==================================================

Train AUC for min_sample = 5 and max_depth = 50 is 99.96%

CV AUC for min_sample = 5 and max_depth = 50 is 65.32%
==================================================

Train AUC for min_sample = 10 and max_depth = 50 is 99.69%

CV AUC for min_sample = 10 and max_depth = 50 is 67.01%
==================================================

Train AUC for min_sample = 100 and max_depth = 50 is 93.85%

CV AUC for min_sample = 100 and max_depth = 50 is 76.35%
==================================================

Train AUC for min_sample = 500 and max_depth = 50 is 86.34%

CV AUC for min_sample = 500 and max_depth = 50 is 79.89%
==================================================

Train AUC for min_sample = 5 and max_depth = 100 is 99.96%

CV AUC for min_sample = 5 and max_depth = 100 is 65.52%
==================================================

Train AUC for min_sample = 10 and max_depth = 100 is 99.70%

CV AUC for min_sample = 10 and max_depth = 100 is 67.20%
==================================================
Train AUC for min_sample = 100 and max_depth = 100 is 93.76%

CV AUC for min_sample = 100 and max_depth = 100 is 76.50%
==================================================

Train AUC for min_sample = 500 and max_depth = 100 is 86.33%

CV AUC for min_sample = 500 and max_depth = 100 is 79.82%
==================================================

Train AUC for min_sample = 5 and max_depth = 500 is 99.96%

CV AUC for min_sample = 5 and max_depth = 500 is 65.88%
==================================================

Train AUC for min_sample = 10 and max_depth = 500 is 99.69%

CV AUC for min_sample = 10 and max_depth = 500 is 66.95%
==================================================

Train AUC for min_sample = 100 and max_depth = 500 is 93.78%

CV AUC for min_sample = 100 and max_depth = 500 is 76.58%
==================================================

Train AUC for min_sample = 500 and max_depth = 500 is 86.35%

CV AUC for min_sample = 500 and max_depth = 500 is 79.86%
==================================================

Train AUC for min_sample = 5 and max_depth = 1000 is 99.96%

CV AUC for min_sample = 5 and max_depth = 1000 is 65.64%
==================================================

Train AUC for min_sample = 10 and max_depth = 1000 is 99.70%

CV AUC for min_sample = 10 and max_depth = 1000 is 66.86%
```

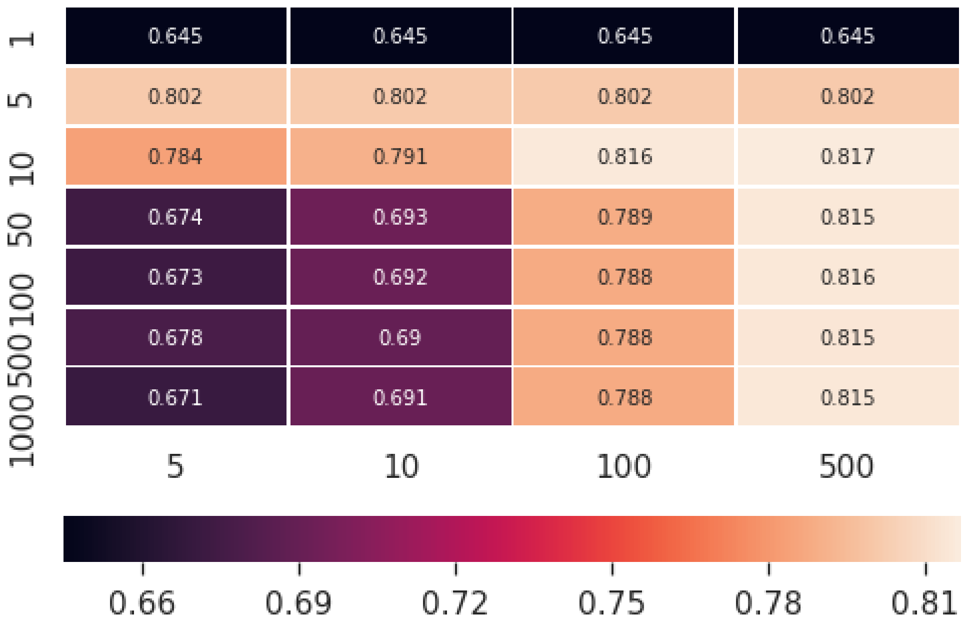
```
==================================================

Train AUC for min_sample = 100 and max_depth = 1000 is 93.81%

CV AUC for min_sample = 100 and max_depth = 1000 is 76.50%
==================================================

Train AUC for min_sample = 500 and max_depth = 1000 is 86.32%

CV AUC for min_sample = 500 and max_depth = 1000 is 79.87%
==================================================
```

In [82]:

```
# cv auc
cv_scores = np.array(tfidf_w2v_cv_auc).reshape(len(depth),len(samples))
df_cm = pd.DataFrame(cv_scores, depth, samples)
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, annot_kws={"size": 10}, fmt='.3g',linewidths=.5, cbar_kws={"orientation": "horizontal"})
```

Out[82]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3a7d6bdc18>
```

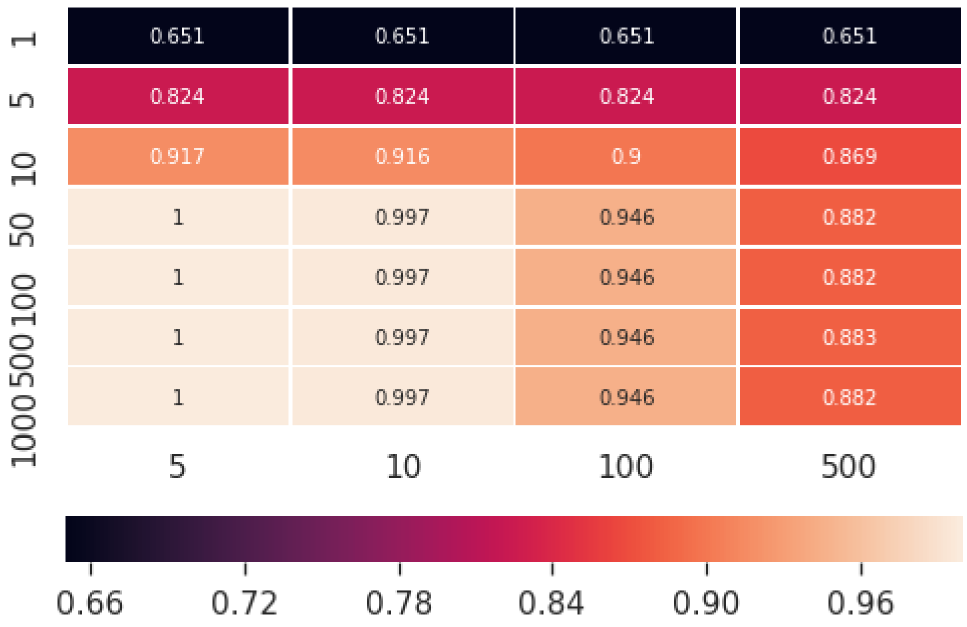


In [83]:

```
# train auc
train_scores = np.array(tfidf_w2v_cv_auc).reshape(len(depth),len(sample
s))
df_cm = pd.DataFrame(train_scores, depth, samples)
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, annot_kws={"size": 10}, fmt='.3g',linewi
dths=.5, cbar_kws={"orientation": "horizontal"})
```

Out[83]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3a7d636390>
```

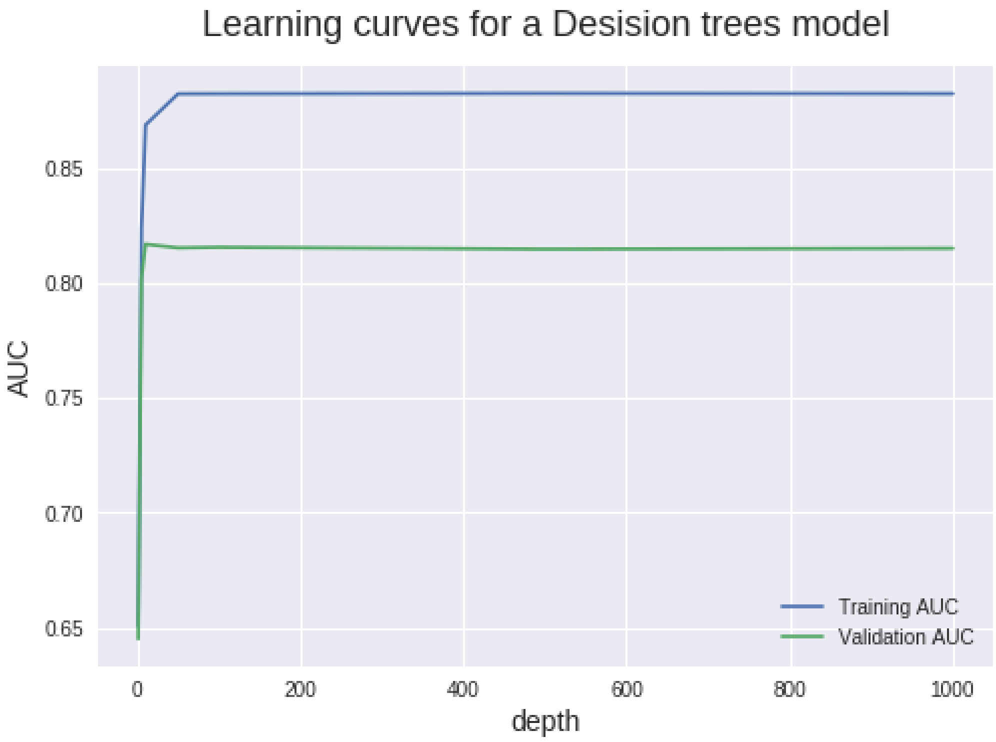


In [84]:

```
depth = [1, 5, 10, 50, 100, 500, 1000]
# https://www.dataquest.io/blog/learning-curves-machine-learning/

import matplotlib.pyplot as plt
%matplotlib inline

plt.style.use('seaborn')

plt.plot(depth,train_scores[:,-1],label = 'Training AUC')
plt.plot(depth,cv_scores[:,-1], label = 'Validation AUC')

plt.ylabel('AUC', fontsize = 14)
plt.xlabel('depth', fontsize = 14)
plt.title('Learning curves for a Desision trees model', fontsize = 18,
y = 1.03)
plt.legend()
```

Out[84]:

```
<matplotlib.legend.Legend at 0x7f3a7d53c3c8>
```



Learning curves for a Desision trees model

In [195]:

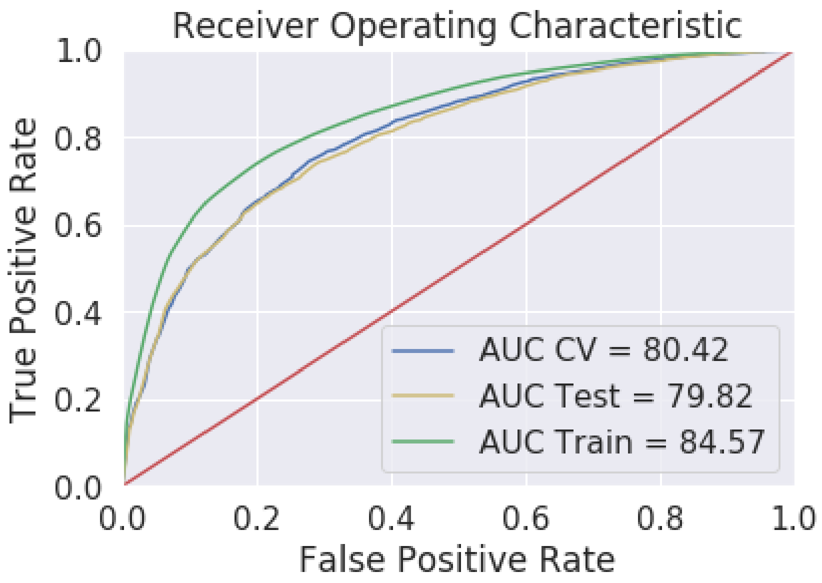
```
# Please write all the code with proper documentation
i = 10
j = 500

DTC = DecisionTreeClassifier(criterion='gini' , max_depth=i, min_sample
s_split=j)
DTC.fit(tfidf_w2v_train, train_y)
# train data
y_prob_train = DTC.predict_proba(tfidf_w2v_train)[:,1]
fprt, tprt, throsholdt = roc_curve(train_y, y_prob_train)
y_pred_train = np.where(y_prob_train > 0.5, 1, 0)
auc_roc_train = roc_auc_score(train_y , y_prob_train)
print('\nTrain AUC for min_sample = %s and depth = %s is %0.2f%%' % (st
r(j),str(i),(auc_roc_train * float(100))))
# CV
y_prob_cv = DTC.predict_proba(tfidf_w2v_cv)[:,1]
```

```
fprc, tprc, throsholdc = roc_curve(cv_y, y_prob_cv)
y_pred_cv = np.where(y_prob_cv > 0.5, 1, 0)
auc_roc_cv = roc_auc_score(cv_y , y_prob_cv)
print('\nCV AUC for min_sample = %s and depth = %s is %0.2f%%' % (str(j
),str(i),(auc_roc_cv * float(100))))
# Test
y_prob_test = DTC.predict_proba(tfidf_w2v_test)[:,1]
fprts, tprts, throsholdts = roc_curve(test_y, y_prob_test)
y_pred_test = np.where(y_prob_test > 0.5, 1, 0)
auc_roc_test = roc_auc_score(test_y , y_prob_test)
print('\nTest AUC for min_sample = %s and depth = %s is %0.2f%%' % (str
(j),str(i),(auc_roc_test * float(100))))
```

```
Train AUC for min_sample = 500 and depth = 10 is 84.57%

CV AUC for min_sample = 500 and depth = 10 is 80.42%

Test AUC for min_sample = 500 and depth = 10 is 79.82%
```

In [196]:

```
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in
-python

import matplotlib.pyplot as plt
plt.clf()
plt.title('Receiver Operating Characteristic')
plt.plot(fprc, tprc, 'b' , label ='AUC CV = %0.2f' % (auc_roc_cv * floa
t(100)))
plt.plot(fprts, tprts, 'y' , label ='AUC Test = %0.2f' % (auc_roc_test
* float(100)))
plt.plot(fprt, tprt, 'g', label='AUC Train = %0.2f' % (auc_roc_train *
float(100)))
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

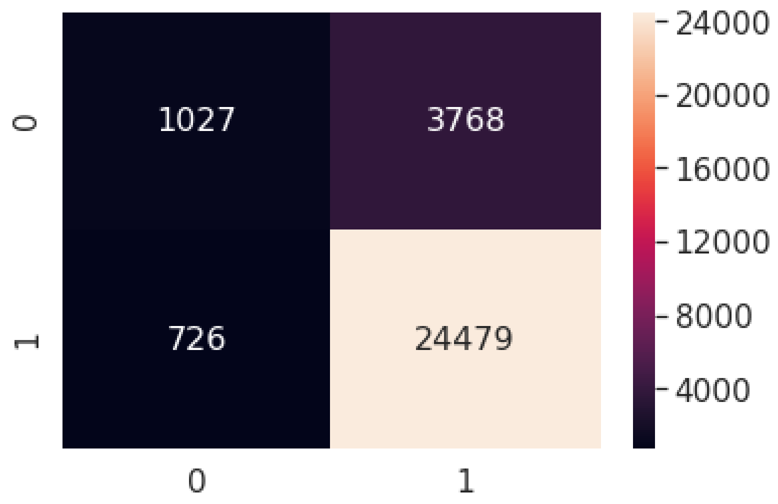


```
In [197]: print("F1-Score on test set: %0.2f"%(f1_score(test_y, y_pred_test)))
```

```
F1-Score on test set: 0.92
```

```
In [198]: df_cm = pd.DataFrame(confusion_matrix(test_y, y_pred_test), range(2), r
ange(2))
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
Out[198]: <matplotlib.axes._subplots.AxesSubplot at 0x7fce88ee08d0>
```



[6] Conclusions

In [199]:
```python
# Please compare all your models using Prettytable library
from prettytable import PrettyTable

x = PrettyTable(["Vectorizer" , "max_depth","min_sample_split", "AUC",
"F1 Score"])

x.add_row(["BOW", 50 ,500, "82.73%", 0.92])
x.add_row(["TFIDF",50 ,500,"83.06%", 0.92])
x.add_row(["AVG-W2V",10 , 500,"83.11%", 0.92])
x.add_row(["TFIDF-W2V",10 , 500,"79.82%", 0.92])

print(x.get_string(title="DTC"))
```

```
+------------+-----------+------------------+--------+----------+
| Vectorizer | max_depth | min_sample_split |  AUC   | F1 Score |
+------------+-----------+------------------+--------+----------+
|    BOW     |     50    |       500        | 82.73% |   0.92   |
|   TFIDF    |     50    |       500        | 83.06% |   0.92   |
|  AVG-W2V   |     10    |       500        | 83.11% |   0.92   |
| TFIDF-W2V  |    10     |      500         | 79.82% |   0.92   |
+------------+-----------+------------------+--------+----------+
```

In [ ]: