Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

Objective:

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

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

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

In [1]:
```
rom google.colab import drive
rive.mount('/content/drive')
```

```
o to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee649
hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fww
.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.
om%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

nter your authorization code:
·····
ounted at /content/drive
```

[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 [0]:
```
matplotlib inline
mport warnings
arnings.filterwarnings("ignore")


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

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

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

rom gensim.models import Word2Vec
rom gensim.models import KeyedVectors
mport pickle

rom tqdm import tqdm
mport os

rom sklearn.model_selection import GridSearchCV,train_test_split

rom sklearn.linear_model import LogisticRegression
```

In [0]:
```
 using SQLite Table to read data.
on = sqlite3.connect('database.sqlite')

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

 filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000""", con)
 for tsne assignment you can take 5k data points

iltered_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 score<3 a negative rating(0).
ef partition(x):
   if x < 3:
       return 0
   return 1

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

```
Number of data points in our data (5000, 10)
```

Out[0]:

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

In [0]:
```
 isplay = pd.read_sql_query("""
 ELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
 ROM Reviews
 ROUP BY UserId
 AVING COUNT(*)>1
""", con)
```

In [0]:
```
rint(display.shape)
isplay.head()
```

```
(80668, 7)
```

Out[0]:

	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 [0]:
```
isplay[display['UserId']=='AZY10LLTJ71NX']
```

Out[0]:

	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 [0]:
```
isplay['COUNT(*)'].sum()
```

Out[0]:
```
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 [0]:
```
 isplay= pd.read_sql_query("""
 ELECT *
 ROM Reviews
 HERE Score != 3 AND UserId="AR5J8UI46CURR"
 RDER BY ProductID
""", con)
 isplay.head()
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Sum
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFEF
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFEF
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFEF
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFEF
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADF VANILL WAFEF

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 [0]:
```
Sorting data according to ProductId in ascending order
orted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_positio
='last')
```

In [0]:
```
Deduplication of entries
inal=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
inal.shape
```

Out[0]: (4986, 10)

In [0]:
```
Checking to see how much % of data still remains
final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[0]: 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 [0]:
```
 isplay= pd.read_sql_query("""
 ELECT *
 ROM Reviews
 HERE Score != 3 AND Id=44737 OR Id=64422
 RDER BY ProductID
""", con)

 isplay.head()
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summa
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste wi crunchy almond inside

In [0]:
```
inal=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [0]:
```
Before starting the next phase of preprocessing lets see the number of entries left
rint(final.shape)

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

```
(4986, 10)
```

Out[0]:
```
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 [0]:
```
 printing some random reviews
ent_0 = final['Text'].values[0]
rint(sent_0)
rint("="*50)

ent_1000 = final['Text'].values[1000]
rint(sent_1000)
rint("="*50)

ent_1500 = final['Text'].values[1500]
rint(sent_1500)
rint("="*50)

ent_4900 = final['Text'].values[4900]
rint(sent_4900)
rint("="*50)
```

```
 hy is this $[...] when the same product is available for $[...] here?<br />http://www.amazon.com/VICTOR-FLY-MAGNET-B
 IT-REFILL/dp/B00004RBDY<br /><br />The Victor M380 and M502 traps are unreal, of course -- total fly genocide. Prett
  stinky, but only right nearby.
==================================================
  recently tried this flavor/brand and was surprised at how delicious these chips are.  The best thing was that there
 ere a lot of "brown" chips in the bsg (my favorite), so I bought some more through amazon and shared with family and
 riends.  I am a little disappointed that there are not, so far, very many brown chips in these bags, but the flavor
 s still very good.  I like them better than the yogurt and green onion flavor because they do not seem to be as salt
 , and the onion flavor is better.  If you haven't eaten Kettle chips before, I recommend that you try a bag before b
 ying bulk.  They are thicker and crunchier than Lays but just as fresh out of the bag.
==================================================
 ow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the other wants crispy cookie
 .  Hey, I'm sorry; but these reviews do nobody any good beyond reminding us to look  before ordering.<br /><br />The
 e are chocolate-oatmeal cookies.  If you don't like that combination, don't order this type of cookie.  I find the c
 mbo quite nice, really.  The oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sort of a coconu
 -type consistency.  Now let's also remember that tastes differ; so, I've given my opinion.<br /><br />Then, these ar
  soft, chewy cookies -- 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 don't see where these taste like raw cookie dough.  Both are
 oft, however, so is this the confusion?  And, yes, they stick together.  Soft cookies tend to do that.  They aren't
 ndividually wrapped, which would add to the cost.  Oh yeah, chocolate chip cookies tend to be somewhat sweet.<br /><
 r />So, if you want something hard and crisp, I suggest Nabiso's Ginger Snaps.  If you want a cookie that's soft, ch
 wy and tastes like a combination of chocolate and oatmeal, give these a try.  I'm here to place my second order.
==================================================
 ove to order my coffee on amazon.  easy and shows up quickly.<br />This k cup is great coffee.  dcaf is very good as
 ell
==================================================
```

In [0]:

```
 remove urls from text python: https://stackoverflow.com/a/40823105/4084039
ent_0 = re.sub(r"http\S+", "", sent_0)
ent_1000 = re.sub(r"http\S+", "", sent_1000)
ent_150 = re.sub(r"http\S+", "", sent_1500)
ent_4900 = re.sub(r"http\S+", "", sent_4900)

rint(sent_0)
```

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

In [0]:
```
 https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-element
rom bs4 import BeautifulSoup

oup = BeautifulSoup(sent_0, 'lxml')
ext = soup.get_text()
rint(text)
rint("="*50)

oup = BeautifulSoup(sent_1000, 'lxml')
ext = soup.get_text()
rint(text)
rint("="*50)

oup = BeautifulSoup(sent_1500, 'lxml')
ext = soup.get_text()
rint(text)
rint("="*50)

oup = BeautifulSoup(sent_4900, 'lxml')
ext = soup.get_text()
rint(text)
```

```
hy is this $[...] when the same product is available for $[...] here? />The Victor M380 and M502 traps are unreal, o
 course -- total fly genocide. Pretty stinky, but only right nearby.
==================================================
 recently tried this flavor/brand and was surprised at how delicious these chips are.  The best thing was that there
ere a lot of "brown" chips in the bsg (my favorite), so I bought some more through amazon and shared with family and
riends.  I am a little disappointed that there are not, so far, very many brown chips in these bags, but the flavor
s still very good.  I like them better than the yogurt and green onion flavor because they do not seem to be as salt
, and the onion flavor is better.  If you haven't eaten Kettle chips before, I recommend that you try a bag before b
ying bulk.  They are thicker and crunchier than Lays but just as fresh out of the bag.
==================================================
ow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the other wants crispy cookie
.  Hey, I'm sorry; but these reviews do nobody any good beyond reminding us to look  before ordering.These are choco
ate-oatmeal cookies.  If you don't like that combination, don't order this type of cookie.  I find the combo quite n
ce, really.  The oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sort of a coconut-type consi
tency.  Now let's also remember that tastes differ; so, I've given my opinion.Then, these are soft, chewy cookies --
s advertised.  They are not "crispy" cookies, or the blurb would say "crispy," rather than "chewy."  I happen to lik
 raw cookie dough; however, I don't see where these taste like raw cookie dough.  Both are soft, however, so is this
he confusion?  And, yes, they stick together.  Soft cookies tend to do that.  They aren't individually wrapped, whic
 would add to the cost.  Oh yeah, chocolate chip cookies tend to be somewhat sweet.So, if you want something hard an
 crisp, I suggest Nabiso's Ginger Snaps.  If you want a cookie that's soft, chewy and tastes like a combination of c
ocolate and oatmeal, give these a try.  I'm here to place my second order.
==================================================
ove to order my coffee on amazon.  easy and shows up quickly.This k cup is great coffee.  dcaf is very good as well
```

In [0]:
```
 https://stackoverflow.com/a/47091490/4084039
mport re

ef 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 [0]:
```
ent_1500 = decontracted(sent_1500)
rint(sent_1500)
rint("="*50)
```

```
ow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the other wants crispy cookie
.  Hey, I am sorry; but these reviews do nobody any good beyond reminding us to look  before ordering.<br /><br />Th
se are chocolate-oatmeal cookies.  If you do not like that combination, do not order this type of cookie.  I find th
 combo quite nice, really.  The oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sort of a coc
nut-type consistency.  Now let is also remember that tastes differ; so, I have given my opinion.<br /><br />Then, th
se are soft, chewy cookies -- as advertised.  They are not "crispy" cookies, or the blurb would say "crispy," rather
han "chewy."  I happen to like raw cookie dough; however, I do not see where these taste like raw cookie dough.  Bot
 are soft, however, so is this the confusion?  And, yes, they stick together.  Soft cookies tend to do that.  They a
e not 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 is Ginger Snaps.  If you want a cookie that is
oft, chewy and tastes like a combination of chocolate and oatmeal, give these a try.  I am here to place my second o
der.
==================================================
```

In [0]:
```
remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
ent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
rint(sent_0)
```

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

In [0]:
```
remove spacial character: https://stackoverflow.com/a/5843547/4084039
ent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
rint(sent_1500)
```

```
ow So far two two star reviews One obviously had no idea what they were ordering the other wants crispy cookies Hey
 am sorry but these reviews do nobody any good beyond reminding us to look before ordering br br These are chocolate
atmeal cookies If you do not like that combination do not order this type of cookie I find the combo quite nice real
y The oatmeal sort of calms the rich chocolate flavor and gives the cookie sort of a coconut type consistency Now le
 is also remember that tastes differ so I have given my opinion br br Then these are soft chewy cookies as advertise
 They are not crispy cookies or the blurb would say crispy rather than chewy I happen to like raw cookie dough howev
r I do not see where these taste like raw cookie dough Both are soft however so is this the confusion And yes they s
ick together Soft cookies tend to do that They are not individually wrapped which would add to the cost Oh yeah choc
late chip cookies tend to be somewhat sweet br br So if you want something hard and crisp I suggest Nabiso is Ginger
naps If you want a cookie that is soft chewy and tastes like a combination of chocolate and oatmeal give these a try
 am here to place my second order
```

In [0]:
```
 https://gist.github.com/sebleier/554280
 we are removing the words from the stop words list: 'no', 'nor', 'not'
 <br /><br /> ==> after the above steps, we are getting "br br"
 we are including them into stop words list
 instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

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

In [0]:
```
 Combining all the above stundents
rom tqdm import tqdm
reprocessed_reviews = []
 tqdm is for printing the status bar
or 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 [0]:
```
reprocessed_reviews[1500]
```

Out[0]:
```
'wow far two two star reviews one obviously no idea ordering wants crispy cookies hey sorry reviews nobody good beyon
 reminding us look ordering chocolate oatmeal cookies not like combination not order type cookie find combo quite ni
e really oatmeal sort calms rich chocolate flavor gives cookie sort coconut type consistency let also remember taste
 differ given opinion soft chewy cookies advertised not crispy cookies blurb would say crispy rather chewy happen li
e raw cookie dough however not see taste like raw cookie dough soft however confusion yes stick together soft cookie
 tend not individually wrapped would add cost oh yeah chocolate chip cookies tend somewhat sweet want something hard
risp suggest nabiso ginger snaps want cookie soft chewy tastes like combination chocolate oatmeal give try place sec
nd order'
```

[3.2] Preprocessing Review Summary

In [0]:
```
# Similartly you can do preprocessing for review summary also.
```

[4] Featurization

In [0]:
```
ith open('/content/drive/My Drive/Applied AI assignments/preprocessed_reviews.pkl','rb') as f:
 preprocessed_reviews=pickle.load(f)
ith open('/content/drive/My Drive/Applied AI assignments/list_of_scores.pkl','rb') as f:
 list_of_scores=pickle.load(f)
```

[4.1] BAG OF WORDS

In [0]:
```
BoW
ount_vect = CountVectorizer() #in scikit-learn
ount_vect.fit(preprocessed_reviews)
rint("some feature names ", count_vect.get_feature_names()[:10])
rint('='*50)

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

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

[4.2] Bi-Grams and n-Grams.

In [0]:
```
bi-gram, tri-gram and n-gram

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

 you can choose these numebrs min_df=10, max_features=5000, of your choice
ount_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
inal_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
rint("the type of count vectorizer ",type(final_bigram_counts))
rint("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
rint("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 [0]:
```
f_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
f_idf_vect.fit(preprocessed_reviews)
rint("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
rint('='*50)

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

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

[4.4] Word2Vec

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

In [0]:
```
 Using Google News Word2Vectors

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


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

s_your_ram_gt_16g=False
ant_to_use_google_w2v = False
ant_to_train_w2v = True

f 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'))

lif 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_train_w2v = True, to train your own w2v ")
```

```
 ('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wonderful', 0.9946032166481018), ('excellent', 0.9
 44332838058472), ('especially', 0.9941144585609436), ('baked', 0.9940600395202637), ('salted', 0.994047224521637),
 'alternative', 0.9937226176261902), ('tasty', 0.9936816692352295), ('healthy', 0.9936649799346924)]
==================================================
 ('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('popcorn', 0.9992750883102417), ('de', 0.9992610
 16140747), ('miss', 0.9992451071739197), ('melitta', 0.999218761920929), ('choice', 0.9992102384567261), ('america
 ', 0.9991837739944458), ('beef', 0.9991780519485474), ('finish', 0.9991567134857178)]
```

In [0]:
```
 2v_words = list(w2v_model.wv.vocab)
 rint("number of words that occured minimum 5 times ",len(w2v_words))
 rint("sample words ", w2v_words[0:50])
```

```
 umber of words that occured minimum 5 times  3817
 ample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby', 'used', 'ca', 'not',
'beat', 'great', 'received', 'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'instead', 'removed', 'eas
 ly', 'daughter', 'designed', 'printed', 'use', 'car', 'windows', 'beautifully', 'shop', 'program', 'going', 'lot',
'fun', 'everywhere', 'like', 'tv', 'computer', 'really', 'good', 'idea', 'final', 'outstanding', 'window', 'everybod
 ', 'asks', 'bought', 'made']
```

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

[4.4.1.1] Avg W2v

In [0]:
```
 average Word2Vec
 compute average word2vec for each review.
ent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
or sent in tqdm(list_of_sentance): # for each review/sentence
   sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you use g
ogle's w2v
   cnt_words =0; # num of words with a valid vector in the sentence/review
   for word in sent: # for each word in a review/sentence
       if word in w2v_words:
           vec = w2v_model.wv[word]
           sent_vec += vec
           cnt_words += 1
   if cnt_words != 0:
       sent_vec /= cnt_words
   sent_vectors.append(sent_vec)
rint(len(sent_vectors))
rint(len(sent_vectors[0]))
```

```
100%|██████████| 4986/4986 [00:03<00:00, 1330.47it/
s]

4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [0]:
```
 S = ["abc def pqr", "def def def abc", "pqr pqr def"]
odel = TfidfVectorizer()
f_idf_matrix = model.fit_transform(preprocessed_reviews)
 we are converting a dictionary with word as a key, and the idf as a value
ictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [0]:
```
 TF-IDF weighted Word2Vec
fidf_feat = model.get_feature_names() # tfidf words/col-names
 final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

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

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

[5] Assignment 5: Apply Logistic Regression

1. **Apply Logistic Regression 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. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**

   - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) 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. **Pertubation Test**

   - Get the weights W after fit your model with the data X i.e Train data.
   - Add a noise to the X (X' = X + e) and get the new data set X' (if X is a sparse matrix, X.data+=e)
   - Fit the model again on data X' and get the weights W'
   - Add a small eps value(to eliminate the divisible by zero error) to W and W' i.e W=W+10^-6 and W' = W'+10^-6
   - Now find the % change between W and W' (| (W-W') / (W) |)*100)
   - Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden rise in the values of percentage_change_vector
   - Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there is sudden rise from 1.3 to 34.6, now calculate the 99.1, 99.2, 99.3,..., 100th percentile values and get the proper value after which there is sudden rise the values, assume it is 2.5
   - Print the feature names whose % change is more than a threshold x(in our example it's 2.5)

4. **Sparsity**

   - Calculate sparsity on weight vector obtained after using L1 regularization

   NOTE: Do sparsity and multicollinearity for any one of the vectorizers. Bow or tf-idf is recommended.

5. **Feature importance**

   - Get top 10 important features for both positive and negative classes separately.

6. **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.

7. **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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.
      (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
     (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
     (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

     (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
8. **Conclusion** (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

   (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
   - 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 (https://seaborn.pydata.org/generated/seaborn.heatmap.html) link (http://zetcode.com/python/prettytable/)

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. (https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

Applying Logistic Regression

[5.1] Logistic Regression on BOW, SET 1

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

In [0]:
```
=preprocessed_reviews
=list_of_scores
```

In [0]:
```
split the dataset into train and test sets
_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
```

In [0]:
```
vectorizing using BoW
owVectorizer=CountVectorizer()
owVectorizer.fit(X_train)
_train_BOW=bowVectorizer.transform(X_train)
_test_BOW=bowVectorizer.transform(X_test)
```

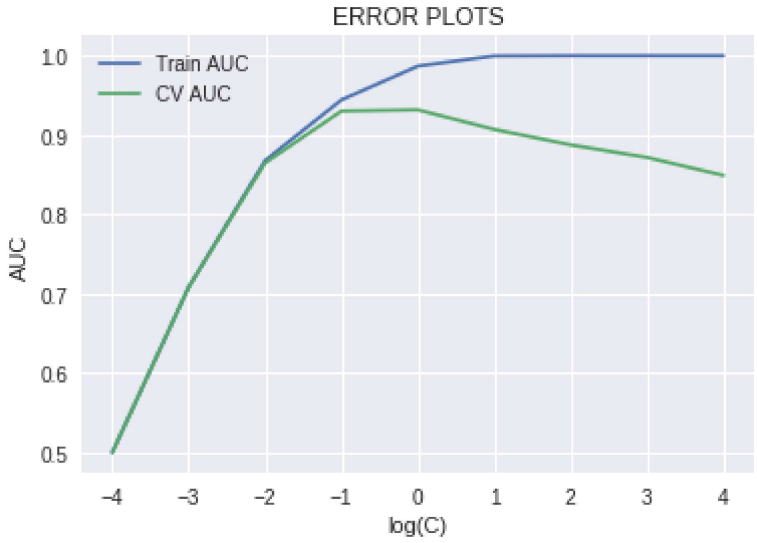
In [0]:
```
Using GridSearhCV for finding the best C
rModel=LogisticRegression(penalty='l1')
yper_parameters={'C':[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]}
odel=GridSearchCV(lrModel,hyper_parameters,scoring='roc_auc',cv=3)
odel.fit(X_train_BOW,y_train)

odel.best_estimator_
odel.best_score_
```

Out[0]:
```
0.9318178140362692
```

In [0]:
```
odel.best_estimator_
```

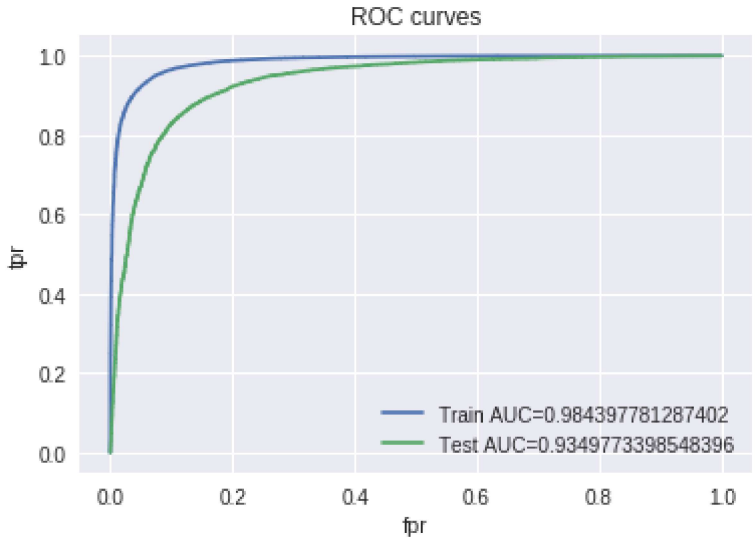
Out[0]:
```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [0]:
```
plotting the train AUC vs CV AUC for different log(C) values.
log is considered because, different scale of values can be displayed on the plot
_values=[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]
rain_auc= model.cv_results_['mean_train_score']
v_auc = model.cv_results_['mean_test_score']

=np.log10(C_values)

lt.plot(C, train_auc, label='Train AUC')
 this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alpha_values,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

lt.plot(C, cv_auc, label='CV AUC')
 this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alpha_values,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
lt.legend()
lt.xlabel("log(C)")
lt.ylabel("AUC")
lt.title("ERROR PLOTS")
lt.show()
```



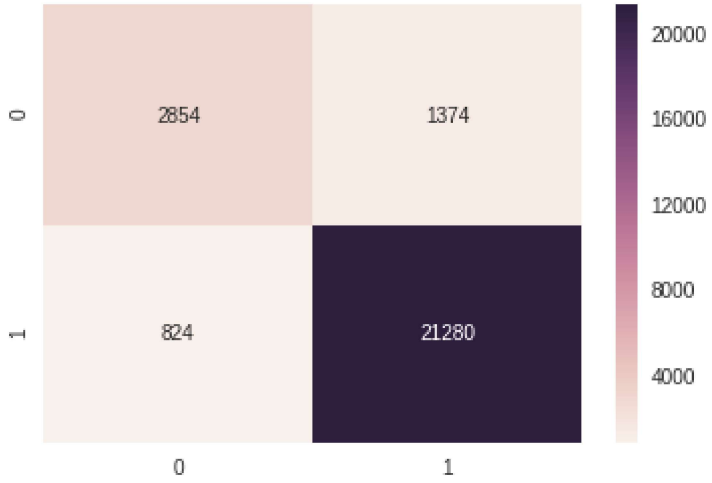
In [0]:
```
est_C=1
```

In [0]:
```
training the model with best C and plotting ROC curves on both train and test data
rModel=LogisticRegression(penalty='l1',C=best_C)
rModel.fit(X_train_BOW,y_train)

rain_fpr,train_tpr,thresholds=roc_curve(y_train,lrModel.predict_proba(X_train_BOW)[:,1])
est_fpr,test_tpr,thresholds=roc_curve(y_test,lrModel.predict_proba(X_test_BOW)[:,1])

lt.plot(train_fpr,train_tpr,label='Train AUC='+str(auc(train_fpr,train_tpr)))
lt.plot(test_fpr,test_tpr,label='Test AUC='+str(auc(test_fpr,test_tpr)))
lt.legend()
lt.xlabel('fpr')
lt.ylabel('tpr')
lt.title('ROC curves')
lt.show()
```



In [0]:
```
confusion matrix using seaborn heatmap

_pred=lrModel.predict(X_test_BOW)
m=confusion_matrix(y_test,y_pred)
ns.heatmap(cm,annot=True,fmt='.5g')
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd197f66cc0>



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

In [0]:
```
calculating sparsity on weight vector obtained using L1 regularization on BoW
eight_vector=lrModel.coef_
otal_no_of_elem=weight_vector.size
onzero_elem=np.count_nonzero(weight_vector)
parsity=(total_no_of_elem-nonzero_elem)/total_no_of_elem
rint('The sparsity of weight vector obtained using L1 regularization is {}'.format(sparsity))
```

```
The sparsity of weight vector obtained using L1 regularization is 0.8997330232958705
```

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

In [0]:
```
using GridSearchCV to determing the best CV for logisticc regression with L2 regularization
rModel=LogisticRegression(penalty='l2')
yper_parameters={'C':[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]}
odel=GridSearchCV(lrModel,hyper_parameters,scoring='roc_auc',cv=3)
odel.fit(X_train_BOW,y_train)
```

Out[0]:
```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

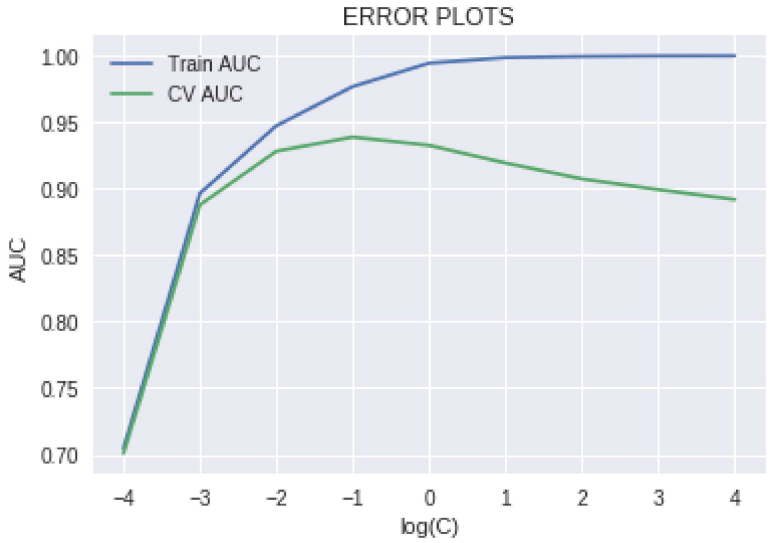
In [0]:
```
odel.best_estimator_
```

Out[0]:
```
LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [0]:
```
odel.best_score_
```

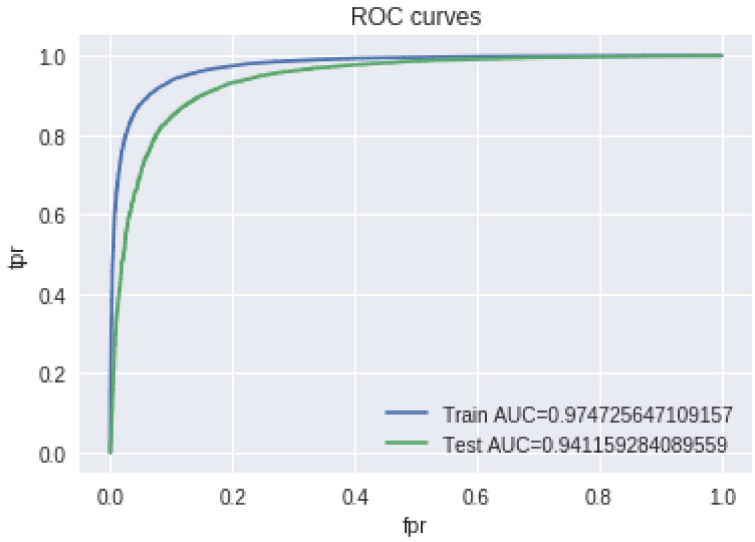
Out[0]:
```
0.9388322723684638
```

In [0]:
```
train AUC vs test AUC
_values=[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]
rain_auc= model.cv_results_['mean_train_score']
v_auc = model.cv_results_['mean_test_score']

=np.log10(C_values)

lt.plot(C, train_auc, label='Train AUC')
 this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alpha_values,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

lt.plot(C, cv_auc, label='CV AUC')
 this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alpha_values,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
lt.legend()
lt.xlabel("log(C)")
lt.ylabel("AUC")
lt.title("ERROR PLOTS")
lt.show()
```



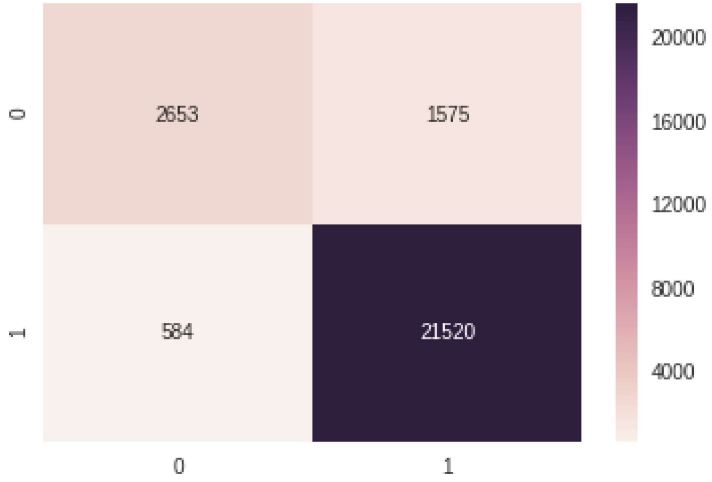
In [0]:
```
est_C=0.1
```

In [75]:
```
training the model with best and plotting ROC curves for train and test data
rModel=LogisticRegression(penalty='l2',C=best_C)
rModel.fit(X_train_BOW,y_train)

rain_fpr,train_tpr,thresholds=roc_curve(y_train,lrModel.predict_proba(X_train_BOW)[:,1])
est_fpr,test_tpr,thresholds=roc_curve(y_test,lrModel.predict_proba(X_test_BOW)[:,1])

lt.plot(train_fpr,train_tpr,label='Train AUC='+str(auc(train_fpr,train_tpr)))
lt.plot(test_fpr,test_tpr,label='Test AUC='+str(auc(test_fpr,test_tpr)))
lt.legend()
lt.xlabel('fpr')
lt.ylabel('tpr')
lt.title('ROC curves')
lt.show()
```



In [0]:
```
confusion matrix using seaborn heatmap

_pred=lrModel.predict(X_test_BOW)
m=confusion_matrix(y_test,y_pred)
ns.heatmap(cm,annot=True,fmt='.5g')
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7f4568eb18d0>



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

In [0]:
```
weight vector W
=lrModel.coef_
```

In [0]:
```
generate noise from normal distribution (mean=0 and std=0.1)
=np.random.normal(loc=0.0, scale=0.1, size=None)
```

In [0]:
```
adding the error to the train dataset
_train_BOW=X_train_BOW.astype('float64')
_train_BOW.data+=e
```

In [80]:
```
training the model on new dataset (with noise)
rModel=LogisticRegression(penalty='l2',C=best_C)
rModel.fit(X_train_BOW,y_train)
```

Out[80]:
```
LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [0]:
```
new weight vector w_new
_new=lrModel.coef_
```

In [0]:
```
adding a small eps value to w and w_new to avoid division by zero error
=w+10**-5
_new=w_new+10**-5
```

In [0]:
```
finding the ppercentage change of weight vectors
iff_w_and_wnew=np.subtract(w,w_new)
ercent_change_in_weights=np.absolute(np.divide(diff_w_and_wnew,w))*100
```

In [0]:
```
calculating percentiles 0,10,20,30,.....100th percentile
ercentiles=[x for x in range(0,110,10)]
ercentile_values=[]
or p in percentiles:
 percentile_values.append(np.percentile(percent_change_in_weights,p))
```

In [91]:
```
ercentile_values
```

Out[91]:
```
[0.0001185713175136682,
 0.9846196282428765,
 1.9933017214623678,
 3.0482663111395114,
 4.118409590879273,
 5.301226907020326,
 6.570835871392785,
 8.123048369118022,
 10.31358485948647,
 14.82530743051162,
 89722.9282600134]
```

In [0]:
```
ercentile_values_2=[]
or p in range(90,101):
 percentile_values_2.append(np.percentile(percent_change_in_weights,p))
```

In [93]:
```
ercentile_values_2
```

Out[93]:
```
[14.82530743051162,
 15.62371325662323,
 16.656428526141813,
 17.92805599247506,
 19.480114184036122,
 21.3808656442978,
 24.587104508894043,
 30.726848430526168,
 42.24108169009264,
 73.39763953172897,
 89722.9282600134]
```

In [0]:
```
threshold value for obtainiing the features
hreshold=73.39
```

In [0]:
```
get the feature names
eatures=bowVectorizer.get_feature_names()
```

In [0]:
```
https://docs.scipy.org/doc/numpy/reference/generated/numpy.argwhere.html
ndices_array=np.argwhere(percent_change_in_weights>=threshold)
ndices=[]
or i in range(indices_array.shape[0]):
 indices.append(indices_array[i][1])
```

In [0]:
```
features with %change in weights more than threshold
eatures=np.array(features)
eatures_above_threshold=list(features[indices])
```

In [120]:
```
printing only first 10 features with %change in weights more than threshold
eatures_above_threshold[:10]
```

Out[120]:
```
['abilities',
 'acted',
 'adjusted',
 'advantage',
 'affected',
 'agenda',
 'aggravated',
 'aggrevated',
 'alcoholic',
 'alkaloid']
```

[5.1.3] Feature Importance on BOW, SET 1

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

In [0]:
```
get features
eatures=bowVectorizer.get_feature_names()
get weight vector as list
eight_vector=lrModel.coef_.tolist()[0]
get the dictionary with key as feature and its weight as value
eat_weight_dict=dict(zip(features,weight_vector))
```

In [0]:
```
https://www.python.org/dev/peps/pep-0265/
eat_weight_dict_items=[(v,k) for k,v in feat_weight_dict.items()]
eat_weight_dict_items.sort()
eat_weight_dict_items.reverse()
```

In [123]:
```
op10_imp_feat_pos_cls=[k for v,k in feat_weight_dict_items][:10]
op10_imp_feat_pos_cls
```

Out[123]:
```
['delicious',
 'perfect',
 'excellent',
 'wonderful',
 'awesome',
 'great',
 'yummy',
 'amazing',
 'best',
 'hooked']
```

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

In [124]:
```
eatures_ordered_by_weights=[k for v,k in feat_weight_dict_items]
eatures_ordered_by_weights.reverse()
op10_imp_feat_neg_cls=features_ordered_by_weights[:10]
op10_imp_feat_neg_cls
```

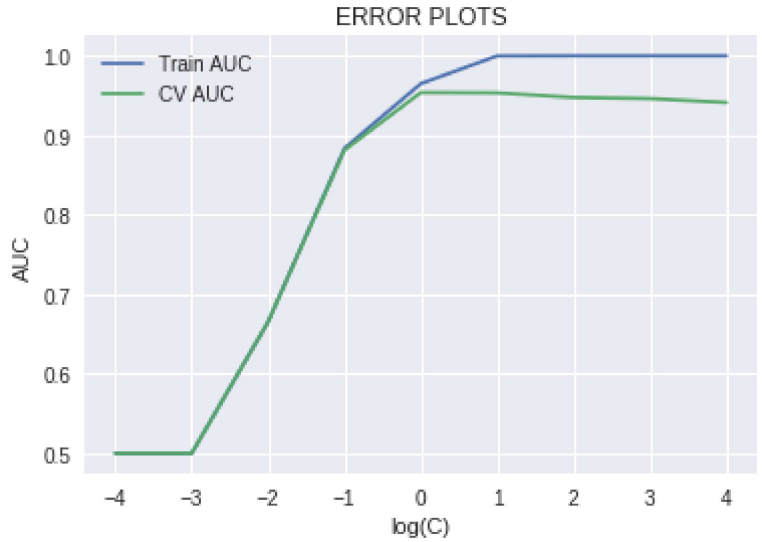
Out[124]:
```
['worst',
 'disappointing',
 'terrible',
 'disappointment',
 'awful',
 'rip',
 'threw',
 'horrible',
 'tasteless',
 'sorry']
```

[5.2] Logistic Regression on TFIDF, SET 2

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

In [0]:
```
Tfidf vectorization
fidf_vectorizer=TfidfVectorizer(ngram_range=(1,2),min_df=10)
fidf_vectorizer.fit(X_train)
_train_tfidf=tfidf_vectorizer.transform(X_train)
_test_tfidf=tfidf_vectorizer.transform(X_test)
```

In [0]:
```
determining the best C using GridSearchCV
rModel=LogisticRegression(penalty='l1')
yper_parameters={'C':[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]}
odel=GridSearchCV(lrModel,hyper_parameters,scoring='roc_auc',cv=3)
odel.fit(X_train_tfidf,y_train)

_values=[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]
rain_auc= model.cv_results_['mean_train_score']
v_auc = model.cv_results_['mean_test_score']

=np.log10(C_values)

lt.plot(C, train_auc, label='Train AUC')
 this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alpha_values,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

lt.plot(C, cv_auc, label='CV AUC')
 this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alpha_values,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
lt.legend()
lt.xlabel("log(C)")
lt.ylabel("AUC")
lt.title("ERROR PLOTS")
lt.show()
```

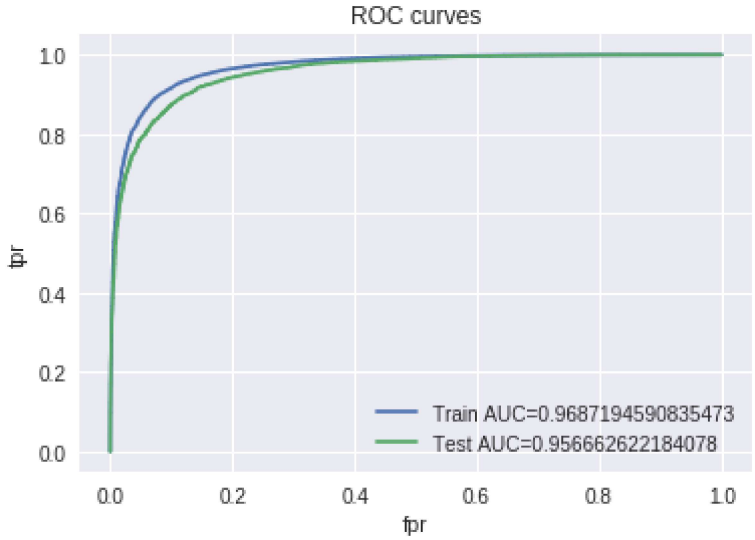


In [0]:
```
odel.best_estimator_
```

Out[0]:
```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

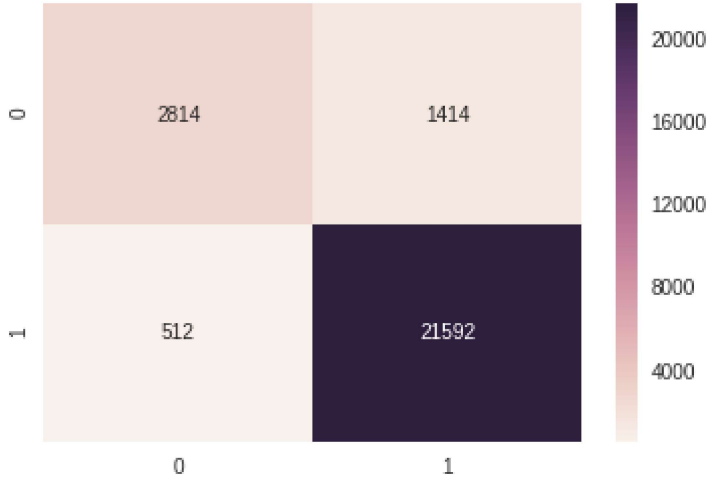
In [0]:
```
est_C=1
```

In [0]:
```
Training model on best C
ROC curves for train and test data
rModel=LogisticRegression(penalty='l1',C=best_C)
rModel.fit(X_train_tfidf,y_train)

rain_fpr,train_tpr,thresholds=roc_curve(y_train,lrModel.predict_proba(X_train_tfidf)[:,1])
est_fpr,test_tpr,thresholds=roc_curve(y_test,lrModel.predict_proba(X_test_tfidf)[:,1])

lt.plot(train_fpr,train_tpr,label='Train AUC='+str(auc(train_fpr,train_tpr)))
lt.plot(test_fpr,test_tpr,label='Test AUC='+str(auc(test_fpr,test_tpr)))
lt.legend()
lt.xlabel('fpr')
lt.ylabel('tpr')
lt.title('ROC curves')
lt.show()
```



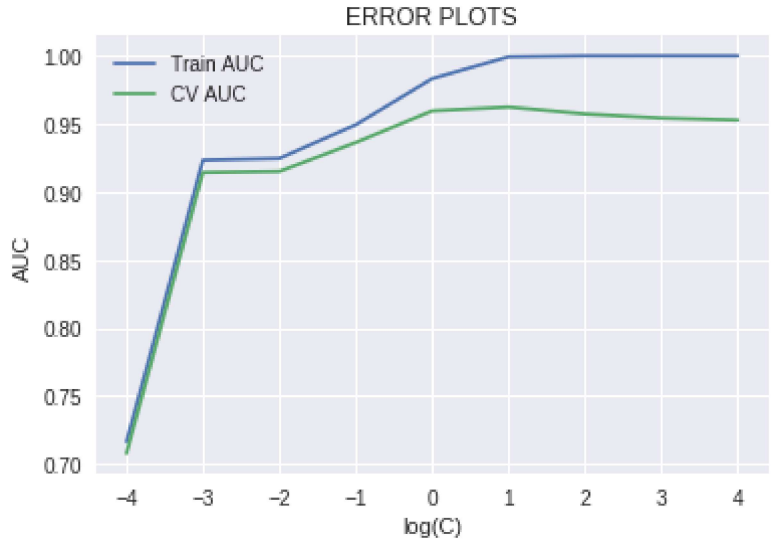
In [0]:
```
confusion matrix using seaborn heatmap

_pred=lrModel.predict(X_test_tfidf)
m=confusion_matrix(y_test,y_pred)
ns.heatmap(cm,annot=True,fmt='.5g')
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7f29d48a40b8>



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

In [0]:
```
rModel=LogisticRegression(penalty='l2')
yper_parameters={'C':[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]}
odel=GridSearchCV(lrModel,hyper_parameters,scoring='roc_auc',cv=3)
odel.fit(X_train_tfidf,y_train)

_values=[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]
rain_auc= model.cv_results_['mean_train_score']
v_auc = model.cv_results_['mean_test_score']

=np.log10(C_values)

lt.plot(C, train_auc, label='Train AUC')
 this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alpha_values,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

lt.plot(C, cv_auc, label='CV AUC')
 this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alpha_values,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
lt.legend()
lt.xlabel("log(C)")
lt.ylabel("AUC")
lt.title("ERROR PLOTS")
lt.show()
```

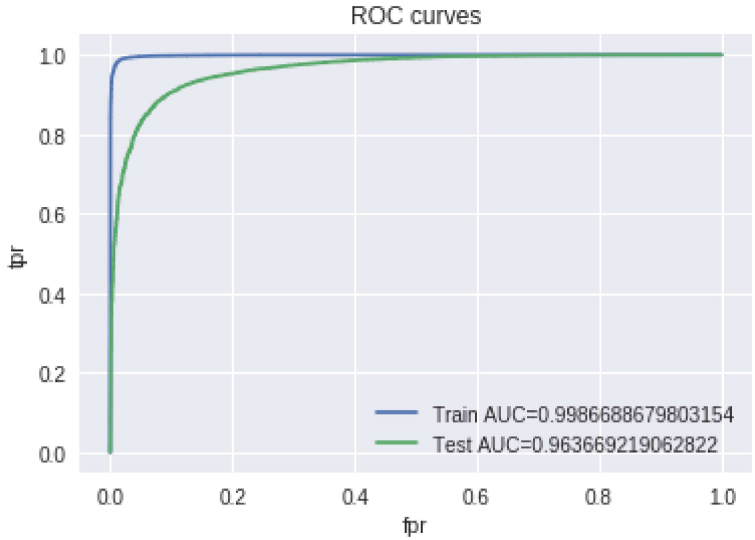


In [0]:
```
odel.best_estimator_
```

Out[0]:
```
LogisticRegression(C=10, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

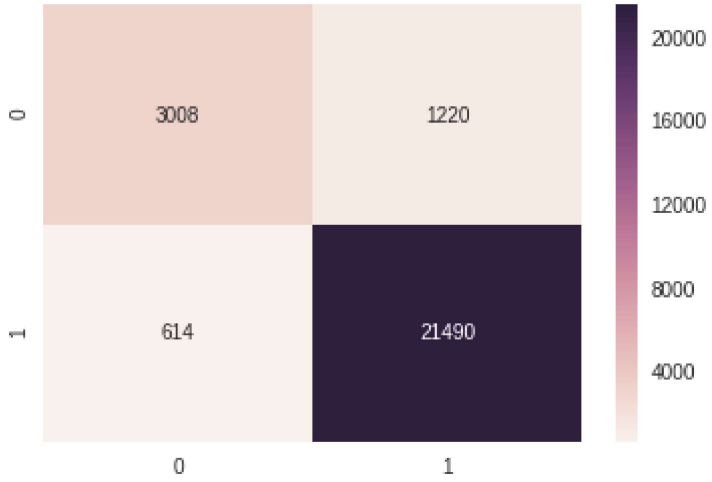
In [0]:
```
est_C=10
```

In [0]:
```
rModel=LogisticRegression(penalty='l2',C=best_C)
rModel.fit(X_train_tfidf,y_train)

rain_fpr,train_tpr,thresholds=roc_curve(y_train,lrModel.predict_proba(X_train_tfidf)[:,1])
est_fpr,test_tpr,thresholds=roc_curve(y_test,lrModel.predict_proba(X_test_tfidf)[:,1])

lt.plot(train_fpr,train_tpr,label='Train AUC='+str(auc(train_fpr,train_tpr)))
lt.plot(test_fpr,test_tpr,label='Test AUC='+str(auc(test_fpr,test_tpr)))
lt.legend()
lt.xlabel('fpr')
lt.ylabel('tpr')
lt.title('ROC curves')
lt.show()
```



In [0]:
```
confusion matrix using seaborn heatmap

_pred=lrModel.predict(X_test_tfidf)
m=confusion_matrix(y_test,y_pred)
ns.heatmap(cm,annot=True,fmt='.5g')
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7f29d8c12400>



[5.2.3] Feature Importance on TFIDF, SET 2

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

In [0]:
```
get features
eatures=tfidf_vectorizer.get_feature_names()
get weight vector as list
eight_vector=lrModel.coef_.tolist()[0]
get the dictionary with key as feature and its weight as value
eat_weight_dict=dict(zip(features,weight_vector))
```

In [0]:
```
https://www.python.org/dev/peps/pep-0265/
eat_weight_dict_items=[(v,k) for k,v in feat_weight_dict.items()]
eat_weight_dict_items.sort()
eat_weight_dict_items.reverse()
```

In [0]:
```
op10_imp_feat_pos_cls=[k for v,k in feat_weight_dict_items][:10]
op10_imp_feat_pos_cls
```

Out[0]:
```
['great',
 'delicious',
 'not disappointed',
 'best',
 'perfect',
 'good',
 'excellent',
 'nice',
 'wonderful',
 'loves']
```

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

In [0]:
```
eatures_ordered_by_weights=[k for v,k in feat_weight_dict_items]
eatures_ordered_by_weights.reverse()
op10_imp_feat_neg_cls=features_ordered_by_weights[:10]
op10_imp_feat_neg_cls
```

Out[0]:
```
['worst',
 'not worth',
 'disappointed',
 'not recommend',
 'disappointing',
 'not good',
 'terrible',
 'disappointment',
 'awful',
 'weak']
```

[5.3] Logistic Regression on AVG W2V, SET 3

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

In [0]:
```
 list of tokenized sentences
=0
ist_of_sentance=[]
or sentance in preprocessed_reviews:
   list_of_sentance.append(sentance.split())
```

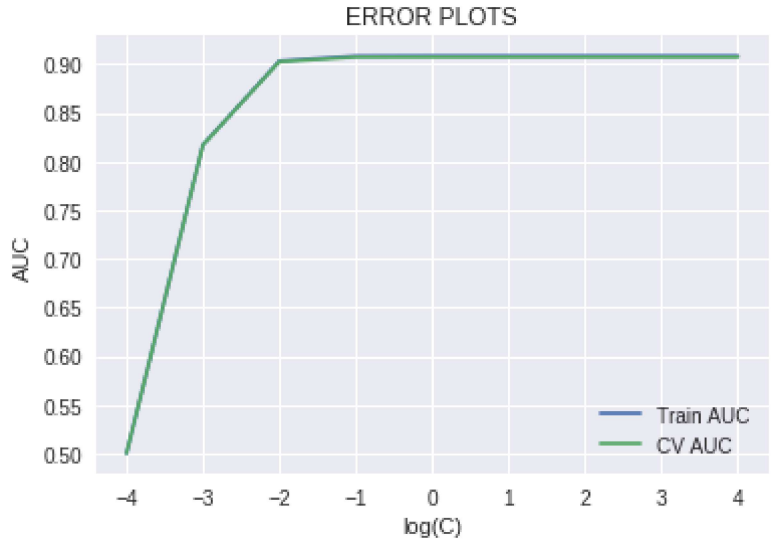
In [0]:
```
_train,X_test,y_train,y_test=train_test_split(list_of_sentance,y,test_size=0.3,random_state=0)
```

In [0]:
```
W2V model on train data
_train_w2v_model=Word2Vec(X_train,min_count=5,size=50,workers=4)
```

In [0]:
```
vectorizing X_train using Avg W2V
_train_w2v_words = list(X_train_w2v_model.wv.vocab)
_train_sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
or sent in tqdm(X_train): # for each review/sentence
   sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you use g
ogle'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 X_train_w2v_words:
           vec = X_train_w2v_model.wv[word]
           sent_vec += vec
           cnt_words += 1
   if cnt_words != 0:
       sent_vec /= cnt_words
   X_train_sent_vectors.append(sent_vec)

vectorizing X_test using Avg W2V
X_test_w2v_words = list(X_test_w2v_model.wv.vocab)
_test_sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
or sent in tqdm(X_test): # for each review/sentence
   sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you use g
ogle'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 X_train_w2v_words:
           vec = X_train_w2v_model.wv[word]
           sent_vec += vec
           cnt_words += 1
   if cnt_words != 0:
       sent_vec /= cnt_words
   X_test_sent_vectors.append(sent_vec)
```

```
100%|██████████| 61441/61441 [02:13<00:00, 459.92it/s]
100%|██████████| 26332/26332 [00:56<00:00, 463.79it/s]
```

In [0]:
```
rModel=LogisticRegression(penalty='l1')
yper_parameters={'C':[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]}
odel=GridSearchCV(lrModel,hyper_parameters,scoring='roc_auc',cv=3)
odel.fit(X_train_sent_vectors,y_train)

_values=[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]
rain_auc= model.cv_results_['mean_train_score']
v_auc = model.cv_results_['mean_test_score']

=np.log10(C_values)

lt.plot(C, train_auc, label='Train AUC')
 this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alpha_values,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

lt.plot(C, cv_auc, label='CV AUC')
 this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alpha_values,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
lt.legend()
lt.xlabel("log(C)")
lt.ylabel("AUC")
lt.title("ERROR PLOTS")
lt.show()
```

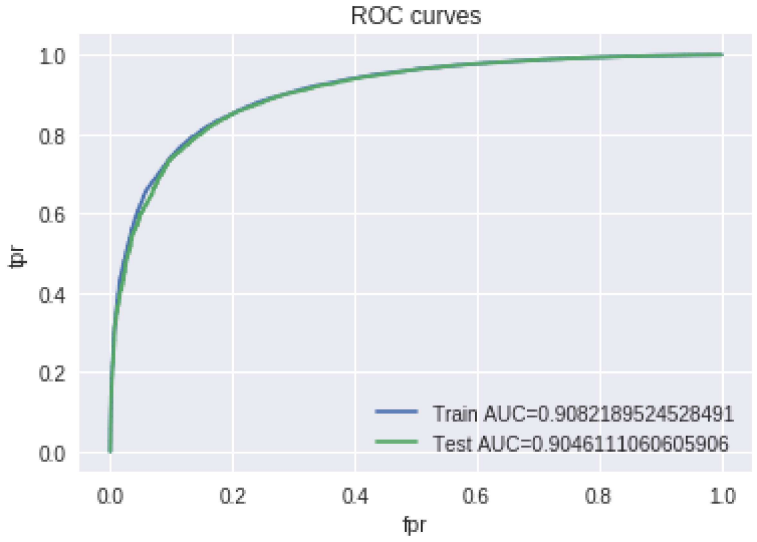


In [0]:
```
odel.best_estimator_
```

Out[0]:
```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

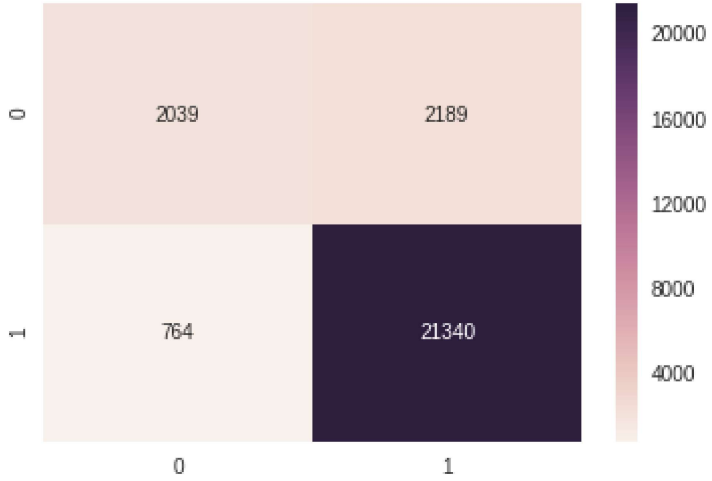
In [0]:
```
est_C=1
```

In [0]:
```
rModel=LogisticRegression(penalty='l1',C=best_C)
rModel.fit(X_train_sent_vectors,y_train)

rain_fpr,train_tpr,thresholds=roc_curve(y_train,lrModel.predict_proba(X_train_sent_vectors)[:,1])
est_fpr,test_tpr,thresholds=roc_curve(y_test,lrModel.predict_proba(X_test_sent_vectors)[:,1])

lt.plot(train_fpr,train_tpr,label='Train AUC='+str(auc(train_fpr,train_tpr)))
lt.plot(test_fpr,test_tpr,label='Test AUC='+str(auc(test_fpr,test_tpr)))
lt.legend()
lt.xlabel('fpr')
lt.ylabel('tpr')
lt.title('ROC curves')
lt.show()
```



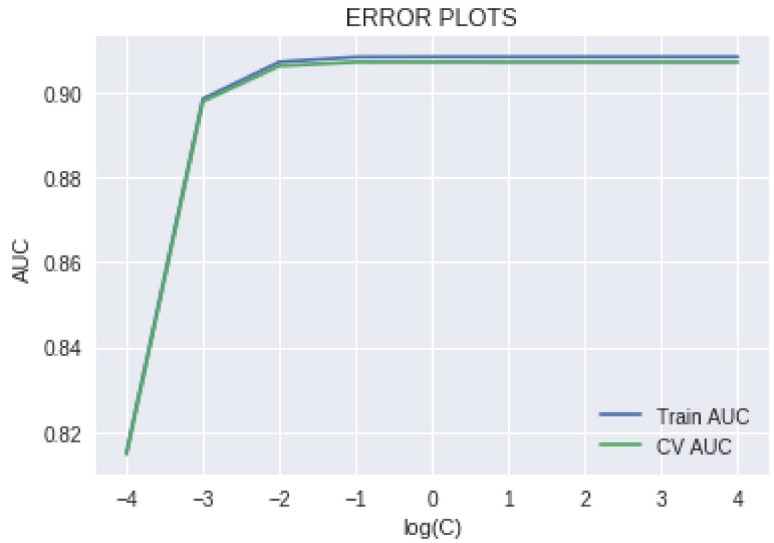
In [0]:
```
confusion matrix using seaborn heatmap

_pred=lrModel.predict(X_test_sent_vectors)
m=confusion_matrix(y_test,y_pred)
ns.heatmap(cm,annot=True,fmt='.5g')
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7f29c28c19b0>



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

In [0]:
```
rModel=LogisticRegression(penalty='l2')
yper_parameters={'C':[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]}
odel=GridSearchCV(lrModel,hyper_parameters,scoring='roc_auc',cv=3)
odel.fit(X_train_sent_vectors,y_train)

_values=[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]
rain_auc= model.cv_results_['mean_train_score']
v_auc = model.cv_results_['mean_test_score']

=np.log10(C_values)

lt.plot(C, train_auc, label='Train AUC')
 this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alpha_values,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

lt.plot(C, cv_auc, label='CV AUC')
 this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alpha_values,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
lt.legend()
lt.xlabel("log(C)")
lt.ylabel("AUC")
lt.title("ERROR PLOTS")
lt.show()
```

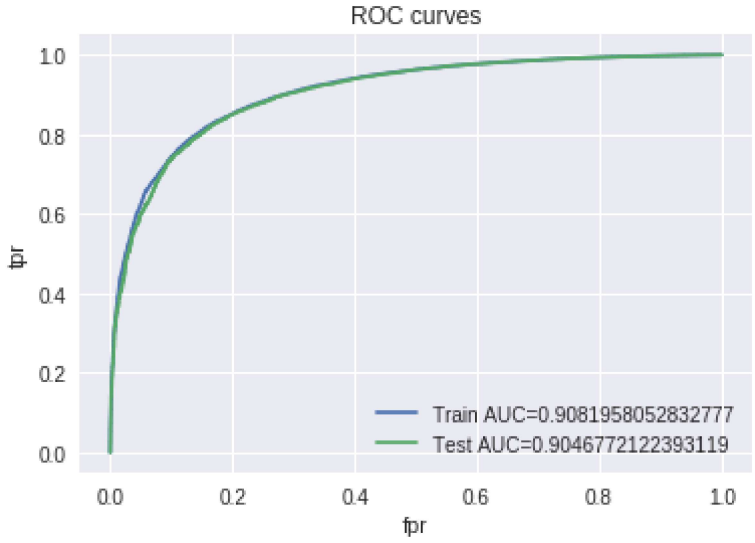


In [0]:
```
odel.best_estimator_
```

Out[0]:
```
LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

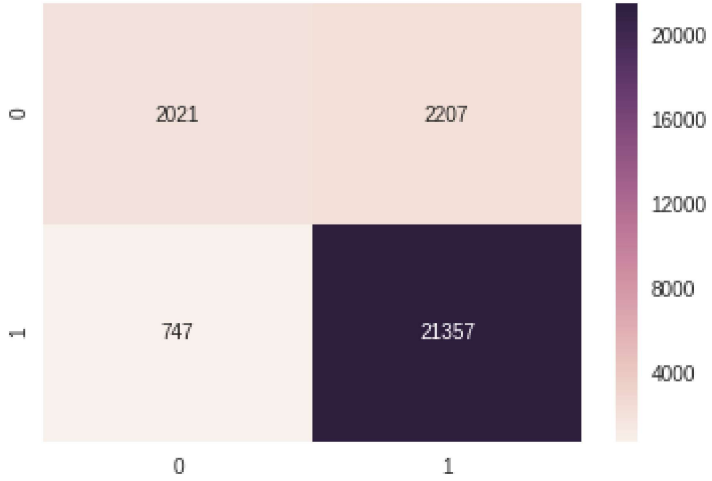
In [0]:
```
est_C=0.1
```

In [0]:
```
rModel=LogisticRegression(penalty='l2',C=best_C)
rModel.fit(X_train_sent_vectors,y_train)

rain_fpr,train_tpr,thresholds=roc_curve(y_train,lrModel.predict_proba(X_train_sent_vectors)[:,1])
est_fpr,test_tpr,thresholds=roc_curve(y_test,lrModel.predict_proba(X_test_sent_vectors)[:,1])

lt.plot(train_fpr,train_tpr,label='Train AUC='+str(auc(train_fpr,train_tpr)))
lt.plot(test_fpr,test_tpr,label='Test AUC='+str(auc(test_fpr,test_tpr)))
lt.legend()
lt.xlabel('fpr')
lt.ylabel('tpr')
lt.title('ROC curves')
lt.show()
```



In [0]:
```
confusion matrix using seaborn heatmap

_pred=lrModel.predict(X_test_sent_vectors)
m=confusion_matrix(y_test,y_pred)
ns.heatmap(cm,annot=True,fmt='.5g')
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7f29c27a8320>



[5.4] Logistic Regression on TFIDF W2V, SET 4

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

In [0]:
```
_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
```

In [0]:
```
odel=TfidfVectorizer()
odel.fit(X_train)
_train_tfidf=model.transform(X_train)
_test_tfidf=model.transform(X_test)
```

In [0]:
```
building a dictionary with features as keys and their corresponding idf's as values
ictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

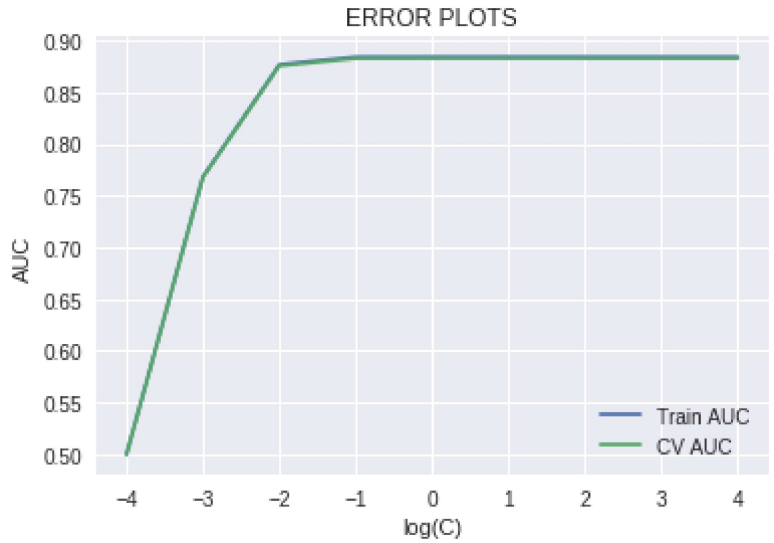
In [0]:
```
vectorizing X_train using tfidf w2v
_train_list_of_sents=[]
or sent in X_train:
 X_train_list_of_sents.append(sent.split())

build w2v model for X_train
_train_w2v_model=Word2Vec(X_train_list_of_sents,min_count=5,size=50,workers=4)
ords=list(X_train_w2v_model.wv.vocab)

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

_train_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
ow=0;
or sent in tqdm(X_train_list_of_sents): # for each review/sentence
   sent_vec = np.zeros(50) # as word vectors are of zero length
   weight_sum =0; # num of words with a valid vector in the sentence/review
   for word in sent: # for each word in a review/sentence
       if word in words and word in tfidf_feat:
           vec = X_train_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
   X_train_tfidf_sent_vectors.append(sent_vec)
   row += 1
```

100%|██████████| 61441/61441 [22:37<00:00, 45.25it/s]

In [0]:
```
vectorizing test dataset

_test_list_of_sents=[]
or sent in X_test:
 X_test_list_of_sents.append(sent.split())

build w2v model for X_test
X_test_w2v_model=Word2Vec(X_test_list_of_sents,min_count=5,size=50,workers=4)
X_test_words=list(X_test_w2v_model.wv.vocab)


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

_test_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
ow=0;
or sent in tqdm(X_test_list_of_sents): # for each review/sentence
   sent_vec = np.zeros(50) # as word vectors are of zero length
   weight_sum =0; # num of words with a valid vector in the sentence/review
   for word in sent: # for each word in a review/sentence
       if word in words and word in tfidf_feat:
           vec = X_train_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
   X_test_tfidf_sent_vectors.append(sent_vec)
   row += 1
```

```
100%|██████████| 26332/26332 [09:33<00:00, 45.91it/s]
```

In [0]:
```
rModel=LogisticRegression(penalty='l1')
yper_parameters={'C':[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]}
odel=GridSearchCV(lrModel,hyper_parameters,scoring='roc_auc',cv=3)
odel.fit(X_train_tfidf_sent_vectors,y_train)

_values=[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]
rain_auc= model.cv_results_['mean_train_score']
v_auc = model.cv_results_['mean_test_score']

=np.log10(C_values)

lt.plot(C, train_auc, label='Train AUC')
 this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alpha_values,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

lt.plot(C, cv_auc, label='CV AUC')
 this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alpha_values,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
lt.legend()
lt.xlabel("log(C)")
lt.ylabel("AUC")
lt.title("ERROR PLOTS")
lt.show()
```

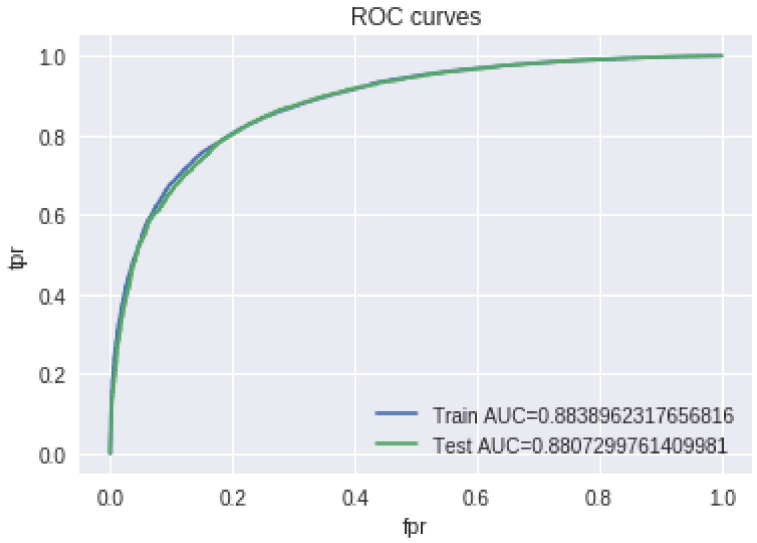


In [0]:
```
odel.best_estimator_
```

Out[0]:
```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

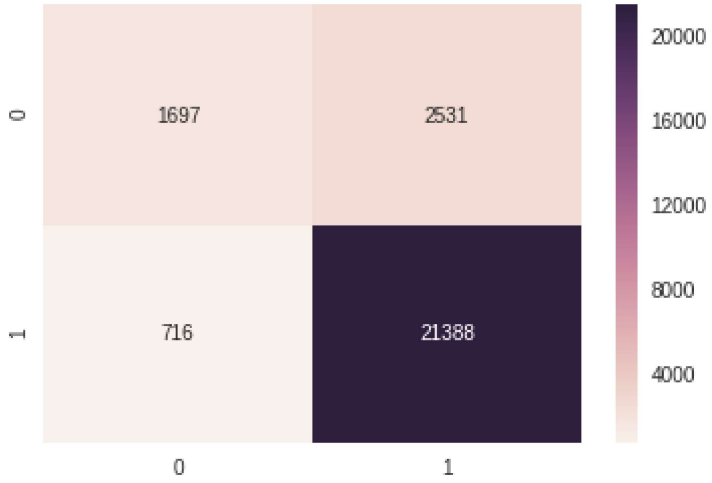
In [0]:
```
est_C=1
```

In [0]:
```
rModel=LogisticRegression(penalty='l1',C=best_C)
rModel.fit(X_train_tfidf_sent_vectors,y_train)

rain_fpr,train_tpr,thresholds=roc_curve(y_train,lrModel.predict_proba(X_train_tfidf_sent_vectors)[:,1])
est_fpr,test_tpr,thresholds=roc_curve(y_test,lrModel.predict_proba(X_test_tfidf_sent_vectors)[:,1])

lt.plot(train_fpr,train_tpr,label='Train AUC='+str(auc(train_fpr,train_tpr)))
lt.plot(test_fpr,test_tpr,label='Test AUC='+str(auc(test_fpr,test_tpr)))
lt.legend()
lt.xlabel('fpr')
lt.ylabel('tpr')
lt.title('ROC curves')
lt.show()
```



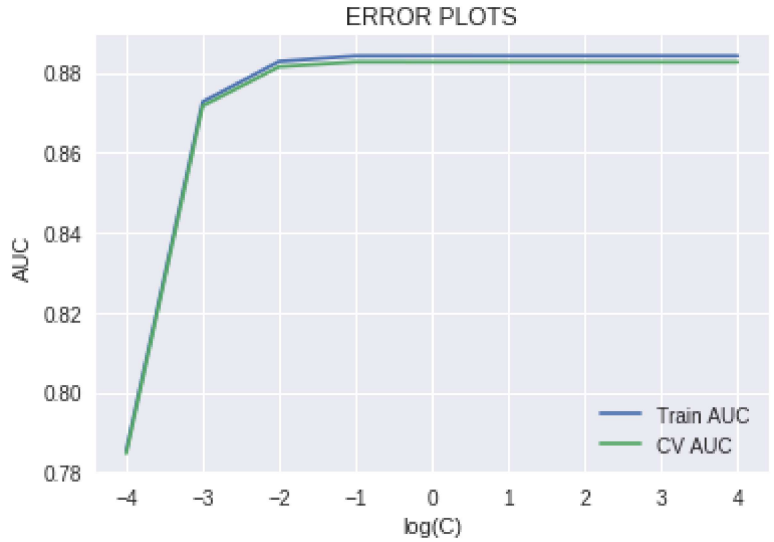
In [0]:
```
confusion matrix using seaborn heatmap

_pred=lrModel.predict(X_test_tfidf_sent_vectors)
m=confusion_matrix(y_test,y_pred)
ns.heatmap(cm,annot=True,fmt='.5g')
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff8e9dcacc0>



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

In [0]:
```
rModel=LogisticRegression(penalty='l2')
yper_parameters={'C':[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]}
odel=GridSearchCV(lrModel,hyper_parameters,scoring='roc_auc',cv=3)
odel.fit(X_train_tfidf_sent_vectors,y_train)

_values=[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]
rain_auc= model.cv_results_['mean_train_score']
v_auc = model.cv_results_['mean_test_score']

=np.log10(C_values)

lt.plot(C, train_auc, label='Train AUC')
 this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alpha_values,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

lt.plot(C, cv_auc, label='CV AUC')
 this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(alpha_values,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
lt.legend()
lt.xlabel("log(C)")
lt.ylabel("AUC")
lt.title("ERROR PLOTS")
lt.show()
```

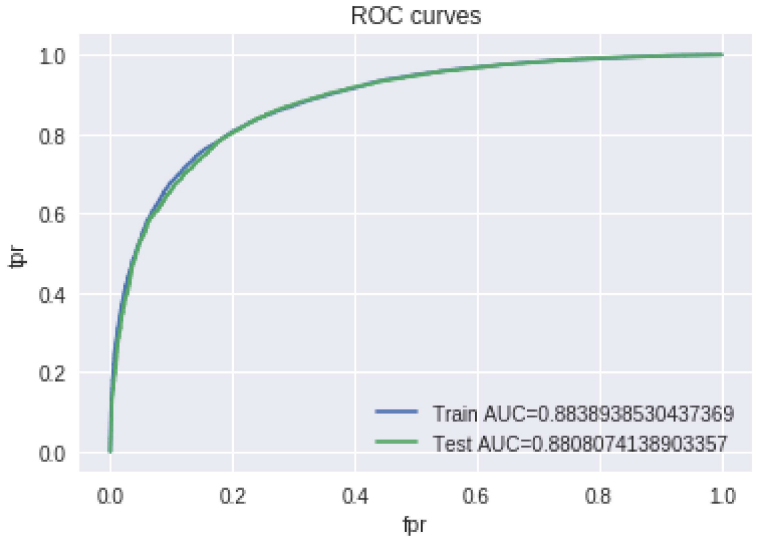


In [0]:
```
odel.best_estimator_
```

Out[0]:
```
LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

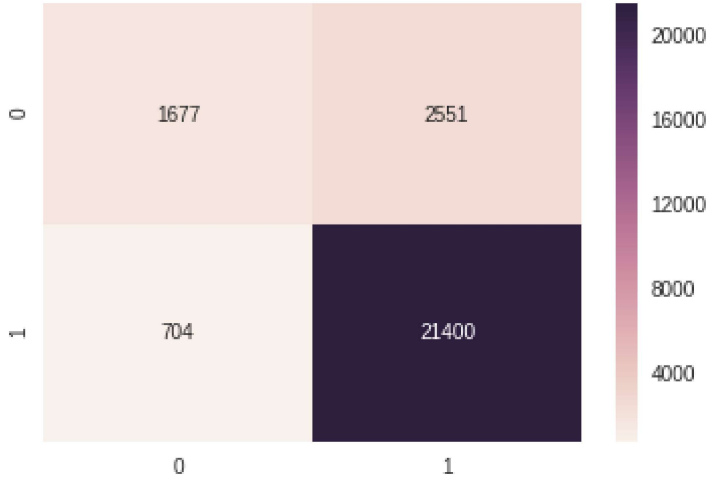
In [0]:
```
est_C=0.1
```

In [0]:
```
rModel=LogisticRegression(penalty='l2',C=best_C)
rModel.fit(X_train_tfidf_sent_vectors,y_train)

rain_fpr,train_tpr,thresholds=roc_curve(y_train,lrModel.predict_proba(X_train_tfidf_sent_vectors)[:,1])
est_fpr,test_tpr,thresholds=roc_curve(y_test,lrModel.predict_proba(X_test_tfidf_sent_vectors)[:,1])

lt.plot(train_fpr,train_tpr,label='Train AUC='+str(auc(train_fpr,train_tpr)))
lt.plot(test_fpr,test_tpr,label='Test AUC='+str(auc(test_fpr,test_tpr)))
lt.legend()
lt.xlabel('fpr')
lt.ylabel('tpr')
lt.title('ROC curves')
lt.show()
```



In [0]:
```
confusion matrix using seaborn heatmap

_pred=lrModel.predict(X_test_tfidf_sent_vectors)
m=confusion_matrix(y_test,y_pred)
ns.heatmap(cm,annot=True,fmt='.5g')
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff8ec2cbe48>



[6] Conclusions

In [125]:
```
http://zetcode.com/python/prettytable/
rom prettytable import PrettyTable

 = PrettyTable()

.field_names = ["Vectorizer", "Regularization", "Best C", "Test AUC"]

.add_row(["BOW", "L1", 1,0.93])
.add_row(["BOW", "L2", 0.1,0.94])
.add_row(["TFIDF", "L1", 1,0.95])
.add_row(["TFIDF", "L2", 10,0.96])
.add_row(["AvgW2V", "L1", 1,0.9])
.add_row(["AvgW2V", "L2", 0.1,0.9])
.add_row(["Tfidf W2V", "L1", 1,0.88])
.add_row(["Tfidf W2V", "L2", 0.1,0.88])


rint(x)
```

```
+------------+----------------+--------+----------+
| Vectorizer | Regularization | Best C | Test AUC |
+------------+----------------+--------+----------+
|    BOW     |       L1       |   1    |   0.93   |
|    BOW     |       L2       |  0.1   |   0.94   |
|   TFIDF    |       L1       |   1    |   0.95   |
|   TFIDF    |       L2       |   10   |   0.96   |
|   AvgW2V   |       L1       |   1    |   0.9    |
|   AvgW2V   |       L2       |  0.1   |   0.9    |
| Tfidf W2V  |       L1       |   1    |   0.88   |
| Tfidf W2V  |       L2       |  0.1   |   0.88   |
+------------+----------------+--------+----------+
```