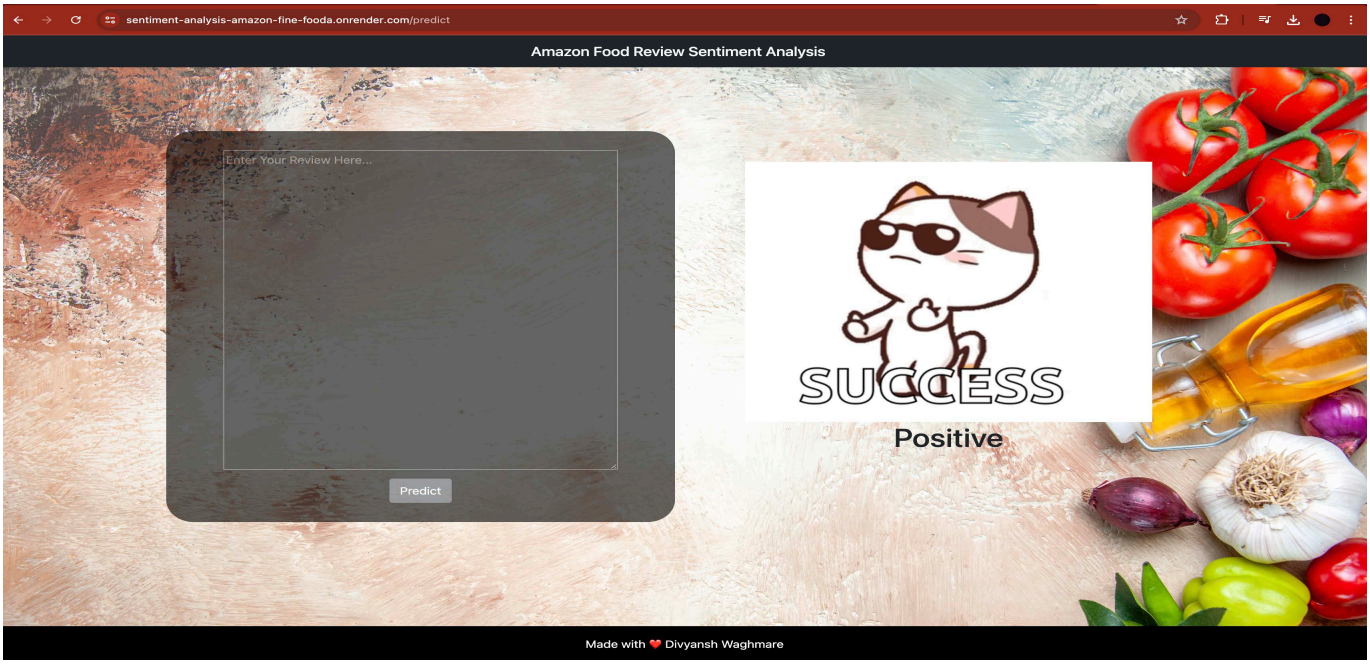
Sentiment Analysis on Amazon fine food review

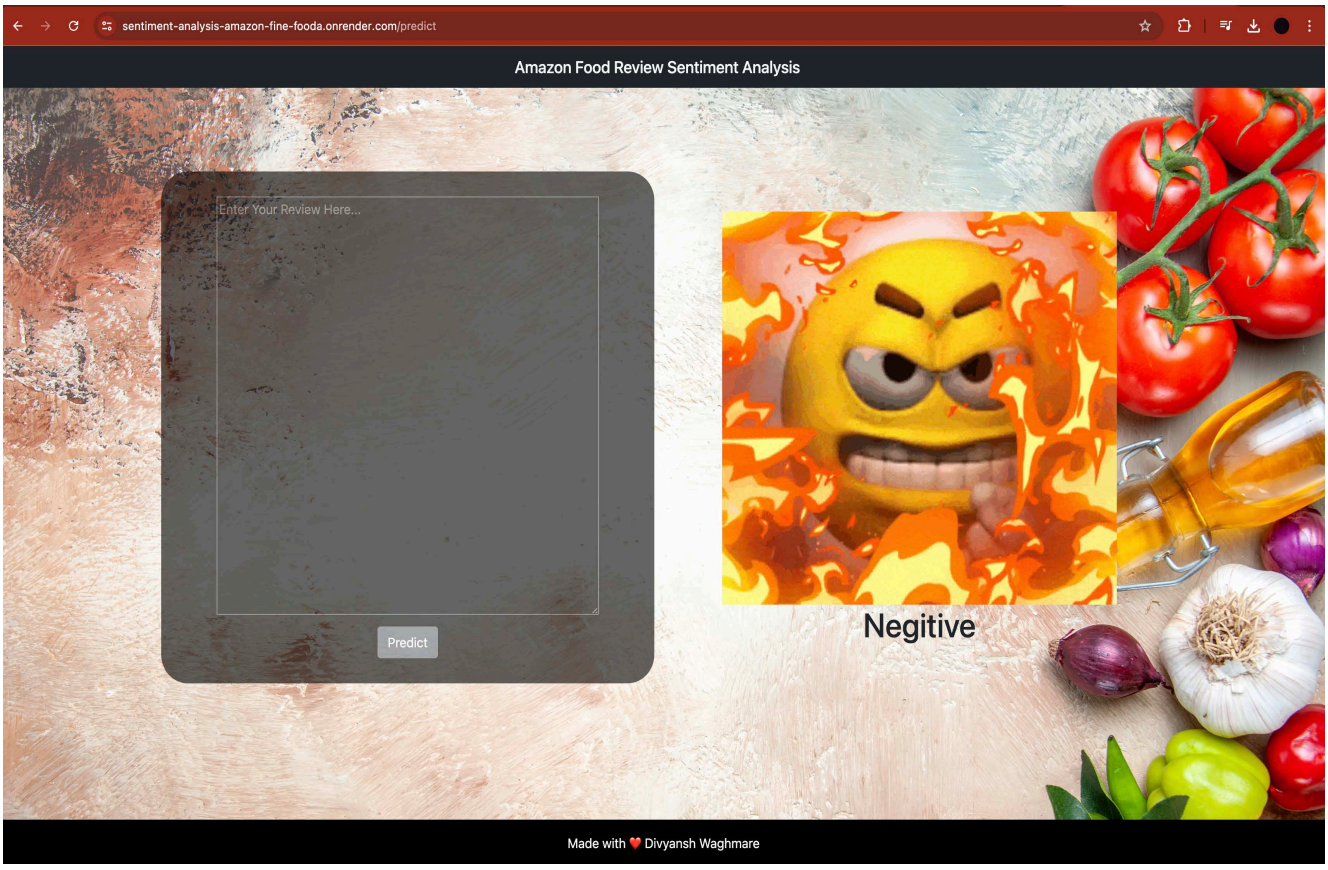
Divyansh Waghmare
project-ID -> PTID-CDS-JUN-24-1953

Github -> https://github.com/divyansh199/Sentiment_Analysis_Amazon_Fine_Fooda

When There is a Good Review



When There is a Bad Review



Sentiment Analysis of Food Reviews Using Score Feature

Introduction

Customer reviews are a valuable resource for businesses aiming to improve their products and services. Sentiment analysis helps in understanding the emotional tone behind customer reviews. This project leverages the Score feature in the provided dataset to categorise reviews directly into positive, negative, or neutral sentiments.

The main objective is to preprocess the review text, create target labels if required and build a sentiment analysis model. The insights obtained will help businesses enhance their products and services based on customer feedback

If the rating is **4 or 5** the it is consider as **Positive Sentiment** and if rating is **1 or 2** then **Negitive Sentiment** and if the rating is **3** then we consider it as **neutral** but will not consider it because we need to find the Polarity(Positive/Negitive) of the review

Attributes

Primary Features:

1. Id: Unique identifier for each review
2. ProductId: Unique identifier for the product
3. UserId: Unique identifier for the user
4. ProfileName: Profile name of the user
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

📥 Lets load our Dataset

In [44]:
```
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [45]:
```
food_review = pd.read_csv('Dataset.csv')
```

In [46]:
```
food_review.sample(50)
```

Out[46]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNu
402422	402423	B002VASD24	ABXMGGZSMQFM2	Ginger "me"	
81126	81127	B0040DWCXG	A1G69SJPXWKWO1	Barbiedoll12	
519885	519886	B000YSTIL0	A3MBU9VI7MB47E	DOc	
438641	438642	B005HGAV8I	AM45D0WU5C4EW	Natasha "sushirama"	
368285	368286	B001ZWHGFO	A2W5IUAKVQRMBF	C. Jensen "K9 Happy Gal"	
395294	395295	B000H7ELTW	AL94DPPY619AM	Carl R. Hinkelman Jr.	
417156	417157	B00112ILZM	ATC610DZTJMH0	Diana	
534843	534844	B0018MR53Y	A2I4ZWBD6W561R	Claudia	
97831	97832	B003GSVSNQ	A2VB3MBWER8QMW	KO	
377309	377310	B004XZMVZK	A2Y08EFRMEFTHD	Cindy Williams "stormloverf5"	
104446	104447	B000YSRK7E	A165MAM6TS892W	fpl	
329085	329086	B003VXHGE6	A1HGXLP6WATS4B	F. D. Gillett "Comparison Shopper"	

	Id	ProductId	UserId	ProfileName	HelpfulnessNu
173679	173680	B00112JV60	ARZCOSCV0HUNB	ashley chen	
61574	61575	B001EO5R0Y	A2OOGNEEW60IQ	M. Bolin	
53935	53936	B003HI2Z6O	A28JP9DX106FQD	I. D. Barbosa "one million dollar smile"	
320265	320266	B003Z6W32E	A29RZZ8DXRCPJ0	E. S. Noli "dzign17"	
46655	46656	B001E530IE	A242Q6K6OMF94D	Annette Weatherman "Rock Around the World"	
141118	141119	B000BNVEDI	A21N5OVH997NA7	Diana	
290210	290211	B000IZ0OC6	A3N4DKNVIOU1G9	Judy A. Bush	
460443	460444	B001CWU9E2	A3QGPFUH2D4S4S	S. Tucker	
165355	165356	B001HWCF4O	A2RNYW2HIQBGSM	Carlos Alvarez	
116033	116034	B0026BQKT6	A1UMQSNGIUMF7C	J. Charchenko "JC"	
508395	508396	B0029JZMA8	A14F7MXMOGCOR8	cehsr55	
288220	288221	B000ENUC3S	AYGJ96W5KQMUJ	SJP	

	Id	ProductId	UserId	ProfileName	HelpfulnessNu
343421	343422	B00004RBDU	ANICAIMW2SF22	Sheila	
306444	306445	B001BOQ3QY	A93OVQE7171W0	GSrepo	
411920	411921	B000CQC05U	AD4MFIC8E2PZB	Cece	
526313	526314	B003VMW0ZC	AKBZD4VMJ1FF4	astroqueen67	
420745	420746	B0040PUUKQ	A19636OLBGELAG	Spatter	
541754	541755	B001EQ52P2	A2L01339XV496V	Patricia Smith	
138184	138185	B000YU0ONE	A3HD4EZXQ6VK0Q	SurreallyReal	
343000	343001	B0012C7VLG	ASLFT5EJ8XWRL	toreyp123	
269331	269332	B0027ZAMXQ	A22YSFQL5OHRSC	jojo	
155760	155761	B0009VO58S	A2YVWIU1B8HJ8	AngelE	
47358	47359	B001EPQVFS	A3PQY16BQSGC3	Karthik	
447114	447115	B00473VGHW	ATZTIU71681SO	Ria Darling	

	Id	ProductId	UserId	ProfileName	HelpfulnessNu
453348	453349	B0029XLH4Y	A1XM83D7NKPOVY	Marnee L. Larson "river7495"	
168608	168609	B0001ES9F8	A4AF016IPVEYY	John Mazzone "smazzone"	
176557	176558	B002E0KW1Q	A30RQO7CIU0HBF	Jodie Walker	
125158	125159	B002MBKF0U	A21KR27KC5JLFK	Sam	
385994	385995	B000GG1O6W	A15825UTP3XHSE	Rita Walls	
106281	106282	B00264TMWU	A1NLP7U3ZOPTT7	Sally Robertson	
383097	383098	B000RJ4ZK0	A3BII2BT4KNZAI	Angela	
535925	535926	B003VXHGDM	A121VLJBL8T0H1	David Rupert	
13191	13192	B002TMV34E	A11FFLD0GV82CQ	M. Boone	
1092	1093	B0025ULYKI	A2MZ43MMIBFL7M	John	
89144	89145	B001E5E1MI	A18R6HLQYSIVP2	gayle	
366867	366868	B002W9JILE	A3SYKD35HCZGVS	Ken H	

	Id	ProductId	UserId	ProfileName	HelpfulnessNu
533004	533005	B009E7YC54	AYFLML14EPVQV	GRANNY BRAD	
187220	187221	B004S036FO	A3FKRI95UOQ9MU	Jordan	

🔎 here we can see that data has so many cleanlliness issue

Cleaning The Dataset

Basic Data Analysis

In [50]:
```
food_review.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 568454 entries, 0 to 568453
Data columns (total 10 columns):
 #   Column                  Non-Null Count   Dtype 
---  ------                  --------------   ----- 
 0   Id                      568454 non-null  int64 
 1   ProductId               568454 non-null  object
 2   UserId                  568454 non-null  object
 3   ProfileName             568428 non-null  object
 4   HelpfulnessNumerator    568454 non-null  int64 
 5   HelpfulnessDenominator  568454 non-null  int64 
 6   Score                   568454 non-null  int64 
 7   Time                    568454 non-null  int64 
 8   Summary                 568427 non-null  object
 9   Text                    568454 non-null  object
dtypes: int64(5), object(5)
memory usage: 43.4+ MB
```

🔎 **Time** should be in datetime format

In [52]:
```
food_review.isnull().sum()
```

Out[52]:
```
Id                      0
ProductId               0
UserId                  0
ProfileName            26
HelpfulnessNumerator    0
HelpfulnessDenominator  0
Score                   0
Time                    0
Summary                27
Text                    0
dtype: int64
```

🔎 Here we can see that there are null values in **ProfileName** and **Summary**

In [54]:
```
food_review.describe(include = 'O')
```

Out[54]:

	ProductId	UserId	ProfileName	Summary	Text
count	568454	568454	568428	568427	568454
unique	74258	256059	218415	295742	393579
top	B007JFMH8M	A3OXHLG6DIBRW8	C. F. Hill "CFH"	Delicious!	This review will make me sound really stupid, ...
freq	913	448	451	2462	199

🔎 Here we can see that there are so many duplicate entries

In [56]:
```
food_review['Text'].sample(50)
```

Out[56]:
```
486293    These artichoke "hearts" contain too much leaf...
334622    I ordered these lollipops to go on top of a ca...
72523     The recipient enjoys having the snacks availab...
387627    If you're buying these coffee pods for use in ...
324626    I purchased the Caramel Apple Pops to share at...
46948     I bought the dreamfields lasagna noodles becau...
45231     Excellent nuts.  hard to find in stores.  grea...
384308    So hard to get in the United States, it was a ...
403996    Some mornings I just don't have time to make m...
477102    This pack of yeast is an amazing deal! It has ...
568415                      as it should be for $6 a bottle.
487815    Our Terrier is super picky but he always finis...
408183    I purchased this product both for my dog and m...
172587    We agree with the other reviews of Quaker Oatm...
37626     I've yet to run into a product where, after sa...
60159     This same product was almost $13 for a 16 oz. ...
417164    Okay, so it isn't the best tasting thing in th...
349036    Snapea Crisps are a splendid snack for husband...
109904    My husband loves this brew! Not too weak, thou...
538144    I was first introduced to this coffee at a res...
113048    Salt & vinegar chips are my favorite flavor so...
213701    They are great if you are on a diet and cant s...
11100     Ingredients on Deep River Snacks's website are...
521701    I like most varieties of Primal Strips. They a...
381041    If you want non-sticky rice, buy Californian r...
410496    I bought a box of these based on the rave revi...
24837     Four stars due to price only. Otherwise, it wo...
536099    I was intrigued how this particular blend got ...
557344    we only want the watermelon jollies and cannot...
313913    Not too spicy, just right!  Great on tacos, sa...
26373     This is the dried sap from coconut blossoms.  ...
275762    Kids love them for snacks at school..and it ge...
82470     This product allows individuals who are restri...
47959     First, let me note that there are several diff...
144682    Unlike some other brands, these chocolate cove...
282047    I tried this even though I had previously not ...
125       One of my cats is allergic to fish and beef. T...
496222    I ordered Kind bars after paying ridiculously ...
299005    I have been trying to find this product at maj...
168286    Vine is such a great innovation to get one to ...
385073    Brewing Machine: Philips Senseo<br /><br />Fir...
214498    No complaints about the chips, they're great. ...
304594    From the first bite to the last, these cran-or...
350125    Great tasting fat free hot chocolate!  The pac...
281324    I think this product is GREAT! I mix 2 tsp wit...
204228    I was introduced to this dog food over tens ye...
200154    My husky, a chewing machine, LOVED the flavor ...
179277    SO WORTH IT.  And when you do the math, it's a...
411175    My family has been pleased with these apples. ...
350026    I got my two very tiny boxes of rock hard "jel...
Name: Text, dtype: object
```

🔎 Here we can see that there is so much dirty data in text attribute

1. URL are present
2. html tags are present
3. stopword also present
4. Lots of Phrases in the data like can't,wont't.....etc

📝 Let's list down all the cleanliness issues

1. **Time** wrong datatype
2. **ProfileName** contains 26 and **Summary** contains 27 **Null** values
3. There is **Duplicacy** in the dataset
4. **Text** contain **URL**, **HTML**, **stopwords**, **Phrases**

🧹 Rectifing Cleanliness issuess

In [61]:
```
df_review = food_review.copy()
```

we will convert Time into datetime

In [63]:
```
df_review['Date'] = pd.to_datetime(df_review['Time'], unit = 's')
```

In [64]:
```
df_review
```

Out[64]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	
...	...	...	...	...	
568449	568450	B001EO7N10	A28KG5XORO54AY	Lettie D. Carter	
568450	568451	B003S1WTCU	A3I8AFVPEE8KI5	R. Sawyer	
568451	568452	B004I613EE	A121AA1GQV751Z	pksd "pk_007"	
568452	568453	B004I613EE	A3IBEVCTXKNOH	Kathy A. Welch "katwel"	
568453	568454	B001LR2CU2	A3LGQPJCZVL9UC	srfell17	

568454 rows × 11 columns

Checking the null values

In [66]:
```
df_review.loc[df_review['ProfileName'].isnull() == True , ['ProfileName']
```

Out[66]:

	ProfileName
10616	NaN
25509	NaN
38874	NaN
47923	NaN
49800	NaN
67077	NaN
106550	NaN
121819	NaN
125452	NaN
137613	NaN
163191	NaN
172462	NaN
211846	NaN
268648	NaN
297275	NaN
306751	NaN
331647	NaN
358674	NaN
431598	NaN
440825	NaN
461668	NaN
490412	NaN
491728	NaN
515436	NaN
517676	NaN
560446	NaN

Lets replace the NaN with Unknown in ProfileName

In [68]:
```
df_review.loc[df_review['ProfileName'].isnull() == True , ['ProfileName']
```

In [69]:
```
df_review['ProfileName'].isnull().sum()
```

Out[69]: 0

In [70]:
```
pd.set_option('display.max_colwidth', None)
```

In [71]:
```
df_review.loc[df_review['Summary'].isnull() == True , ['Text']]
```

Out[71]:

	Text
33958	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
40548	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
101106	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
102979	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
117515	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
155712	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
178290	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
198474	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
212691	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
237565	This is a cool system only problem is that the coffee is no longer hot once it drains into your cup. If you are using it to make ice coffee though, it's great!
293906	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
299495	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
300961	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
333556	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
352043	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
357215	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
357814	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
360782	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
379473	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
380558	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
381313	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
386283	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.

	Text
392529	I like the product and tried it before I puchased it. However, one of the boxes arrived have only 10 in it instead of 12.
484367	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
486640	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
503260	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
530716	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.

we can see that the place where null value is present in the summary has maximum duplicate values in the Text attribute

removing the duplicates

In [74]:
```
df_review[df_review.duplicated(subset = ['UserId','ProfileName','Time','T
```

Out[74]: (232415, 11)

there are 232415 rows are duplicate

In [76]:
```
df_review.drop_duplicates(subset = ['UserId','ProfileName','Time','Text']
```

now check the null values

In [78]:
```
df_review.shape
```

Out[78]: (393933, 11)

In [79]:
```
df_review.loc[df_review['Summary'].isnull() == True , ['Text']]
```

Out[79]:

	Text
31845	I only used two maybe three tea bags and got pregnant - can not drink during pregnancy. Not a bad taste, but I'm not a big tea fan either.
188426	This is a cool system only problem is that the coffee is no longer hot once it drains into your cup. If you are using it to make ice coffee though, it's great!
286451	I like the product and tried it before I puchased it. However, one of the boxes arrived have only 10 in it instead of 12.

we can put summary by manaully in the data set for these three rows

In [81]:
```
df_review.loc[31845,['Summary']] = 'Not so good not so bad'
df_review.loc[188426,['Summary']] = 'good for ice coffee problem with hot
df_review.loc[286451,['Summary']] = 'quantity issue'
```

In [82]:
```
df_review['Summary'].isnull().sum()
```

Out[82]: 0

Now we will handel the Text data

Important libreries

In [85]:
```
# pip install nltk
```

In [86]:
```
#pip install textblob
```

In [87]:
```
# pip install tqdm
```

In [88]:
```
import string
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from bs4 import BeautifulSoup
from textblob import TextBlob
from tqdm import tqdm
```

In [89]:
```
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/divyansh/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [90]:
```
print(stop_words)
```

```
{'not', 'their', "mustn't", 'haven', 'out', 'both', 'from', "it's", 'all',
'aren', 'weren', 'with', 'an', 'wouldn', 'own', 'm', 'has', 'theirs', "tha
t'll", 'on', 'should', "won't", 'each', 've', 'there', "isn't", 'does', 'f
urther', 'being', 'who', "you've", 'yourselves', 'wasn', 'won', 'your', 't
hose', 'been', 'or', 'hers', "didn't", 'while', 'doesn', "shan't", 'his',
'our', 'do', 'no', 'these', 'did', 'he', 'here', 'and', 'this', 'where',
'himself', 'yourself', "she's", 'some', 'me', 'will', 'have', 'nor', 're',
'above', 'i', "haven't", 'too', 'as', 'just', 't', 'against', "aren't", 'u
p', "you're", 'the', 'any', 'myself', 'them', 'was', 'her', 'into', 'sha
n', "wouldn't", "couldn't", "hadn't", "shouldn't", 'she', 'my', "you'd",
'are', 'is', 'for', 'of', 'they', 'shouldn', 'ain', 'once', "mightn't", 'o
urselves', 'under', 'we', 'but', 'during', 'when', 'couldn', "hasn't", "we
ren't", 'how', 'were', 'about', 'few', 'more', "you'll", 'off', 'it', 'you
rs', 'be', 'between', 'didn', 'down', 'had', 'what', 'll', 'that', 'now',
"doesn't", 'in', 'mightn', "needn't", "don't", 'whom', 'a', 'to', "shoul
d've", 'hadn', 'don', 'by', 'hasn', 'such', 'only', 'you', 'themselves',
'after', 'why', 'o', 'most', 'through', 'can', 'mustn', 'ma', 'over', 's',
'having', 'below', 'herself', 'itself', 'than', 'am', 'very', 'ours', 'hi
m', 'so', 'd', 'if', 'before', 'its', 'then', 'other', "wasn't", 'which',
'same', 'until', 'because', 'isn', 'needn', 'at', 'y', 'doing', 'again'}
```

In [91]:
```
# some phrases to be convert
def expanded(phrase):
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    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

clean_reviews = []

# tqdm is for printing the status bar
for document in tqdm(df_review['Text'].values):
    # removing urls
    document = re.sub(r"http\S+", "",document)
    # removing html tags
    document = BeautifulSoup(document, 'lxml').get_text()
    # expanding phrases
    document = expanded(document)
    # removing extra spaces
    document = re.sub("\S*\d\S*","",document).strip()
    # spelling mistake
    # document = TextBlob(document)
    # document = document.correct().string
    # removing non alphabate
    document = re.sub('[^A-Za-z]+', ' ',document )
    # removing stopwords
    document =  ' '.join(i.lower() for i in document.split() if i.lower()

    clean_reviews.append(document.strip())
```

```
100%|██████████████████████████████████| 393933/393933 [00:30<00:00, 12820.3
0it/s]
```

In [92]:
```
a = TextBlob(clean_reviews[1])
a = a.correct().string
```

In [93]:
```
a
```

Out[93]:
```
'product arrived labelled lumbo halted peanuts peanuts actually small si
zed insulted sure error vendor intended represent product lumbo'
```

In [94]:
```
df_review['clean_text'] = clean_reviews
```

In [95]:
```
#### now we convert score to our target variable by coverting the points
```

In [96]:
```
df_review['Sentiment'] = df_review['Score'].apply(lambda x: 1 if x > 3 el
```

In [97]:
```
df_review['Sentiment'].unique()
```

Out[97]:
```
array([1, 0])
```

In [98]:
```
#### drop column Text
```

In [99]:
```
df_review.drop(columns = 'Text', axis = 1 , inplace =True)
```

In [100…
```
df_review
```

Out[100...

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

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	
...	...	...	...	...	
393928	568450	B001EO7N10	A28KG5XORO54AY	Lettie D. Carter	
393929	568451	B003S1WTCU	A3I8AFVPEE8KI5	R. Sawyer	
393930	568452	B004I613EE	A121AA1GQV751Z	pksd "pk_007"	

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
393931	568453	B004I613EE	A3IBEVCTXKNOH	Kathy A. Welch "katwel"	
393932	568454	B001LR2CU2	A3LGQPJCZVL9UC	srfell17	

393933 rows × 12 columns

We had performed the Data cleaning part now we will perform the EDA part

1. Analyzing Review with Time

In [103…
```
df = df_review[['Score', 'Date', 'clean_text']]
```

In [104…
```
df['Date'] = pd.to_datetime(df['Date'],unit='s')
df['Date'] = df['Date'].dt.to_period('M')
```

In [105…
```
df['Date']
```

Out[105…
```
0         2011-04
1         2012-09
2         2008-08
3         2011-06
4         2012-10
           ...
393928    2011-03
393929    2012-03
393930    2012-02
393931    2012-03
393932    2012-05
Name: Date, Length: 393933, dtype: period[M]
```

In [106…
```
df = df.sort_values(by = 'Date').reset_index(drop = True)
```

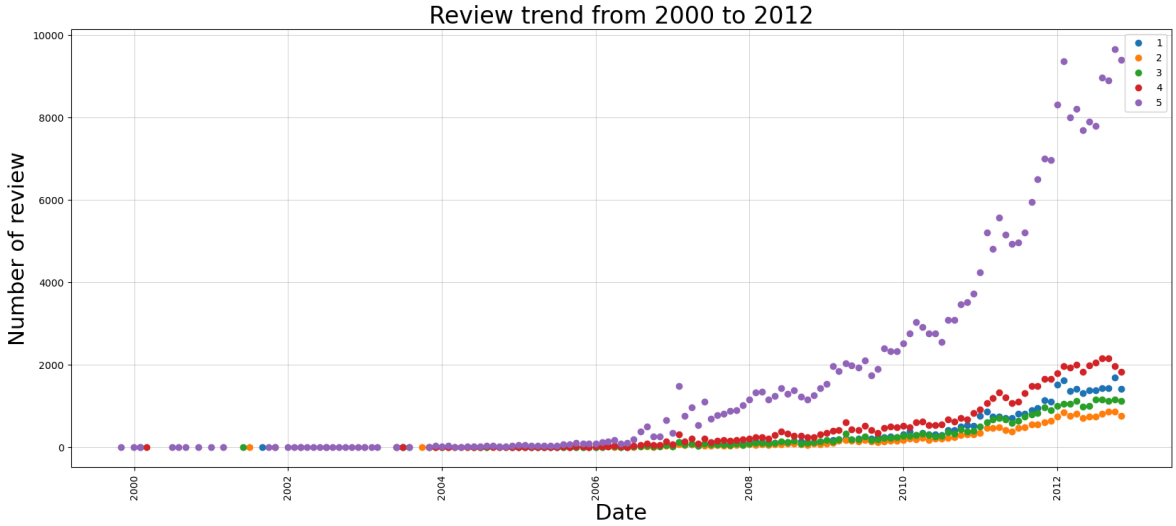
In [107…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 393933 entries, 0 to 393932
Data columns (total 3 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   Score       393933 non-null  int64
 1   Date        393933 non-null  period[M]
 2   clean_text  393933 non-null  object
dtypes: int64(1), object(1), period[M](1)
memory usage: 9.0+ MB
```

In [108…
```
score_1 = df[df['Score'] == 1]
score_2 = df[df['Score'] == 2]
score_3 = df[df['Score'] == 3]
score_4 = df[df['Score'] == 4]
score_5 = df[df['Score'] == 5]
```

In [109…
```
df_1 = score_1.groupby('Date')['Score'].count().reset_index()
df_2 = score_2.groupby('Date')['Score'].count().reset_index()
df_3 = score_3.groupby('Date')['Score'].count().reset_index()
df_4 = score_4.groupby('Date')['Score'].count().reset_index()
df_5 = score_5.groupby('Date')['Score'].count().reset_index()
```

In [110…
```
plt.figure(figsize=(20,8))
plt.plot_date(x = df_1['Date'], y = df_1['Score'], label = '1')
plt.plot_date(x = df_2['Date'], y = df_2['Score'], label = '2')
plt.plot_date(x = df_3['Date'], y = df_3['Score'], label = '3')
plt.plot_date(x = df_4['Date'], y = df_4['Score'], label = '4')
plt.plot_date(x = df_5['Date'], y = df_5['Score'], label = '5')
plt.grid(linewidth=0.5,alpha=0.75)
plt.xticks(rotation=90)
plt.xlabel('Date',fontsize=22)
plt.ylabel('Number of review',fontsize=22)
plt.title('Review trend from 2000 to 2012',fontsize=24);
# plt.savefig('review_trend.png')
plt.legend()
plt.show()
```



🔎 here we can see that as the Time inceases score and the number of reviews are also increases

In [112…
```
df_review
```

Out[112...

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

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	
...	...	...	...	...	
393928	568450	B001EO7N10	A28KG5XORO54AY	Lettie D. Carter	
393929	568451	B003S1WTCU	A3I8AFVPEE8KI5	R. Sawyer	
393930	568452	B004I613EE	A121AA1GQV751Z	pksd "pk_007"	

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
393931	568453	B004I613EE	A3IBEVCTXKNOH	Kathy A. Welch "katwel"	
393932	568454	B001LR2CU2	A3LGQPJCZVL9UC	srfell17	

393933 rows × 12 columns

Lets Analyze **HelpfulnessNumerator** and **HelpfulnessDenominator**

In [114…
```
df_review[df_review["HelpfulnessNumerator"] > df_review["HelpfulnessDenom
```

Out[114...

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerato
41832	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3
58539	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator

🔎 There is only 2 reviews in which **HelpfulnessNumerator** exceeds **HelpfulnessDenominator** which is not justifing the reviews

Lets create a common column for **HelpfulnessNumerator** and **HelpfulnessDenominator** named as Helpfulness_per , in which we calculate the percentage of helpfulness

In [117…
```
df_review['Helpfulness_Per'] = df_review[ ['HelpfulnessNumerator' , 'Help
```

In [118…
```
df_review['Helpfulness_Per'].unique()
```

Out[118…

```
array([100.        ,   0.        ,  80.        ,  50.        ,
        66.66666667,  25.        ,  89.47368421,  83.33333333,
        75.        ,  33.33333333,  30.        ,  11.11111111,
        42.85714286,  87.5       ,  85.71428571,  20.        ,
        26.31578947,  60.        ,  71.42857143,  53.84615385,
        57.14285714,  91.4893617 ,  86.66666667,  82.35294118,
        78.57142857,  74.07407407,  40.        ,  37.5       ,
        28.57142857,  14.28571429,  77.77777778,  12.5       ,
        90.        ,  94.11764706,  92.30769231,  70.        ,
        45.45454545,  88.88888889,  83.87096774,  90.47619048,
        92.85714286,  90.90909091,  91.66666667,  84.61538462,
        10.52631579,  98.21428571,  97.82608696,  75.18796992,
        31.25      ,  10.        ,  18.51851852,  88.        ,
        69.23076923,  62.5       ,  54.54545455,  41.66666667,
        45.83333333,  22.22222222,  81.81818182,  81.25      ,
        16.66666667,  93.10344828,  88.23529412,  23.52941176,
        63.63636364,  81.48148148,  95.65217391,  64.28571429,
        58.33333333,  94.44444444,  92.1875    ,  86.57407407,
        96.        ,  91.30434783,  64.70588235,  95.83333333,
         9.09090909,  13.33333333,  52.94117647,  96.96969697,
        36.36363636,   7.14285714,  72.72727273,  18.18181818,
        96.66666667,  99.07407407,  97.2972973 ,  80.64516129,
        64.1025641 ,  55.55555556,  43.75      ,  76.92307692,
        28.        ,  15.38461538,  44.44444444,  56.25      ,
        53.33333333,  47.05882353,  47.22222222,  23.07692308,
        25.92592593,  98.87640449,  88.37209302,  19.04761905,
        94.59459459,  84.31372549,  96.62921348,  72.22222222,
         5.88235294,  27.27272727,  97.95918367,  26.66666667,
        30.76923077,  94.73684211,  27.77777778,  68.75      ,
        92.        ,  90.56603774,  95.        ,  93.75      ,
        91.37931034,  82.85714286,  86.36363636,  85.        ,
        96.42857143,  95.23809524,   8.33333333,  97.56097561,
        93.33333333,  46.66666667,  96.15384615,  24.        ,
        92.68292683,  93.5483871 ,  86.95652174,   6.66666667,
        98.46153846,  97.        ,  97.61904762,  92.5       ,
        88.46153846,  61.53846154,   9.375     ,  79.16666667,
        70.58823529,  45.        ,  93.93939394,  90.32258065,
        68.        ,  95.45454545,   4.16666667,  89.65517241,
        88.57142857,  38.46153846,   7.69230769,  12.12121212,
        92.23744292,  92.15686275,  36.58536585,  88.0952381 ,
        84.        ,  61.9047619 ,  96.12903226,  96.38554217,
        90.58823529,  87.87878788,   5.55555556,  80.95238095,
        20.68965517,   7.40740741,  35.        ,  77.27272727,
        91.42857143,   4.54545455,  70.83333333,  73.33333333,
        93.65079365,  86.71875   ,  75.94936709,  65.95744681,
        57.69230769,  41.17647059,  40.90909091,  34.69387755,
        30.26315789,  16.17647059,  65.        ,  96.2962963 ,
        96.80851064,  94.91525424,  98.29059829,  98.93617021,
        95.74468085,  96.26865672,  98.30508475,  61.11111111,
        59.18367347,  98.91304348,  98.80952381,  92.98245614,
        78.94736842,  75.75757576,  76.47058824,  82.60869565,
        96.49122807,  84.50704225,  98.41269841,  96.55172414,
        87.34177215,  73.91304348,  70.37037037,  98.88888889,
        78.26086957,  17.64705882,  96.22641509,  94.33962264,
        97.05882353,  57.89473684,  47.36842105,  51.06382979,
        97.77777778,  92.35294118,  78.37837838,  97.6744186 ,
        35.71428571,  94.80519481,  94.28571429,  86.53846154,
        43.47826087,  99.18699187,  86.2745098 ,  97.14285714,
        98.48484848,  73.07692308,  68.18181818,  63.33333333,
        64.58333333,  96.77419355,   5.26315789,  36.84210526,
```

```
 82.92682927,  92.04545455,  34.7826087 ,  85.36585366,
 91.80327869,  97.22222222,  46.15384615,  21.73913043,
 82.05128205,  29.03225806,  95.75471698,  91.17647059,
  4.76190476,  65.71428571,  13.63636364,  77.14285714,
 95.3125    ,  92.59259259,   8.62068966,  80.55555556,
 20.51282051,  29.41176471,  99.25187032,  98.5645933 ,
 99.25373134,  80.48780488,  82.14285714,  76.        ,
 21.42857143,  31.91489362,   2.7027027 ,  20.83333333,
 92.10526316,  78.125     ,  61.29032258,  97.43589744,
  7.89473684,  72.4137931 ,   3.125     ,  68.42105263,
 97.97979798,  38.88888889,  97.5       ,  80.76923077,
  6.06060606,  93.02325581,  97.26027397,  90.76923077,
 31.37254902,  15.78947368,  32.25806452,  95.95959596,
 21.05263158,  84.21052632,  32.        ,  92.63157895,
  3.7037037 , 150.        ,  11.42857143,  88.33333333,
 18.75      ,  96.875     ,  64.        ,  30.43478261,
 93.15068493,  88.70967742,  75.6097561 ,  60.60606061,
 54.16666667,  52.38095238,  98.27586207,  98.63013699,
 76.19047619,  85.10638298,  79.06976744,  89.74358974,
 93.61702128,  87.23404255,   6.25      ,   7.5       ,
 39.39393939,  74.10714286,  49.09090909,  90.24390244,
 56.52173913,  27.02702703,   3.84615385,  31.14754098,
 24.52830189,  97.72727273,  60.71428571,  98.36065574,
 95.91836735,  94.        ,  72.        ,  15.        ,
 12.90322581,  35.29411765,  14.08450704,  13.88888889,
  8.21917808,   3.63636364,  13.04347826,  55.17241379,
 64.51612903,  98.        ,  76.27118644,  98.33333333,
 95.38461538,  85.29411765,  13.51351351,  32.14285714,
 87.91208791,  82.75862069,  72.88135593,  73.68421053,
 86.11111111,  81.3559322 ,  72.83950617,  73.80952381,
 74.19354839,  51.61290323,  71.09375   ,  69.56521739,
  2.94117647,  17.39130435,  85.18518519,   6.4516129 ,
 92.72727273,   8.69565217,   3.33333333,  47.5       ,
 32.35294118,  22.72727273,  98.11320755,  42.30769231,
300.        ,  90.625     ,  84.04907975,  72.09302326,
 98.18181818,  69.04761905,   5.66037736,  93.15960912,
 95.6043956 ,  95.34883721,  98.82352941,  95.77464789,
 94.52054795,  62.06896552,  22.05882353,  25.82781457,
 86.84210526,  82.22222222,  89.04109589,  78.84615385,
 63.15789474,  98.71794872,  93.40659341,  11.53846154,
  4.        ,  86.20689655,  38.0952381 ,  95.55555556,
 97.4025974 ,  94.23076923,  47.61904762,  99.16666667,
 98.38709677,  93.58974359,  89.91596639,  88.48920863,
 89.28571429,  89.8989899 ,  84.41558442,   2.85714286,
 98.4962406 ,  96.59090909,  91.24087591,  94.54545455,
 90.52631579,  67.9245283 ,  10.9375    ,  89.09090909,
 98.79518072,   2.77777778,  94.64285714,  18.91891892,
 67.6056338 ,  55.        ,  53.03030303,  45.09803922,
  5.45454545,  96.36363636,   6.12244898,  98.03921569,
 99.44341373,  98.68852459,  27.5862069 ,  10.25641026,
 11.76470588,   5.12820513,  81.39534884,  69.3877551 ,
 98.61111111,  99.46619217,  98.95178197,  98.72340426,
 97.12230216,  97.18309859,  75.86206897,  98.55072464,
 97.36842105,  56.        ,  98.65771812,  90.19607843,
 77.41935484,  65.625     ,  87.01298701,  25.58139535,
 21.15384615,  71.79487179,  52.        ,   2.22222222,
 15.625     ,   5.        ,  10.71428571,  89.02439024,
 79.31034483,  65.38461538,  94.17475728,  65.11627907,
 59.45945946,  58.82352941,   9.52380952,  10.63829787,
 20.43010753,  89.36170213,  65.2173913 ,  84.09090909,
```

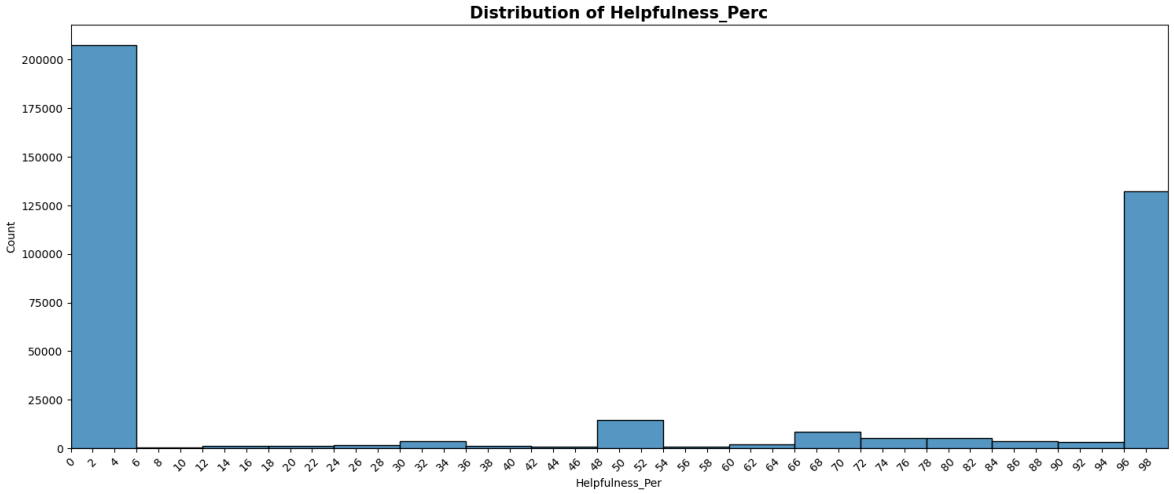
```
92.75362319,  89.15662651,  89.33333333,  89.0625    ,
38.70967742,  60.86956522,  65.85365854,  42.10526316,
88.4057971 ,  92.47311828,  86.48648649,   2.98507463,
40.625     ,  97.91666667,  52.63157895,  19.23076923,
98.57142857,  53.57142857,  12.76595745,  97.33333333,
67.85714286,  93.50649351,  88.97637795,  87.03703704,
81.08108108,  12.24489796,  51.72413793,  89.50276243,
51.85185185,  93.18181818,  82.69230769,  73.52941176,
22.85714286,  62.96296296,  31.03448276,  97.87234043,
96.07843137,  45.71428571,  98.03370787,  93.87755102,
86.9047619 ,  98.26839827,  98.85057471,  98.14814815,
56.75675676,  99.14529915,  17.94871795,  71.64179104,
91.11111111,  82.65306122,  98.73417722,  98.4375    ,
58.06451613,  47.82608696,  44.        ,  39.13043478,
20.45454545,  98.35164835,  95.71428571,  96.5034965 ,
86.26373626,  12.        ,  76.59574468,  86.44067797,
89.87341772,  91.52542373,  91.07142857,  88.63636364,
48.27586207,   3.57142857,  98.5915493 ,  69.09090909,
95.16129032,  48.7804878 ,  13.79310345,   8.10810811,
11.9047619 ,  80.59701493,  92.73743017,  89.3081761 ,
39.6039604 ,  98.24561404,  99.41520468,  98.96907216,
96.72131148,  64.78873239,  23.33333333,  99.19678715,
87.60330579,  86.56716418,  87.09677419,  83.26996198,
84.05797101,  82.9787234 ,  78.33333333,  80.43478261,
78.78787879,  95.12195122,  13.15789474,  96.92307692,
 8.57142857,  98.7012987 ,  77.5       ,  90.16393443,
92.45283019,  34.28571429,  14.81481481,  83.52941176,
79.48717949,  74.66666667,  73.23943662,  74.28571429,
63.85542169,  63.41463415,   4.34782609,  84.7826087 ,
81.63265306,  94.87179487,   4.30107527,  22.58064516,
98.92473118,  84.375     ,  94.04761905,  91.89189189,
85.41666667,  67.30769231,  97.46835443,  74.54545455,
73.11827957,  45.94594595,  97.93814433,  92.40506329,
97.10144928,  68.49315068,  58.        ,  79.10447761,
68.88888889,  99.27884615,  87.17948718,  36.11111111,
36.20689655,  21.62162162,   8.5106383 ,  62.74509804,
48.38709677,  25.80645161,  98.6332574 ,  97.76119403,
97.89915966,  97.5308642 ,  96.47058824,  95.58823529,
89.70588235,  85.48387097,  72.46376812,   5.40540541,
16.12903226,  29.16666667,  23.25581395,  94.62365591,
95.41284404,  75.55555556,  67.64705882,   8.        ,
98.8372093 ,  41.37931034,  59.375     ,  52.77777778,
48.        ,  46.55172414,  34.14634146,  36.61971831,
24.13793103,  30.18867925,  26.5625    ,   9.75609756,
13.55932203,  91.44095341,  91.50943396,  89.62264151,
86.08695652,  85.4368932 ,  85.24590164,  81.6091954 ,
80.3030303 ,  78.        ,  77.64705882,  79.54545455,
76.52173913,  77.96610169,  72.32142857,  72.97297297,
67.74193548,  62.22222222,  98.65229111,  78.87323944,
26.08695652,  71.875     ,  39.28571429,  87.80487805,
69.44444444,  79.41176471,  99.2       ,  97.64705882,
31.57894737,  31.70731707,  88.67924528,  79.59183673,
92.61744966,  86.29173989,  98.66666667,  26.92307692,
17.85714286,  38.23529412,  99.18032787,  15.94202899,
90.27777778,  36.        ,  98.50746269,  77.21518987,
 4.65116279,  68.96551724,  95.89041096,   6.76691729,
56.60377358,  69.76744186,  93.44262295,  97.80775717,
52.17391304,  75.47169811,  70.96774194,  98.07692308,
23.80952381,  95.52238806,  87.14285714,  74.41860465,
83.78378378,  75.51020408,  59.09090909,  89.71193416,
```

```
87.3015873 ,  89.79591837,  73.4939759 ,  99.12280702,
96.6442953 ,  95.87628866,  86.04651163,   7.95454545,
76.66666667,  16.21621622,   2.73972603,   9.67741935,
27.65957447,  83.63636364,  65.30612245,  53.52112676,
97.58064516,  93.47826087,  77.55102041,  98.67256637,
99.61977186,  98.76160991,  97.16981132,  92.95774648,
97.53424658,  97.12389381,  97.19101124,  97.96954315,
96.47887324,  95.49180328,   3.50877193,  73.58490566,
96.34146341,  69.13580247,  61.76470588,  74.35897436,
92.42819843,  93.42105263,  78.72340426,  37.93103448,
95.76271186,  92.78350515,  16.        ,  34.61538462,
76.38888889,  63.46153846,  68.51851852,  67.56756757,
67.5       ,  98.39449541,  40.54054054,  57.57575758,
89.18918919,  86.33093525,  18.60465116,  98.89705882,
96.73913043,  93.79562044,  93.24324324,  98.34710744,
19.44444444,  91.13924051,  84.44444444,  93.22033898,
96.8       ,  53.125     ,  71.69811321,  80.67226891,
 2.3255814 ,  21.875     ,  89.51965066,  71.60493827,
28.26086957,   7.46268657,  97.88732394,  58.97435897,
33.92857143,  21.81818182,   6.77966102,  28.94736842,
96.25      ,  95.08196721,  91.54929577,  10.34482759,
99.21259843,  84.84848485,   7.31707317,  97.8313253 ,
97.97297297,  96.51162791,  92.02453988,  90.14084507,
14.49275362,  37.83783784,  46.51162791,  98.7654321 ,
98.6970684 ,  91.875     ,  84.27419355,  84.69387755,
82.8125    ,  69.6969697 ,   2.43902439,  99.09090909,
94.07894737,  94.66666667,  98.97959184,  81.13207547,
87.65432099,  15.27777778,  68.08510638,  82.02247191,
96.7032967 ,   8.95522388,  31.81818182,  96.61016949,
95.30201342,  94.375     ,  81.57894737,  98.31932773,
98.63945578,  89.71962617,  99.10714286,  64.86486486,
88.94736842,  78.68852459,  82.5       ,  34.48275862,
95.09803922,   4.44444444,  95.8041958 ,  18.46153846,
97.6635514 ,  97.19626168,  72.64150943,  38.7755102 ,
85.96491228,  43.33333333,  34.92063492,   9.1954023 ,
98.52941176,  36.66666667,  96.95121951,  83.92857143,
58.62068966,  98.7804878 ,  48.64864865,   3.44827586,
 4.28571429,  96.03960396,   1.0989011 ,  43.90243902,
31.42857143,  95.3271028 ,  98.69565217,  94.94949495,
89.23076923,  83.67346939,  75.43859649,  24.24242424,
25.96153846,  75.40983607,  14.63414634,  96.27906977,
30.23255814,  97.16312057,  37.03703704,   2.12765957,
59.52380952,  99.5       ,  99.02912621,  99.68944099,
96.91358025,  74.        ,  34.21052632,  38.59649123,
97.85714286,  90.69767442,  20.58823529,  97.80701754,
77.57009346,  11.66666667,  85.38461538,  92.38095238,
76.82926829,  91.61290323,  91.02564103,  99.01960784,
58.18181818,  46.875     ,  73.17073171,  97.32142857,
70.73170732,  59.25925926,  10.81081081,  89.83050847,
44.82758621,  97.45762712,  89.61038961,  54.28571429,
 7.84313725,  98.25174825,  90.234375  ,   2.83018868,
98.94736842,  98.68421053,  17.07317073,  69.72477064,
97.6       ,  98.16513761,  97.59036145,  90.43824701,
40.74074074,  84.90566038,  96.41025641,  93.80530973,
77.35849057,  17.77777778,  44.11764706,   3.07692308,
87.75510204,   9.62962963,  91.83673469,  87.95180723,
80.85106383,  99.14110429])
```

In [119…

```
plt.figure(figsize=(18,7))
sns.histplot(data=df_review["Helpfulness_Per"], bins=50)
```

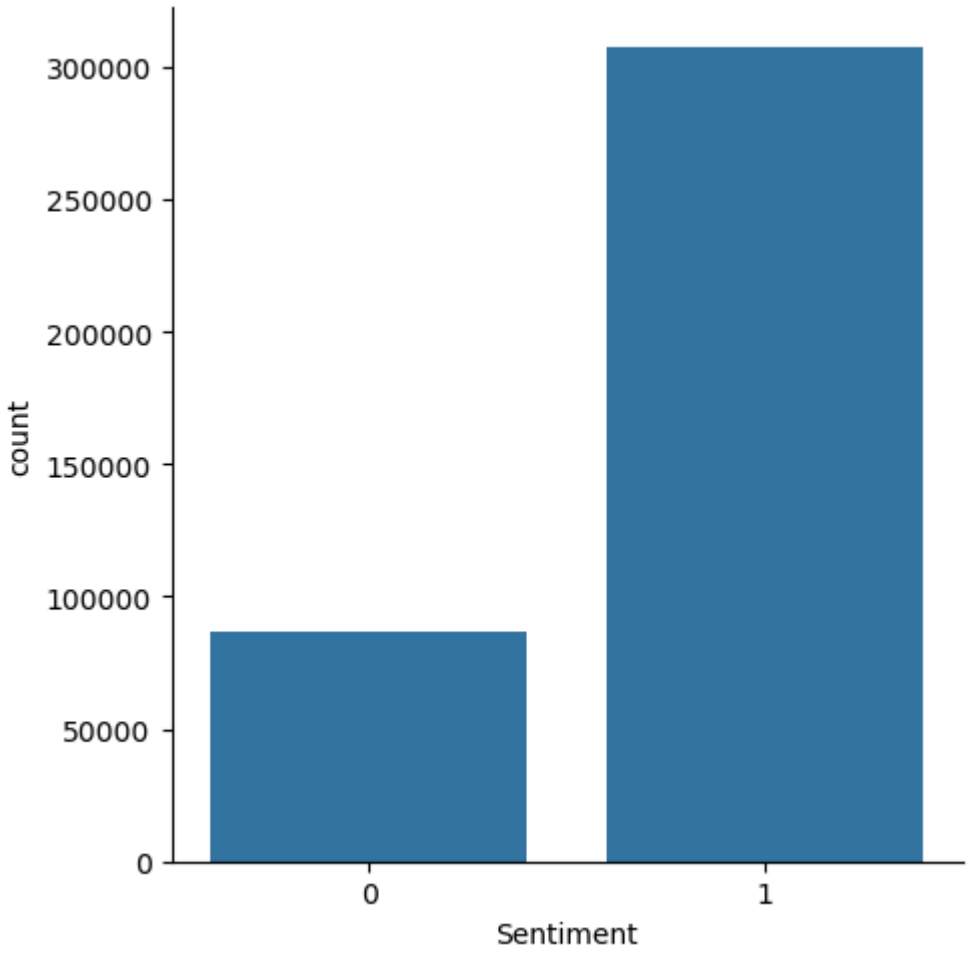
```
plt.title("Distribution of Helpfulness_Perc",fontweight='bold', fontsize=
plt.xticks(range(0,100,2), rotation=45)
plt.xlim(0, 100)
plt.show();
```



Lets check out target variable

In [121…
```
sns.catplot(data = df_review , x = 'Sentiment' , kind = 'count' )
```

Out[121…
```
<seaborn.axisgrid.FacetGrid at 0x326039790>
```



here we can clearly see that out Target feature is immbalance so we cannot apply accuracy matric, so we have to go with the AUC-ROC curve (Area under Roc curve)

we can't choose Accuracy matric because it dosent it will make our model wrongly train , it will baise towards the single category ,our model will became overfit and it also not tells about the nature of the mistakes

we will choose confusion matrics and AUC because if higher the AUC then there is higher chances that model will predict 0 as 0 and 1 as 1 , ROC curve is ploted with TPR against FPR where TPR on y-axis and FPR on x-axis

In [ ]:

lets give the category to the Helpfullness_per,

if percentage greater then equal to 75 then helpful, if percentange greater 40 and less then 75 Intermmidate if percentage greater then 0 and less then equal to 40 then Not useful , if equal to zero then not avilable

In [124…
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_name = ['Condition', 'Class']
x.add_rows([ ["Helpfulness_Per >= 75", "Useful"],
    ["40 < Helpfulness_Per < 75","Intermediate"],
    ["0 < Helpfulness_Per <= 40","Not Useful"],
    ["Helpfulness_Per = 0","Not Available"]])
```

In [125…
```
print(x)
```

```
+--------------------------+---------------+
|          Field 1         |    Field 2    |
+--------------------------+---------------+
|   Helpfulness_Per >= 75  |     Useful    |
| 40 < Helpfulness_Per < 75 |  Intermediate |
| 0 < Helpfulness_Per <= 40 |   Not Useful  |
|    Helpfulness_Per = 0   | Not Available |
+--------------------------+---------------+
```

In [126…
```
df_review.loc[df_review['Helpfulness_Per'] >= 75 , 'type_of_helpful'] = '
df_review.loc[((df_review['Helpfulness_Per'] > 40) & ( df_review['Helpful
df_review.loc[((df_review['Helpfulness_Per'] > 0) & ( df_review['Helpfuln
df_review.loc[df_review['Helpfulness_Per'] == 0 , 'type_of_helpful'] = 'n
```

In [127…
```
df_review
```

Out[127...

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

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	
...	...	...	...	...	
393928	568450	B001EO7N10	A28KG5XORO54AY	Lettie D. Carter	
393929	568451	B003S1WTCU	A3I8AFVPEE8KI5	R. Sawyer	
393930	568452	B004I613EE	A121AA1GQV751Z	pksd "pk_007"	

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
393931	568453	B004I613EE	A3IBEVCTXKNOH	Kathy A. Welch "katwel"	
393932	568454	B001LR2CU2	A3LGQPJCZVL9UC	srfell17	

393933 rows × 14 columns

In [128…
```
df_review.drop(columns = 'Helpfulness_Per', axis =1 , inplace = True)
```

In [129…
```
df_review
```

Out[129...

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

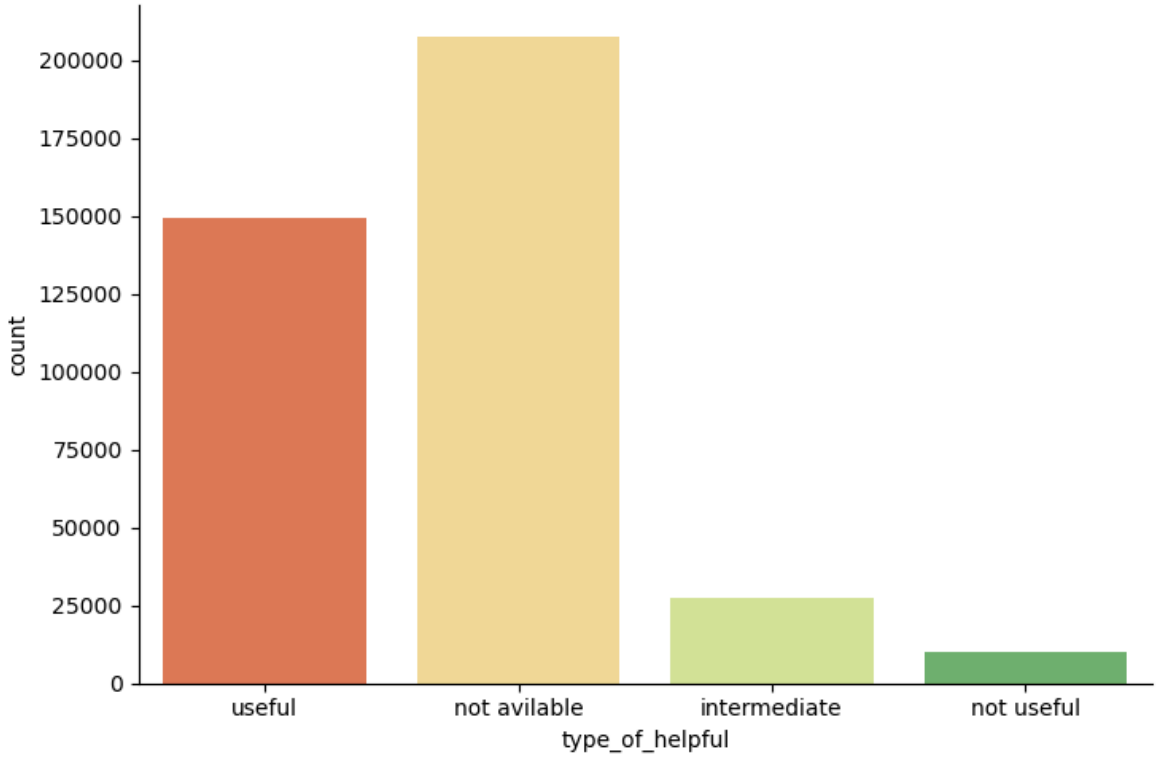
	Id	ProductId	UserId	ProfileName	HelpfulnessNume
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	
...	...	...	...	...	
393928	568450	B001EO7N10	A28KG5XORO54AY	Lettie D. Carter	
393929	568451	B003S1WTCU	A3I8AFVPEE8KI5	R. Sawyer	
393930	568452	B004I613EE	A121AA1GQV751Z	pksd "pk_007"	

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
393931	568453	B004I613EE	A3IBEVCTXKNOH	Kathy A. Welch "katwel"	
393932	568454	B001LR2CU2	A3LGQPJCZVL9UC	srfell17	

393933 rows × 13 columns

In [130…
```
sns.catplot(data = df_review, x = 'type_of_helpful', kind = 'count', pale
```

Out[130…
```
<seaborn.axisgrid.FacetGrid at 0x32545e390>
```



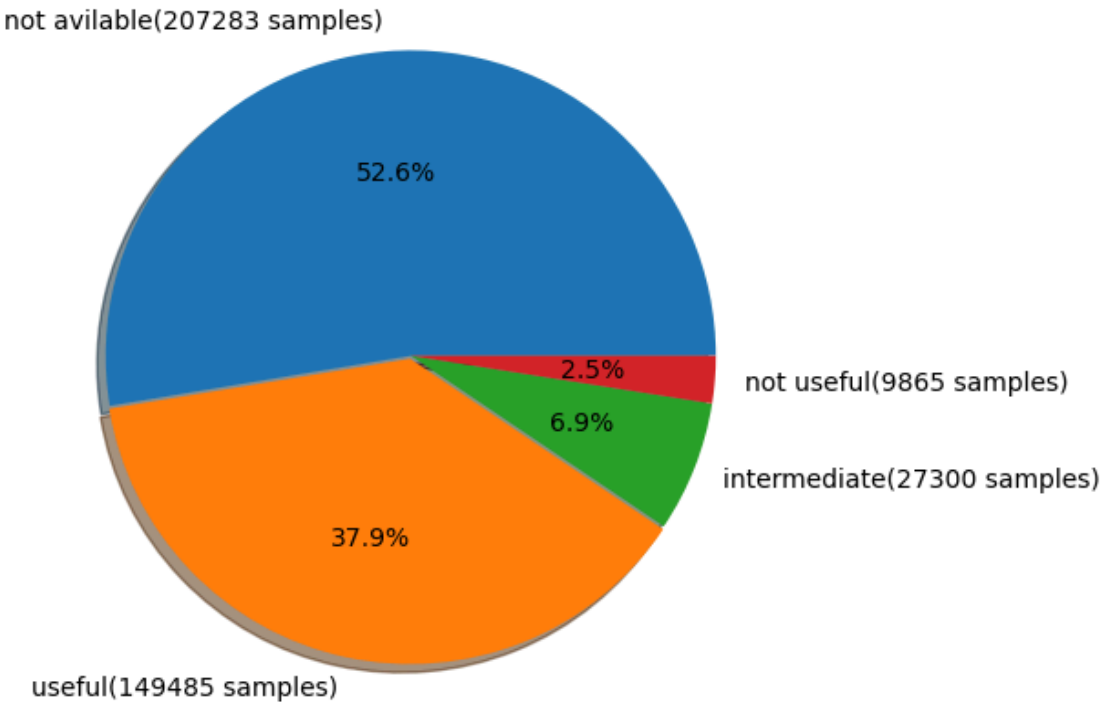
In [131…
```
plt.figure(figsize=(10, 8))
value_counts = df_review['type_of_helpful'].value_counts()
labels = [f'{k}({value_counts[k]} samples)' for k in value_counts.keys()]
```

```
<Figure size 1000x800 with 0 Axes>
```

In [132…
```
labels
```

Out[132…
```
['not avilable(207283 samples)',
 'useful(149485 samples)',
 'intermediate(27300 samples)',
 'not useful(9865 samples)']
```

In [133…
```
plt.pie(value_counts, labels = labels , autopct = '%1.1f%%', explode = [0
plt.axis('equal')
plt.show()
```



we have to consider only polarity ones that are **useful** and **not useful**, ignore **intermediate** and **not avilable**

In [135…
```
# plt.figure(figsize=(10, 8))
# value_counts = df_review['type_of_helpful'].value_counts()
# labels = [f'{k}({value_counts[k]} samples)' for k in value_counts.keys(
```

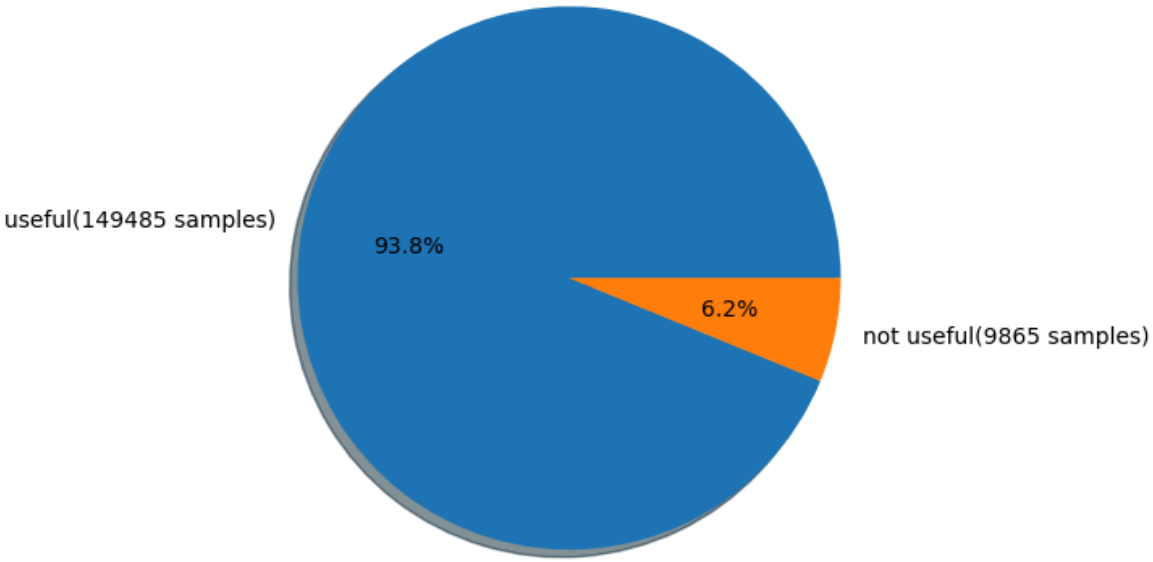
In [136…
```
value_counts.index[0]
```

Out[136… 'not avilable'

In [137…
```
plt.figure(figsize=(10, 8))
value_counts = df_review['type_of_helpful'].value_counts()
labels = [f'{k}({value_counts[k]} samples)' for k in ['useful', 'not usef
```

```
<Figure size 1000x800 with 0 Axes>
```

In [138…
```
plt.pie(value_counts[1:4:2], labels = labels , autopct = '%1.1f%%', shado
plt.axis('equal')
plt.show()
```



94 percent data is useful and **6** percent is not useful

In [140…
```
df_x = df_review[(df_review['type_of_helpful'] == 'useful') | (df_review[
df_x
```

Out[140...

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	
8	9	B000E7L2R4	A1MZYO9TZK0BBI	R. James	

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
10	11	B0001PB9FE	A3HDKO7OW0QNK4	Canadian Fan	
...	...	...	...	...	
393919	568441	B005ZC0RRO	A2TO5R8QLIITEF	SAK	

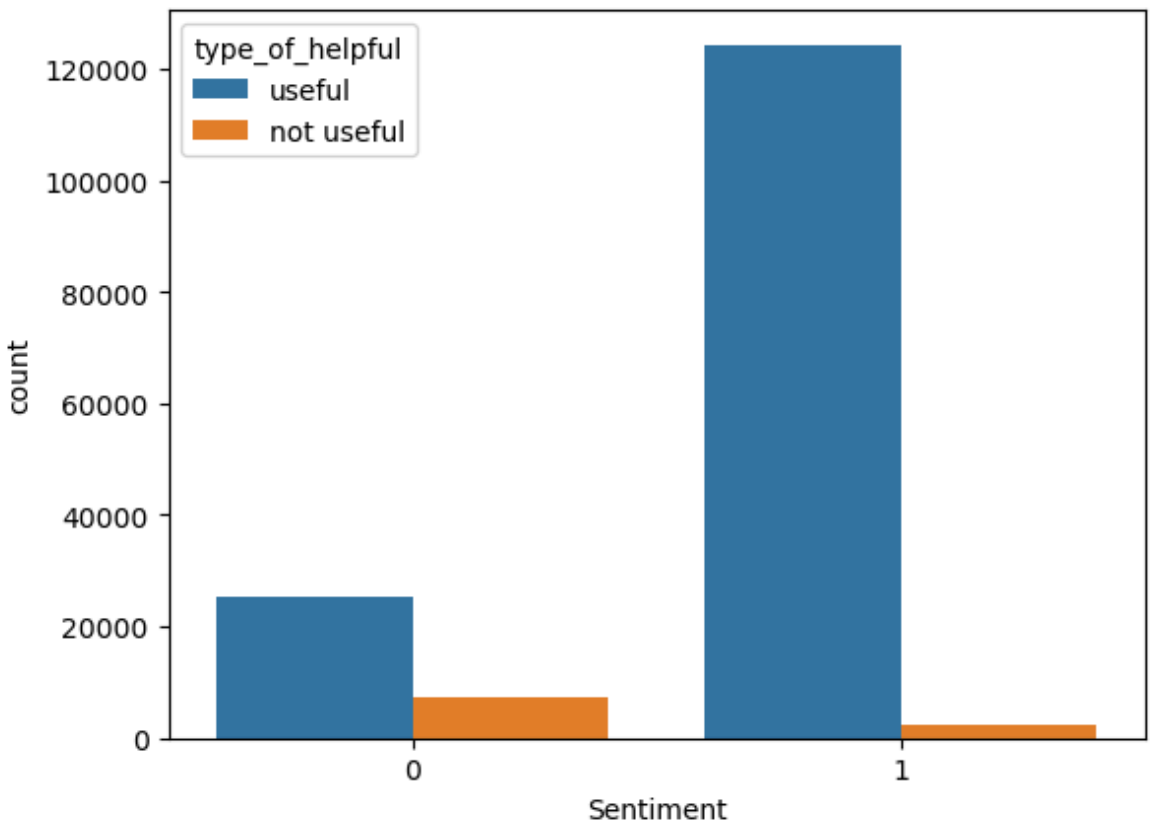
	Id	ProductId	UserId	ProfileName	HelpfulnessNume
393923	568445	B001EO7N10	A2SD7TY3IOX69B	BayBay "BayBay Knows Best"	
393924	568446	B001EO7N10	A2E5C8TTAED4CQ	S. Linkletter	
393930	568452	B004I613EE	A121AA1GQV751Z	pksd "pk_007"	

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
393931	568453	B004I613EE	A3IBEVCTXKNOH	Kathy A. Welch "katwel"	

159350 rows × 13 columns

In [141…
```
sns.countplot(data = df_x, x = 'Sentiment', hue = 'type_of_helpful'  )
```

Out[141…
```
<Axes: xlabel='Sentiment', ylabel='count'>
```



we can clearly see that there are maximum reviews are useful in positive sentiment and also in negitive sentiment people find some amount of useful reviews

Lets assign the word count or the length of each Text review

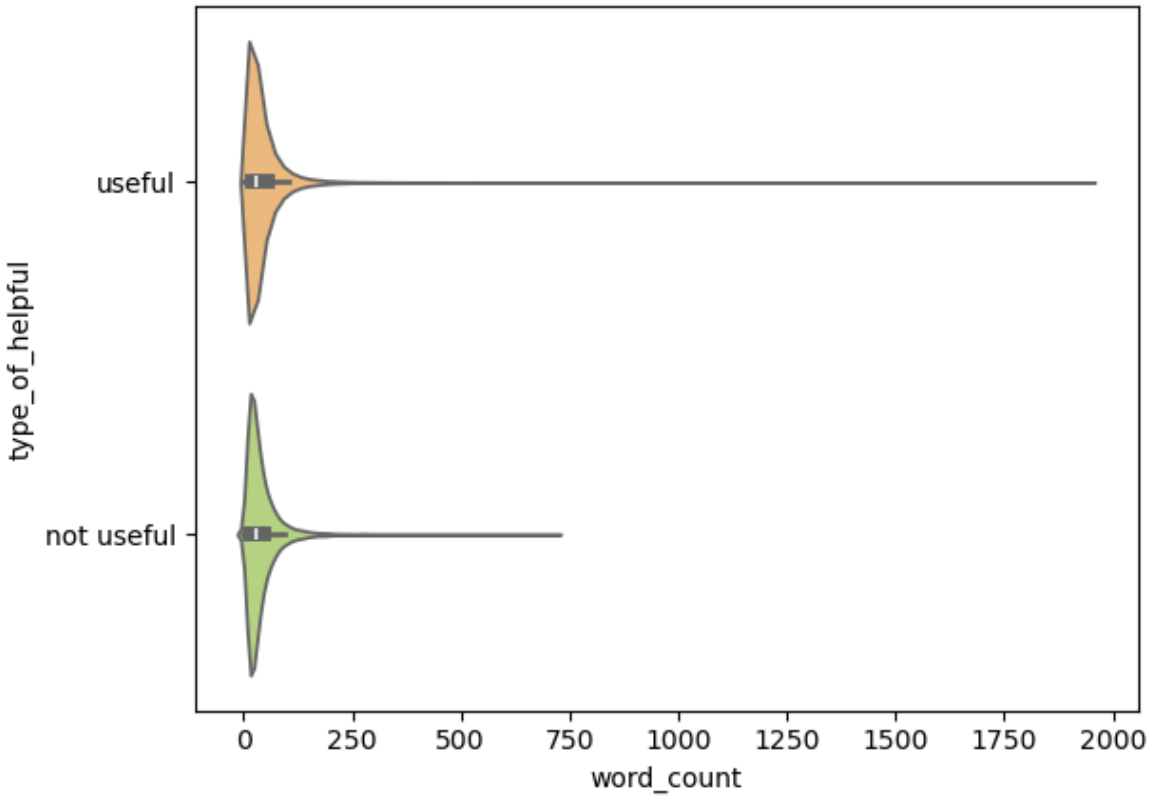
In [144…
```
df_review['word_count'] = df_review['clean_text'].apply(lambda x: len(x.s
```

In [145…
```
df_x['word_count'] = df_x['clean_text'].apply(lambda x: len(x.split()))
```

lets review the length of the reviews on basis of helpfulness

In [147…
```
sns.violinplot(data = df_x , x = 'word_count', y = 'type_of_helpful', pal
```

Out[147… <Axes: xlabel='word_count', ylabel='type_of_helpful'>



there are so many outliers so we will consider only those text having length less then equal to 500

In [149…
```
df_x_500 = df_x[df_x['word_count']<500]
df_x_500
```

Out[149...

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	
8	9	B000E7L2R4	A1MZYO9TZK0BBI	R. James	

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
10	11	B0001PB9FE	A3HDKO7OW0QNK4	Canadian Fan	
...	...	...	...	...	
393919	568441	B005ZC0RRO	A2TO5R8QLIITEF	SAK	

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
393923	568445	B001EO7N10	A2SD7TY3IOX69B	BayBay "BayBay Knows Best"	
393924	568446	B001EO7N10	A2E5C8TTAED4CQ	S. Linkletter	
393930	568452	B004I613EE	A121AA1GQV751Z	pksd "pk_007"	

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
393931	568453	B004I613EE	A3IBEVCTXKNOH	Kathy A. Welch "katwel"	

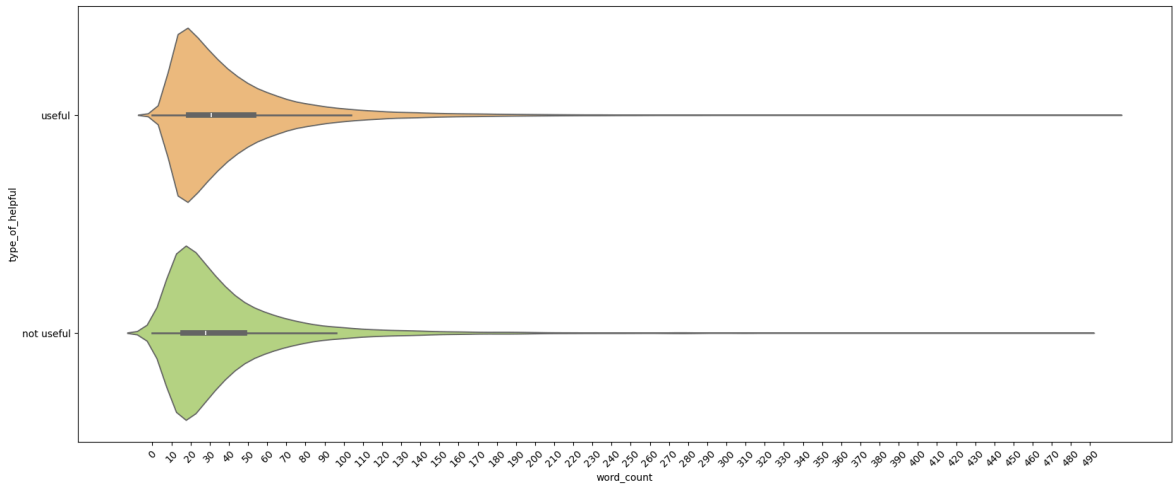
159264 rows × 14 columns

In [150…

```
plt.figure(figsize = (20,8))
sns.violinplot(data = df_x_500 , x = 'word_count', y = 'type_of_helpful',
plt.xticks(range(0,500,10), rotation=45)
```

Out[150… ([<matplotlib.axis.XTick at 0x32e468f50>,
<matplotlib.axis.XTick at 0x32e48fa90>,
<matplotlib.axis.XTick at 0x32f486050>,
<matplotlib.axis.XTick at 0x32e442210>,
<matplotlib.axis.XTick at 0x32e41c550>,
<matplotlib.axis.XTick at 0x32e41e6d0>,
<matplotlib.axis.XTick at 0x32e4177d0>,
<matplotlib.axis.XTick at 0x32e415610>,
<matplotlib.axis.XTick at 0x32e4c6050>,
<matplotlib.axis.XTick at 0x32e40c710>,
<matplotlib.axis.XTick at 0x32e40e890>,
<matplotlib.axis.XTick at 0x32e4076d0>,
<matplotlib.axis.XTick at 0x32e405650>,
<matplotlib.axis.XTick at 0x32e406c50>,
<matplotlib.axis.XTick at 0x32e3fef50>,
<matplotlib.axis.XTick at 0x32e3fced0>,
<matplotlib.axis.XTick at 0x32e3fad50>,
<matplotlib.axis.XTick at 0x32e3f8b50>,
<matplotlib.axis.XTick at 0x32e3f6a10>,
<matplotlib.axis.XTick at 0x32e3f6510>,
<matplotlib.axis.XTick at 0x32e3f43d0>,
<matplotlib.axis.XTick at 0x32e3ea290>,
<matplotlib.axis.XTick at 0x32e3e8150>,
<matplotlib.axis.XTick at 0x32e3e20d0>,
<matplotlib.axis.XTick at 0x32e3fc8d0>,
<matplotlib.axis.XTick at 0x32f5dbb90>,
<matplotlib.axis.XTick at 0x32f5d9b90>,
<matplotlib.axis.XTick at 0x32f5d7a10>,
<matplotlib.axis.XTick at 0x32f5d59d0>,
<matplotlib.axis.XTick at 0x32f5db1d0>,
<matplotlib.axis.XTick at 0x32f5cf350>,
<matplotlib.axis.XTick at 0x32f5cd450>,
<matplotlib.axis.XTick at 0x32f5c8b50>,
<matplotlib.axis.XTick at 0x32f5cadd0>,
<matplotlib.axis.XTick at 0x32f5dbdd0>,
<matplotlib.axis.XTick at 0x32f5bed10>,
<matplotlib.axis.XTick at 0x32f5bcd10>,
<matplotlib.axis.XTick at 0x32f5b6c90>,
<matplotlib.axis.XTick at 0x32f5b4cd0>,
<matplotlib.axis.XTick at 0x32f5b2b10>,
<matplotlib.axis.XTick at 0x32f5cc690>,
<matplotlib.axis.XTick at 0x32f5b0610>,
<matplotlib.axis.XTick at 0x32f5a6610>,
<matplotlib.axis.XTick at 0x32f5a4710>,
<matplotlib.axis.XTick at 0x32f5a26d0>,
<matplotlib.axis.XTick at 0x32f5b0e10>,
<matplotlib.axis.XTick at 0x32f5a0190>,
<matplotlib.axis.XTick at 0x32f59a190>,
<matplotlib.axis.XTick at 0x32f598250>,
<matplotlib.axis.XTick at 0x32f58a350>],
[Text(0, 0, '0'),
Text(10, 0, '10'),
Text(20, 0, '20'),
Text(30, 0, '30'),
Text(40, 0, '40'),
Text(50, 0, '50'),
Text(60, 0, '60'),
Text(70, 0, '70'),
Text(80, 0, '80'),
Text(90, 0, '90'),

```
Text(100, 0, '100'),
Text(110, 0, '110'),
Text(120, 0, '120'),
Text(130, 0, '130'),
Text(140, 0, '140'),
Text(150, 0, '150'),
Text(160, 0, '160'),
Text(170, 0, '170'),
Text(180, 0, '180'),
Text(190, 0, '190'),
Text(200, 0, '200'),
Text(210, 0, '210'),
Text(220, 0, '220'),
Text(230, 0, '230'),
Text(240, 0, '240'),
Text(250, 0, '250'),
Text(260, 0, '260'),
Text(270, 0, '270'),
Text(280, 0, '280'),
Text(290, 0, '290'),
Text(300, 0, '300'),
Text(310, 0, '310'),
Text(320, 0, '320'),
Text(330, 0, '330'),
Text(340, 0, '340'),
Text(350, 0, '350'),
Text(360, 0, '360'),
Text(370, 0, '370'),
Text(380, 0, '380'),
Text(390, 0, '390'),
Text(400, 0, '400'),
Text(410, 0, '410'),
Text(420, 0, '420'),
Text(430, 0, '430'),
Text(440, 0, '440'),
Text(450, 0, '450'),
Text(460, 0, '460'),
Text(470, 0, '470'),
Text(480, 0, '480'),
Text(490, 0, '490')])
```



In [151…

```
df_x_500.groupby('type_of_helpful')['word_count'].describe()
```

Out[151…

type_of_helpful	count	mean	std	min	25%	50%	75%	max
not useful	9852.0	39.715185	41.062526	0.0	16.0	28.0	48.0	479.0
useful	149412.0	43.368518	40.388302	0.0	19.0	31.0	53.0	499.0

here we can see that both are in log normal form, means shot reviews are helpful

1. average length of useful reviews are according to stats are 30-40

Wordcloud for reviews

In [154…
```
from wordcloud import WordCloud
```

In [155…
```
text = " ".join(r for r in df_review['clean_text'])
wordcloud = WordCloud(background_color="white", width=1200, height=600).g
```

In [156…
```
plt.figure(figsize=(30,12))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

lets for only positive sentiment

In [158…
```
df_p = df_review[df_review['Sentiment'] == 1]
text = " ".join(r for r in df_p['clean_text'])
wordcloud = WordCloud(background_color="white", width=1200, height=600).g
```

In [159…
```
plt.figure(figsize=(30,12))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

here we can see that there are words for positive sentiments like

1. well
2. delicious
3. highly recommend
4. really good
5. great Product

etc

In [161…
```
df_n = df_review[df_review['Sentiment'] == 0]
text = " ".join(r for r in df_n['clean_text'])
wordcloud = WordCloud(background_color="white", width=1200, height=600).g
```

In [162…
```
plt.figure(figsize=(30,12))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

for negitive reviews there are

1. waste money
2. tried
3. disappointed
4. expensice etc

Now last step is we will check which is better stemming or lemmatization both are the technique use to convert word into its root form

first we will perform all type of stemming

1. Poerter stemmer

In [167…
```
from nltk.stem.porter import PorterStemmer
from nltk import word_tokenize
```

In [168…
```
porter = PorterStemmer()
a =df_review['clean_text'][0]
word_list = word_tokenize(a)
```

In [169…
```
word_list
```

Out[169…
```
['bought',
 'several',
 'vitality',
 'canned',
 'dog',
 'food',
 'products',
 'found',
 'good',
 'quality',
 'product',
 'looks',
 'like',
 'stew',
 'processed',
 'meat',
 'smells',
 'better',
 'labrador',
 'finicky',
 'appreciates',
 'product',
 'better']
```

In [170…
```
for i in word_list:
    print( f'{i} -> {porter.stem(i)}')
```

```
bought -> bought
several -> sever
vitality -> vital
canned -> can
dog -> dog
food -> food
products -> product
found -> found
good -> good
quality -> qualiti
product -> product
looks -> look
like -> like
stew -> stew
processed -> process
meat -> meat
smells -> smell
better -> better
labrador -> labrador
finicky -> finicki
appreciates -> appreci
product -> product
better -> better
```

several -> sever ,quality -> qualiti, appreciates -> appreci wrongly spelled

2. Snowball Stemmer

In [173…

```
from nltk.stem.snowball import SnowballStemmer
snowball = SnowballStemmer('english')
for i in word_list:
    print( f'{i} -> {snowball.stem(i)}')
```

```
bought -> bought
several -> sever
vitality -> vital
canned -> can
dog -> dog
food -> food
products -> product
found -> found
good -> good
quality -> qualiti
product -> product
looks -> look
like -> like
stew -> stew
processed -> process
meat -> meat
smells -> smell
better -> better
labrador -> labrador
finicky -> finicki
appreciates -> appreci
product -> product
better -> better
```

In [174…
```
#### several -> sever ,quality -> qualiti, appreciates -> appreci wrongly
```

3. Lancanster Stemmer

In [176…
```
from nltk.stem.lancaster import LancasterStemmer
```

In [177…
```
lancaster = LancasterStemmer()
```

In [178…
```
for i in word_list:
    print( f'{i} -> {lancaster.stem(i)}')
```

```
bought -> bought
several -> sev
vitality -> vit
canned -> can
dog -> dog
food -> food
products -> produc
found -> found
good -> good
quality -> qual
product -> produc
looks -> look
like -> lik
stew -> stew
processed -> process
meat -> meat
smells -> smel
better -> bet
labrador -> labrad
finicky -> finicky
appreciates -> apprecy
product -> produc
better -> bet
```

It is performing worst

Lemmatization

1. Wordnet Lemmatizer

In [182…
```
from nltk.stem import WordNetLemmatizer
```

In [183…
```
lemmatizer = WordNetLemmatizer()
for i in word_list:
    print( f'{i} -> {lemmatizer.lemmatize(i)}')
```

```
bought -> bought
several -> several
vitality -> vitality
canned -> canned
dog -> dog
food -> food
products -> product
found -> found
good -> good
quality -> quality
product -> product
looks -> look
like -> like
stew -> stew
processed -> processed
meat -> meat
smells -> smell
better -> better
labrador -> labrador
finicky -> finicky
appreciates -> appreciates
product -> product
better -> better
```

it's look like correct but not as we want

2. Spacy Library

In [186…
```
import spacy
```

In [187…
```
nlp = spacy.load('en_core_web_sm', disable= ['parser','ner'])
doc = nlp(df_review['clean_text'][0])
```

In [188…
```
doc
```

Out[188…
```
bought several vitality canned dog food products found good quality prod
uct looks like stew processed meat smells better labrador finicky apprec
iates product better
```

In [189…
```
print(f"After Lemmatization: {' '.join([token.lemma_ if token.lemma_ != '
```

```
After Lemmatization: buy several vitality can dog food product find good q
uality product look like stew process meat smell well labrador finicky app
reciate product well
```

we can see here **better** -> well , **bought** -> buy

In [191…
```
def spacy_lemmatize(x):
    doc = nlp(x)
    return ' '.join([token.lemma_ if token.lemma_ != '-PRON-' else token
```

so lemmatization by spacy performs well

we will perform Spacy lemmatization on whole data

In [194…
```
from tqdm import tqdm
tqdm.pandas()
```

In [195…
```
df_review['clean_text'] = df_review['clean_text'].progress_apply(lambda x
```

```
100%|██████████| 393933/393933 [16:41<00:00, 393.1
9it/s]
```

In [ ]:
```
```

lets exreact out the important feature

In [197…
```
df_review = df_review.sort_values(by= 'Time').reset_index(drop = True)
```

In [266…
```
df_review
```

Out[266...

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
0	150524	0006641040	ACITT7DI6IDDL	shari zychinski	
1	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	
2	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	
3	230285	B00004RYGX	A344SMIA5JECGM	Vincent P. Ross	

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
4	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	
...	...	...	...	...	
393928	487720	B00816PNK2	ABDQA93G2GTXC	Sheila Fox	
393929	427648	B001VNGHSO	AGIFGYN717K2K	RIchard Sarta	
393930	497428	B0015A2W32	A1SEHFQQ30AR0E	jmble	

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
393931	267764	B000FMZO8G	A1EBWGUV88OZ2G	Kimberly A. Dickens	
393932	41317	B008NDSNAU	A2Z0XFW79HXASE	Kelby Scandrett "Kaiahso"	

393933 rows × 14 columns

In [284…
```
df_final = df_review[['clean_text','Sentiment']]
```

lets save this dataframe into excel file

In [202…
```
df_final.to_excel('final_dataset.xlsx',index = False)
```

Feature Engineering

now we will convert textual data into vectors

we are only going to take 100000 random samples because dataset is to big and it will take so much time

In [286…
```
df = df_final.sample(100000, random_state = 100)
```

In [288…
```
X = df['clean_text']
y = df['Sentiment']
print(X.shape[0])
print(y.shape[0])
```

```
100000
100000
```

In [ ]:
```
```

In [293…
```
from sklearn.model_selection import train_test_split
```

In [295…
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3,
```

In [297…
```
print(X_train.shape[0])
print(X_test.shape[0])
print(y_train.shape[0])
print(y_test.shape[0])
```

```
70000
30000
70000
30000
```

In [299…
```
X_train.info()
```

```
<class 'pandas.core.series.Series'>
Index: 70000 entries, 345312 to 171739
Series name: clean_text
Non-Null Count  Dtype
--------------  -----
70000 non-null  object
dtypes: object(1)
memory usage: 1.1+ MB
```

In [301…
```
# save this array
np.save('split dataset/X_train',X_train)
np.save('split dataset/X_test',X_test)
np.save('split dataset/y_train',y_train)
np.save('split dataset/y_test',y_test)
```

1. BOW (Bag of Words)

In [317…
```
import sklearn
from sklearn.feature_extraction.text import CountVectorizer
```

In [319…
```
bow = CountVectorizer()
bow.fit(X_train)
X_train_bow = bow.transform(X_train)
```

```
X_test_bow = bow.transform(X_test)
bow_features = bow.get_feature_names_out()
print(X_train_bow.get_shape())
print(X_test_bow.get_shape())
```

```
(70000, 43340)
(30000, 43340)
```

In [321…
```
# saving bow vectors
import pickle
```

In [323…
```
pickle.dump(X_train_bow,open('bow_vectors/X_train_bow','wb'))
pickle.dump(X_test_bow,open('bow_vectors/X_test_bow','wb'))
pickle.dump(np.array(bow_features),open('bow_vectors/bow_features','wb'))
```

In [ ]:

2. TF-IDF

In [325…
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [327…
```
tfidf = TfidfVectorizer()
tfidf.fit(X_train)
X_train_tfidf = tfidf.transform(X_train)
X_test_tfidf = tfidf.transform(X_test)
tfidf_features = tfidf.get_feature_names_out()
print(X_train_tfidf.get_shape())
print(X_test_tfidf.get_shape())
```

```
(70000, 43340)
(30000, 43340)
```

In [329…
```
#saving
pickle.dump(X_train_tfidf,open('tfidf_vectors/X_train_tfidf','wb'))
pickle.dump(X_test_tfidf,open('tfidf_vectors/X_test_tfidf','wb'))
pickle.dump(np.array(tfidf_features),open('tfidf_vectors/tfidf_features',
```

Word2Vec

In [ ]:
```
x_reviews = X_train.values
train_sentance = [i.split() for i in x_reviews]
```

In [ ]:
```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
```

In [ ]:
```
w2v_train_model = Word2Vec(train_sentance, min_count = 5, vector_size = 5
w2v_words = list(w2v_train_model.wv.key_to_index.keys())
```

In [ ]:
```
w2v_words
```

1. `w2v_train = Word2Vec(train_sentance, min_count = 5, vector_size = 50, workers = 6)` - This line creates a Word2Vec model using the `Word2Vec` class. The `train_sentance` is the input data for

training the model. The `min_count` parameter specifies the minimum frequency of a word to be included in the model's vocabulary. The `vector_size` parameter sets the dimensionality of the word vectors, and the `workers` parameter specifies the number of threads to use for training.

2. `w2v_words = list(w2v_train.wv.key_to_index.keys())` - This line retrieves the list of words present in the Word2Vec model's vocabulary. It accesses the `wv` attribute of the trained Word2Vec model to get the word vectors, and then retrieves the keys (words) from the `key_to_index` dictionary. Finally, it converts the keys into a list and assigns it to the variable `w2v_words`.

In [ ]:
```
# saving the model
```

In [ ]:
```
pickle.dump(np.array(w2v_words), open('w2v_words','wb'))
```

3. Average Word2Vec

In [ ]:
```
train_reviews = X_train.values
train_sentance = [i.split() for i in train_reviews]
```

In [ ]:
```
vector_train = []
for sen in tqdm(train_sentance):
    sen_vec = np.zeros(50)
    cnt_words = 0
    for word in sen:
        if word in w2v_words:
            vector = w2v_train_model.wv[word]
            sen_vec += vector
            cnt_words += 1
    if cnt_words!=0:
        sen_vec /= cnt_words
    vector_train.append(sen_vec)
```

In [ ]:
```
test_reviews = X_test.values
test_sentance = [i.split() for i in test_reviews]
```

In [ ]:
```
vector_test = []
for sen in tqdm(test_sentance):
    sen_vec = np.zeros(50)
    cnt = 0
    for word in sen:
        if word in w2v_words:
            vector = w2v_train_model.wv[word]
            sen_vec += vector
            cnt += 1
    if cnt!=0:
        sen_vec /= cnt
    vector_test.append(sen_vec)
```

In [ ]:
```
#saving
X_train_avgw2v = np.array(vector_train)
X_test_avgw2v = np.array(vector_test)
```

```
pickle.dump(np.array(X_train_avgw2v), open('avgw2v_train','wb'))
pickle.dump(np.array(X_test_avgw2v), open('avgw2v_test','wb'))
```

In [331…
```
pickle.dump(tfidf, open('vectorizer.pkl','wb'))
```

In [ ]:

In [ ]:

Model Creation

In [74]:
```
import pandas as pd
import numpy as np
import sklearn
import pickle
from sklearn.metrics import auc,roc_curve,roc_auc_score

import matplotlib.pyplot as plt
import seaborn as sns
```

Navie Bayes

In [76]:
```
from sklearn.naive_bayes import MultinomialNB
```

In [77]:
```
#loding the vectors
```

Loading BOW vectors

In [79]:
```
X_train_bow = pickle.load(open('bow_vectors/X_train_bow','rb'))
X_test_bow = pickle.load(open('bow_vectors/X_test_bow','rb'))
bow_features = pickle.load(open('bow_vectors/bow_features','rb'))
```

In [80]:
```
y_train = np.load('split dataset/y_train.npy',allow_pickle = True)
y_test = np.load('split dataset/y_test.npy',allow_pickle = True)
```

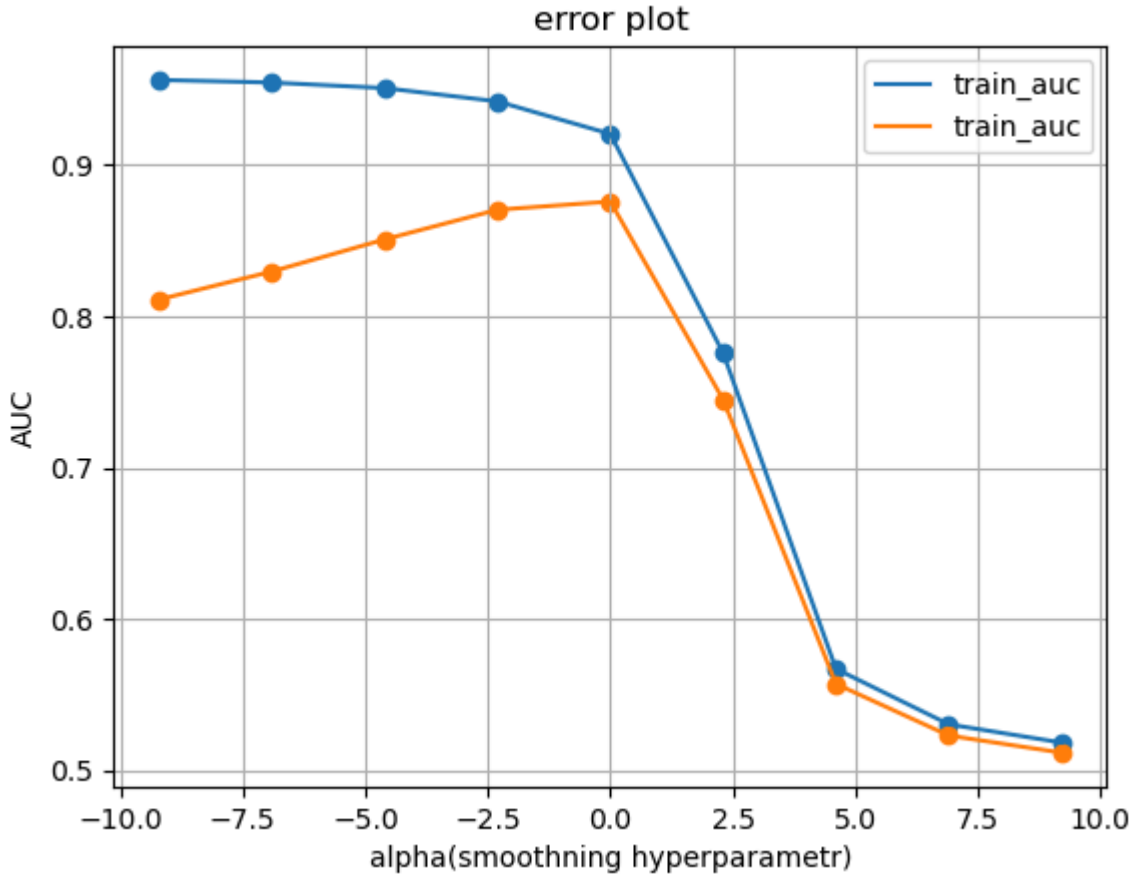
In [ ]:
```

```

In [81]:
```
train_auc = []
test_auc = []
alpha = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
for i in alpha:
    naive = MultinomialNB(alpha = i)
    naive.fit(X_train_bow,y_train)
    y_train_pred = naive.predict_proba(X_train_bow)[:,1]
    y_test_pred = naive.predict_proba(X_test_bow)[:,1]
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    test_auc.append(roc_auc_score(y_test,y_test_pred))
```

In [82]:
```
#graph
```

In [83]:
```
plt.grid(True)
plt.plot(np.log(alpha), train_auc, label = 'train_auc' )
plt.plot(np.log(alpha), test_auc, label = 'train_auc' )
plt.scatter(np.log(alpha), train_auc)
plt.scatter(np.log(alpha), test_auc)
plt.legend()
plt.xlabel('alpha(smoothning hyperparametr)')
plt.ylabel('AUC')
plt.title('error plot')
```

Out[83]:
```
Text(0.5, 1.0, 'error plot')
```

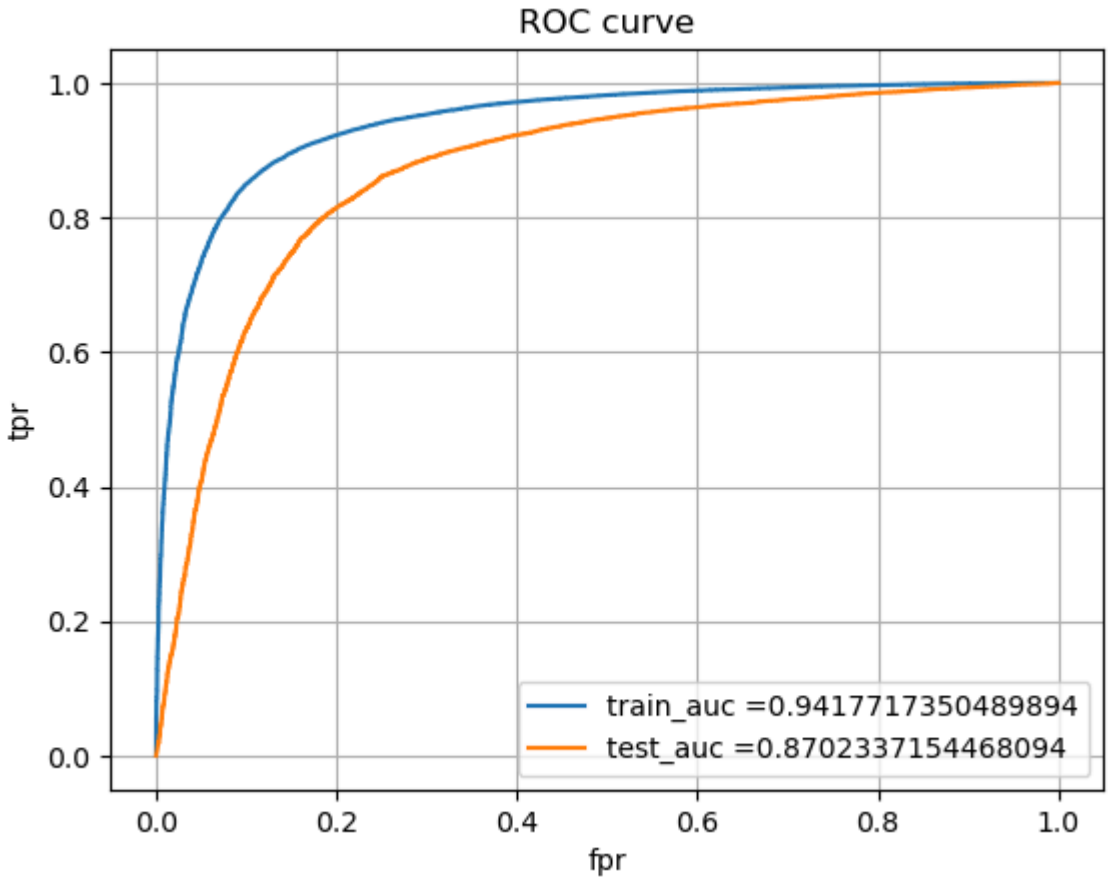


here we can see that at alpha = 0.1 both test and traning has max AUC

In [85]:
```
optimize_alpha = 0.1
naive_b = MultinomialNB(alpha = optimize_alpha)
naive_b = naive_b.fit(X_train_bow,y_train)
train_fpr,train_tpr,train_thresholds = roc_curve(y_train, naive_b.predict
test_fpr,test_tpr,test_thresholds = roc_curve(y_test, naive_b.predict_pro
```

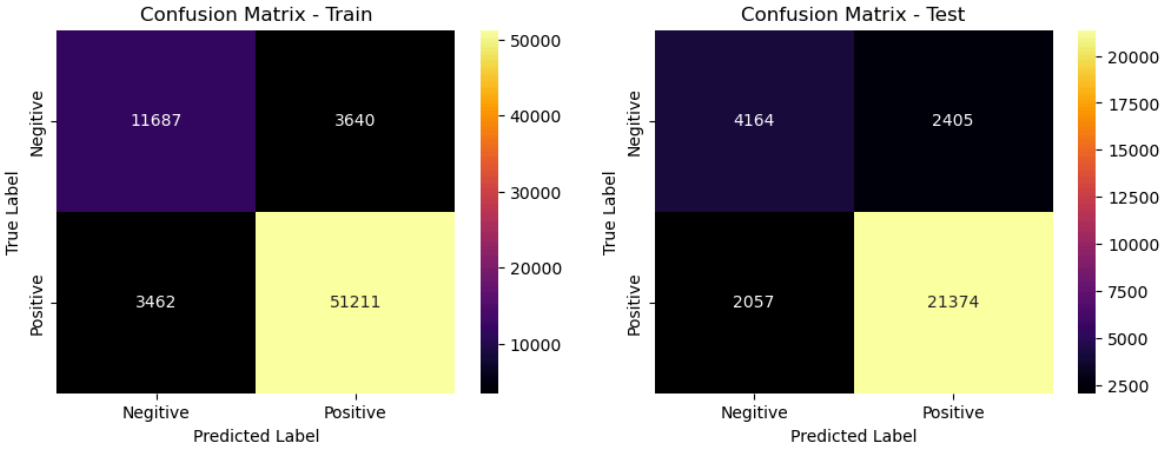
In [86]:
```
#graph roc curve
```

In [87]:
```
plt.grid(True)
plt.plot(train_fpr, train_tpr, label = 'train_auc ='+str(auc(train_fpr, t
plt.plot(test_fpr, test_tpr, label = 'test_auc ='+str(auc(test_fpr,test_t
plt.legend()
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('ROC curve')
plt.show()
```



In [88]:
```
from sklearn.metrics import confusion_matrix
# confusion matrix
def confusion_matrixes(model, X_train, y_train, X_test, y_test):
    train_cm = confusion_matrix(y_train,model.predict(X_train))
    test_cm = confusion_matrix(y_test, model.predict(X_test))
    col_n = ['Negitive','Positive']
    df_train = pd.DataFrame(train_cm,index = col_n, columns= col_n)
    df_test = pd.DataFrame(test_cm,index = col_n, columns= col_n)
    f, axes = plt.subplots(1,2,figsize= (12,4))

    for i in range(2):
        df = df_train if i == 0 else df_test
        sns.heatmap(df,annot = True, cmap = 'inferno', ax = axes[i], fmt
        axes[i].set_title(f"Confusion Matrix - {'Train' if i==0 else 'Tes
        axes[i].set_xlabel("Predicted Label")
        axes[i].set_ylabel("True Label")
    plt.show()
```

In [89]:
```
confusion_matrixes(naive_b,X_train_bow,y_train,X_test_bow,y_test)
```



In [90]:
```
### Top positive and negitive Features
```

In [91]:
```
coff = naive_b.feature_log_prob_[1].reshape(-1,1)
df_imp = pd.DataFrame(coff,columns = ['coef'],index = bow_features)
print('positive words')
positive = df_imp.sort_values(by = 'coef', ascending= False).head(10)
print(positive)
negitive = df_imp.sort_values(by = 'coef').head(10)
print(negitive)
```

```
positive words
           coef
good   -4.376268
like   -4.423173
taste  -4.496069
love   -4.691931
great  -4.692837
flavor -4.725639
use    -4.726798
make   -4.739068
one    -4.758813
get    -4.817477
                coef
stawberrie  -16.83427
mustardms   -16.83427
ewwww       -16.83427
ewwwww      -16.83427
ewwwwww     -16.83427
unmitigated -16.83427
streache    -16.83427
disgutingly -16.83427
porcupine   -16.83427
porduct     -16.83427
```

In [ ]:

In [ ]:

Using TI-IDF

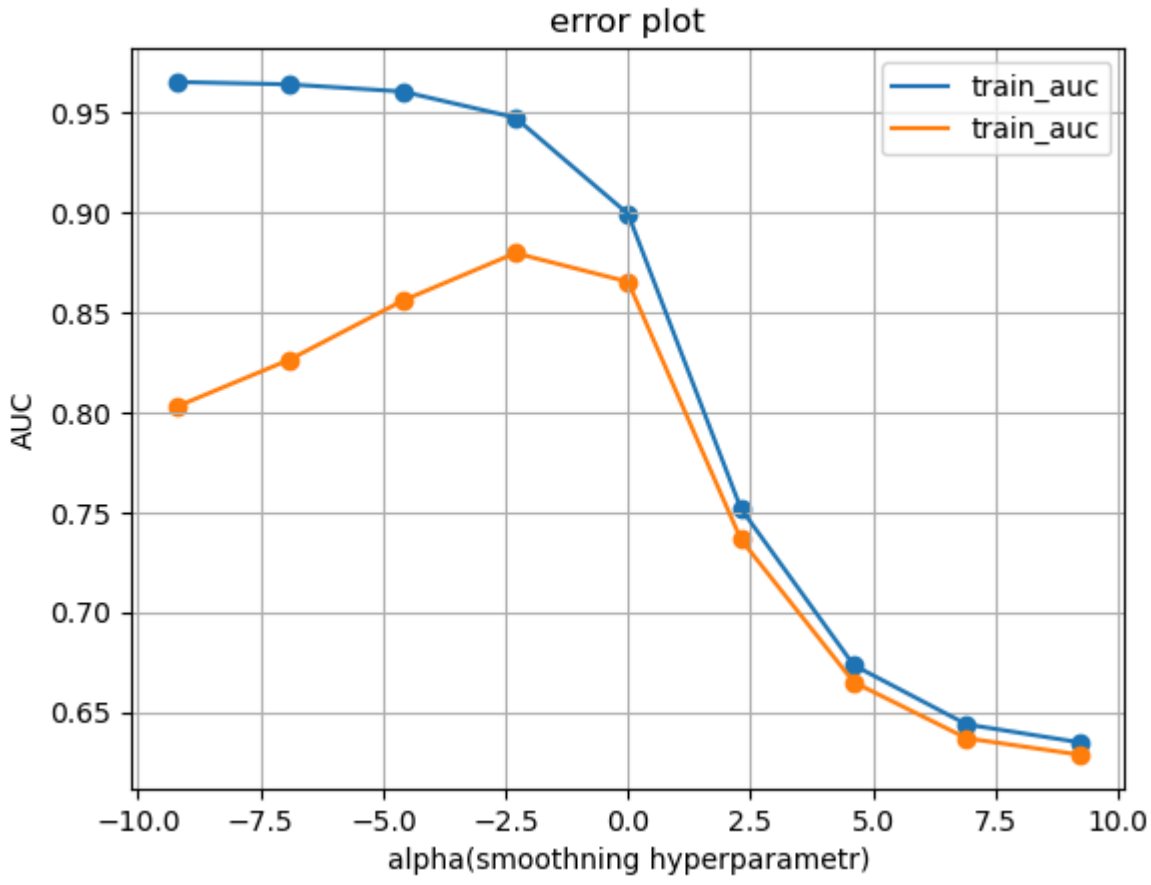
In [94]:
```
# loading vectors
```

In [95]:
```
X_train_tfidf = pickle.load(open('tfidf_vectors/X_train_tfidf','rb'))
X_test_tfidf = pickle.load(open('tfidf_vectors/X_test_tfidf','rb'))
tfidf_features = pickle.load(open('tfidf_vectors/tfidf_features','rb'))
```

In [96]:
```
train_auc = []
test_auc = []
alpha = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
for i in alpha:
    naive = MultinomialNB(alpha = i)
    naive.fit(X_train_tfidf,y_train)
    y_train_pred = naive.predict_proba(X_train_tfidf)[:,1]
    y_test_pred = naive.predict_proba(X_test_tfidf)[:,1]
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    test_auc.append(roc_auc_score(y_test,y_test_pred))
```

In [97]:
```
# error graph

plt.grid(True)
plt.plot(np.log(alpha), train_auc, label = 'train_auc' )
plt.plot(np.log(alpha), test_auc, label = 'train_auc' )
plt.scatter(np.log(alpha), train_auc)
plt.scatter(np.log(alpha), test_auc)
plt.legend()
plt.xlabel('alpha(smoothning hyperparametr)')
plt.ylabel('AUC')
plt.title('error plot')
```

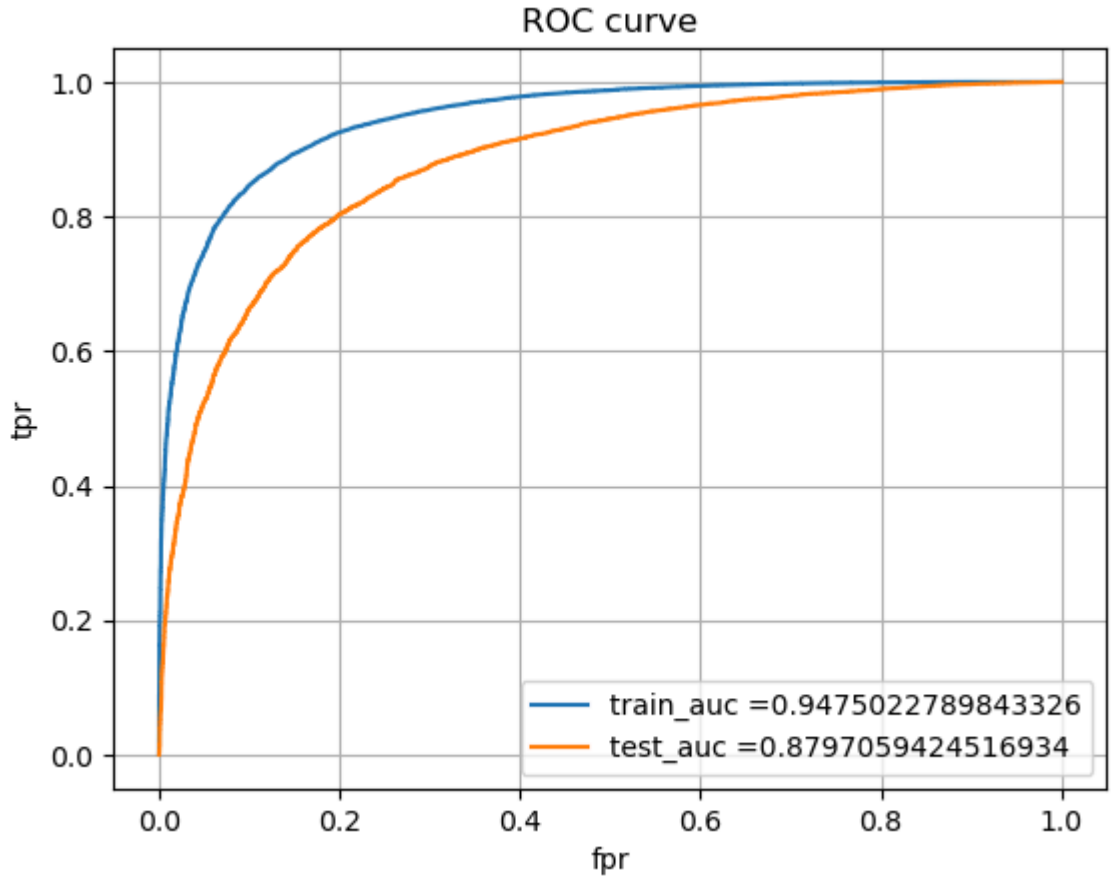
Out[97]: Text(0.5, 1.0, 'error plot')



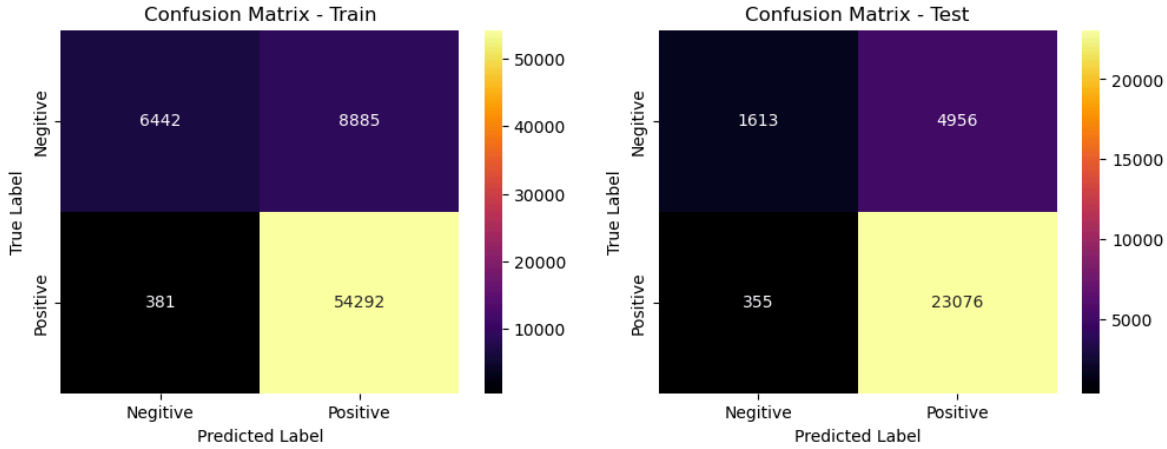
In [98]:
```
optimize_alpha = 0.1
naive_b = MultinomialNB(alpha = optimize_alpha)
naive_b = naive_b.fit(X_train_tfidf,y_train)
```

```
train_fpr,train_tpr,train_thresholds = roc_curve(y_train, naive_b.predict
test_fpr,test_tpr,test_thresholds = roc_curve(y_test, naive_b.predict_pro
```

In [99]:
```
plt.grid(True)
plt.plot(train_fpr, train_tpr, label = 'train_auc ='+str(auc(train_fpr, t
plt.plot(test_fpr, test_tpr, label = 'test_auc ='+str(auc(test_fpr,test_t
plt.legend()
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('ROC curve')
plt.show()
```



In [100…
```
confusion_matrixes(naive_b,X_train_tfidf,y_train,X_test_tfidf,y_test)
```



In [101…
```
coff = naive_b.feature_log_prob_[1].reshape(-1,1)
df_imp = pd.DataFrame(coff,columns = ['coef'],index = tfidf_features)
print('positive words')
positive = df_imp.sort_values(by = 'coef', ascending= False).head(10)
```

```
print(positive)
print('='*50)
print('negitive words')
negitive = df_imp.sort_values(by = 'coef').head(10)
print(negitive)
```

```
positive words
             coef
good    -4.987432
great   -5.059118
love    -5.081436
taste   -5.166487
like    -5.176879
coffee  -5.253241
tea     -5.254010
flavor  -5.284794
product -5.328445
use     -5.347157
==================================================
negitive words
                           coef
boondock               -14.76285
snaile                 -14.76285
oniongarlicoreganosoya -14.76285
isugary                -14.76285
onlythat               -14.76285
onmto                  -14.76285
isucrose               -14.76285
isucralose             -14.76285
isubtle                -14.76285
istyle                 -14.76285
```

In [ ]:

Observation

1. Here we applied both BOW and TFIDF features on naive bayes model
2. TFIDF features model performs slightly better then BOW
3. both looks like overfitter ,also after hyperparameter tunning
4. Naive bayes is not good for this dataset

In [104…

```
from prettytable import PrettyTable
```

In [105…

```
z = PrettyTable()
z.field_names = ["Vector","Algorithm","Hyperparametr-alpha","Train AUC","
z.add_row(['BOW', 'Naive-Bayes', '0.1', '0.941','0.8702' ])
z.add_row(['TF-IDF', 'Naive-Bayes', '0.1', '0.947','0.8797'])
```

In [106…

```
print(z)
```

```
+--------+-------------+---------------------+-----------+----------+
| Vector |  Algorithm  | Hyperparametr-alpha | Train AUC | Test AUC |
+--------+-------------+---------------------+-----------+----------+
|  BOW   | Naive-Bayes |         0.1         |   0.941   |  0.8702  |
| TF-IDF | Naive-Bayes |         0.1         |   0.947   |  0.8797  |
+--------+-------------+---------------------+-----------+----------+
```

In [ ]:

In [107…
```
pickle.dump(naive_b,open('model.pkl','wb'))
```

In [ ]:

In [ ]:

2. Logistic Regression

In [71]:
```
import numpy as np
import pandas as pd
import sklearn
import pickle
import sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve,auc
```

1.BOW

loading dataset

Loading BOW vectors

In [7]:
```
X_train_bow = pickle.load(open('bow_vectors/X_train_bow','rb'))
X_test_bow = pickle.load(open('bow_vectors/X_test_bow','rb'))
bow_features = pickle.load(open('bow_vectors/bow_features','rb'))
```

In [8]:
```
y_train = np.load('split dataset/y_train.npy',allow_pickle = True)
y_test = np.load('split dataset/y_test.npy',allow_pickle = True)
```

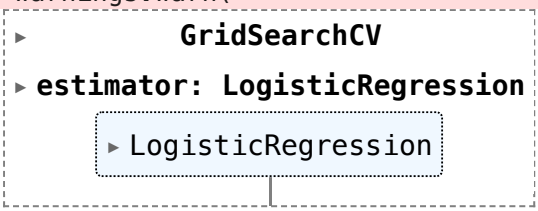
In [ ]:
```

```

In [9]:
```
penalty = ['l1', 'l2']
C = [0.0001,0.001,0.01,0.1,1,10,100,1000]
grid = {'penalty': penalty, 'C': C}
log_reg = LogisticRegression(solver = 'liblinear')
gridcv = GridSearchCV(log_reg, grid, cv =5,return_train_score = True,n_jo
gridcv.fit(X_train_bow,y_train)
```

```
/opt/anaconda3/lib/python3.11/site-packages/sklearn/svm/_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of iter
ations.
  warnings.warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/svm/_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of iter
ations.
  warnings.warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/svm/_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of iter
ations.
  warnings.warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/svm/_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of iter
ations.
  warnings.warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/svm/_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of iter
ations.
  warnings.warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/svm/_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of iter
ations.
  warnings.warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/svm/_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of iter
ations.
  warnings.warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/svm/_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of iter
ations.
  warnings.warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/svm/_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of iter
ations.
  warnings.warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/svm/_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of iter
ations.
  warnings.warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/svm/_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of iter
ations.
  warnings.warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/svm/_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of iter
ations.
  warnings.warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/svm/_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of iter
ations.
  warnings.warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/svm/_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of iter
ations.
  warnings.warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/svm/_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of iter
ations.
  warnings.warn(
```

```
/opt/anaconda3/lib/python3.11/site-packages/sklearn/svm/_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of iter
ations.
  warnings.warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/svm/_base.py:1244: Con
vergenceWarning: Liblinear failed to converge, increase the number of iter
ations.
  warnings.warn(
```

Out[9]:



In [10]:
```
print(gridcv.best_params_)
```

```
{'C': 0.1, 'penalty': 'l2'}
```

In [22]:
```
gridcv.cv_results_['mean_train_score']
```

Out[22]:
```
array([0.78104286, 0.78123929, 0.7812    , 0.81880357, 0.82720714,
       0.86631429, 0.87328214, 0.90054643, 0.917425  , 0.9356    ,
       0.97317143, 0.95697143, 0.98299286, 0.95890357, 0.98496071,
       0.95671071])
```

In [24]:
```
gridcv.cv_results_
```

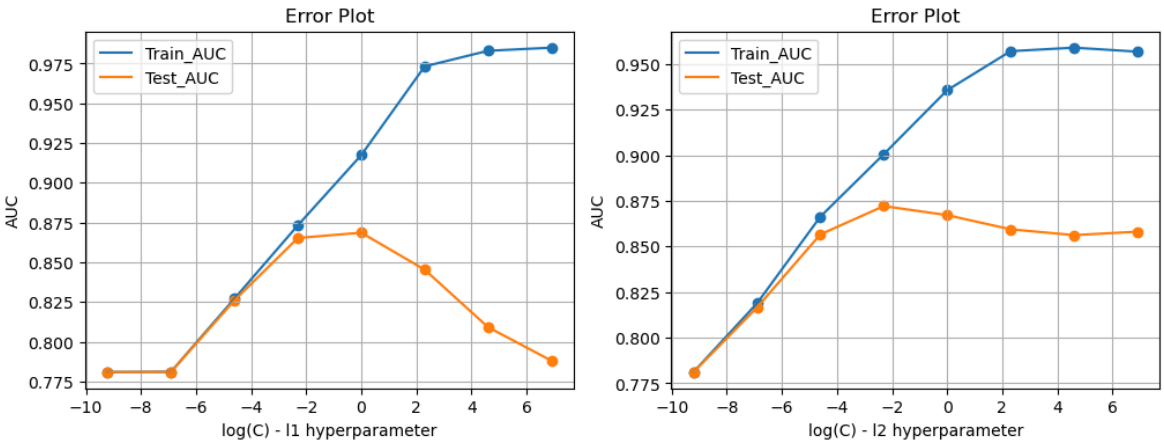
Out[24]:

```
{'mean_fit_time': array([1.09757805e-01, 2.07765484e-01, 1.92910671e-01,
3.51239252e-01,
       2.44152164e-01, 9.23582411e-01, 3.38210297e-01, 2.12069688e+00,
       5.40400076e-01, 6.66661582e+00, 1.50696673e+00, 9.37419658e+00,
       4.52928934e+00, 9.03660913e+00, 1.56261914e+02, 7.67839222e+0
0]),
 'std_fit_time': array([1.75966184e-02, 8.24725611e-03, 2.63698549e-02,
3.52620850e-02,
       3.05940739e-02, 1.73277621e-01, 3.28872317e-02, 3.18443791e-01,
       5.19580739e-02, 8.73339568e-01, 2.67318277e-01, 1.61491312e+00,
       2.52973576e+00, 1.81205754e+00, 1.83904484e+02, 9.98426524e-0
1]),
 'mean_score_time': array([0.00380936, 0.00216684, 0.0039608 , 0.0024338
7, 0.00238242,
       0.00281415, 0.00437484, 0.00219865, 0.00303073, 0.00235815,
       0.00205746, 0.00244536, 0.00288458, 0.00249009, 0.00203805,
       0.00226722]),
 'std_score_time': array([0.00150413, 0.00033518, 0.00202644, 0.0004451
6, 0.00069058,
       0.00114241, 0.00245416, 0.00071884, 0.00138292, 0.00055108,
       0.00024207, 0.00073139, 0.00075526, 0.00092487, 0.00065206,
       0.00052917]),
 'param_C': masked_array(data=[0.0001, 0.0001, 0.001, 0.001, 0.01, 0.01,
0.1, 0.1, 1,
                   1, 10, 10, 100, 100, 1000, 1000],
             mask=[False, False, False, False, False, False, False, Fal
se,
                   False, False, False, False, False, False, False, Fal
se],
       fill_value='?',
            dtype=object),
 'param_penalty': masked_array(data=['l1', 'l2', 'l1', 'l2', 'l1', 'l2',
'l1', 'l2', 'l1',
                   'l2', 'l1', 'l2', 'l1', 'l2', 'l1', 'l2'],
             mask=[False, False, False, False, False, False, False, Fal
se,
                   False, False, False, False, False, False, False, Fal
se],
       fill_value='?',
            dtype=object),
 'params': [{'C': 0.0001, 'penalty': 'l1'},
  {'C': 0.0001, 'penalty': 'l2'},
  {'C': 0.001, 'penalty': 'l1'},
  {'C': 0.001, 'penalty': 'l2'},
  {'C': 0.01, 'penalty': 'l1'},
  {'C': 0.01, 'penalty': 'l2'},
  {'C': 0.1, 'penalty': 'l1'},
  {'C': 0.1, 'penalty': 'l2'},
  {'C': 1, 'penalty': 'l1'},
  {'C': 1, 'penalty': 'l2'},
  {'C': 10, 'penalty': 'l1'},
  {'C': 10, 'penalty': 'l2'},
  {'C': 100, 'penalty': 'l1'},
  {'C': 100, 'penalty': 'l2'},
  {'C': 1000, 'penalty': 'l1'},
  {'C': 1000, 'penalty': 'l2'}],
 'split0_test_score': array([0.781     , 0.78092857, 0.78092857, 0.81592
857, 0.82442857,
       0.85714286, 0.865     , 0.87185714, 0.87057143, 0.86892857,
       0.84785714, 0.86021429, 0.8125    , 0.85035714, 0.792     ,
```

```
        0.86171429]),
 'split1_test_score': array([0.781     , 0.78128571, 0.78107143, 0.82057
143, 0.82857143,
        0.85735714, 0.86564286, 0.87371429, 0.86635714, 0.86507143,
        0.84485714, 0.85957143, 0.80821429, 0.85978571, 0.78757143,
        0.85957143]),
 'split2_test_score': array([0.78107143, 0.78128571, 0.78157143, 0.81414
286, 0.82471429,
        0.85542857, 0.86671429, 0.87378571, 0.872     , 0.87185714,
        0.84757143, 0.86364286, 0.8115    , 0.85814286, 0.78978571,
        0.85671429]),
 'split3_test_score': array([0.78107143, 0.78135714, 0.781     , 0.81371
429, 0.825     ,
        0.85607143, 0.86435714, 0.871     , 0.867     , 0.86492857,
        0.84414286, 0.85464286, 0.80792857, 0.85564286, 0.7865    ,
        0.85414286]),
 'split4_test_score': array([0.78107143, 0.78114286, 0.78128571, 0.81835
714, 0.82792857,
        0.857     , 0.86485714, 0.87021429, 0.86728571, 0.865     ,
        0.8425    , 0.85892857, 0.80607143, 0.85707143, 0.785     ,
        0.85835714]),
 'mean_test_score': array([0.78104286, 0.7812    , 0.78117143, 0.8165428
6, 0.82612857,
        0.8566    , 0.86531429, 0.87211429, 0.86864286, 0.86715714,
        0.84538571, 0.8594    , 0.80924286, 0.8562    , 0.78817143,
        0.8581    ]),
 'std_test_score': array([3.49927106e-05, 1.52529689e-04, 2.32992949e-0
4, 2.59457833e-03,
        1.75336411e-03, 7.32064456e-04, 8.11398390e-04, 1.43327796e-03,
        2.22508885e-03, 2.79992711e-03, 2.05118184e-03, 2.88260087e-03,
        2.38943252e-03, 3.21996324e-03, 2.46882605e-03, 2.56411659e-0
3]),
 'rank_test_score': array([16, 14, 15, 11, 10,  7,  4,  1,  2,  3,  9,
5, 12,  8, 13,  6],
       dtype=int32),
 'split0_train_score': array([0.78105357, 0.78128571, 0.78125   , 0.8186
0714, 0.82701786,
        0.86610714, 0.87257143, 0.90016071, 0.91691071, 0.93425   ,
        0.97232143, 0.96219643, 0.98230357, 0.96969643, 0.98433929,
        0.95617857]),
 'split1_train_score': array([0.78105357, 0.78123214, 0.78119643, 0.8180
7143, 0.82692857,
        0.86544643, 0.87364286, 0.90003571, 0.91814286, 0.93519643,
        0.97333929, 0.94676786, 0.98276786, 0.94514286, 0.98489286,
        0.94566071]),
 'split2_train_score': array([0.78103571, 0.78119643, 0.781125  , 0.8196
7857, 0.82723214,
        0.867125  , 0.87308929, 0.90075   , 0.91725   , 0.93598214,
        0.97285714, 0.95769643, 0.98244643, 0.96346429, 0.984     ,
        0.964375  ]),
 'split3_train_score': array([0.78103571, 0.78125   , 0.78128571, 0.8193
75  , 0.82726786,
        0.86641071, 0.87325   , 0.901     , 0.91746429, 0.93619643,
        0.97330357, 0.96421429, 0.98317857, 0.96057143, 0.98491071,
        0.96269643]),
 'split4_train_score': array([0.78103571, 0.78123214, 0.78114286, 0.8182
8571, 0.82758929,
        0.86648214, 0.87385714, 0.90078571, 0.91735714, 0.936375  ,
        0.97403571, 0.95398214, 0.98426786, 0.95564286, 0.98666071,
        0.95464286]),
```

```
 'mean_train_score': array([0.78104286, 0.78123929, 0.7812    , 0.818803
57, 0.82720714,
        0.86631429, 0.87328214, 0.90054643, 0.917425  , 0.9356    ,
        0.97317143, 0.95697143, 0.98299286, 0.95890357, 0.98496071,
        0.95671071]),
 'std_train_score': array([8.74817765e-06, 2.90144229e-05, 6.12372436e-0
5, 6.22085039e-04,
        2.29684788e-04, 5.46043282e-04, 4.48125347e-04, 3.77896973e-04,
        4.04124147e-04, 7.85844145e-04, 5.68137846e-04, 6.21826227e-03,
        7.04902656e-04, 8.24690418e-03, 9.17196816e-04, 6.65179314e-0
3])}
```

In [44]:
```
f, axes = plt.subplots(1, 2 , figsize=(12,4))
for i in range(2):
    title ='log(C) - l1 hyperparameter' if i== 0 else 'log(C) - l2 hyperp
    train_auc = gridcv.cv_results_['mean_train_score'][0+i:16+i:2]
    test_auc = gridcv.cv_results_['mean_test_score'][0+i:16+i:2]
    axes[i].grid(True)
    axes[i].plot(np.log(C), train_auc, label = 'Train_AUC')
    axes[i].plot(np.log(C), test_auc, label = 'Test_AUC')
    axes[i].scatter(np.log(C), train_auc)
    axes[i].scatter(np.log(C), test_auc)
    axes[i].set_xlabel(title)
    axes[i].set_ylabel('AUC')
    axes[i].set_title('Error Plot')
    axes[i].legend()
plt.show()
```



In [48]:
```
import math
```

In [50]:
```
math.exp(-4)
```

Out[50]: 0.01831563888873418

In [52]:
```
math.exp(-6.5)
```

Out[52]: 0.0015034391929775724

In [54]:
```
math.exp(-2.2)
```

Out[54]: 0.11080315836233387

we can see that at C =0.1 and whith L2 regularizaion model showing good result/score on both test and traning dataset

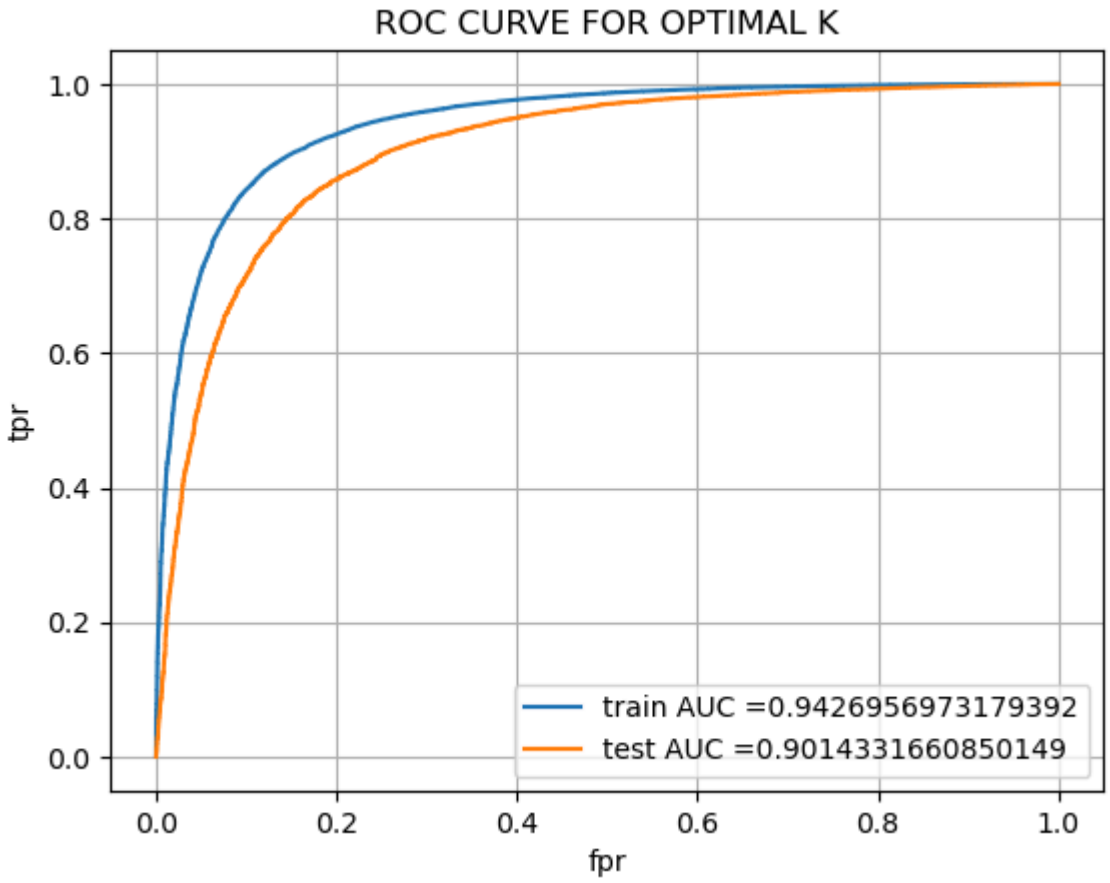
In [59]:
```
# making ROC Curve
```

In [156…
```
log_r = LogisticRegression(solver = 'liblinear', C = 0.1, penalty = 'l2'
```

In [158…
```
log_r = log_r.fit(X_train_bow, y_train)
```

In [160…
```
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, log_r.predict
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, log_r.predict_pro
```

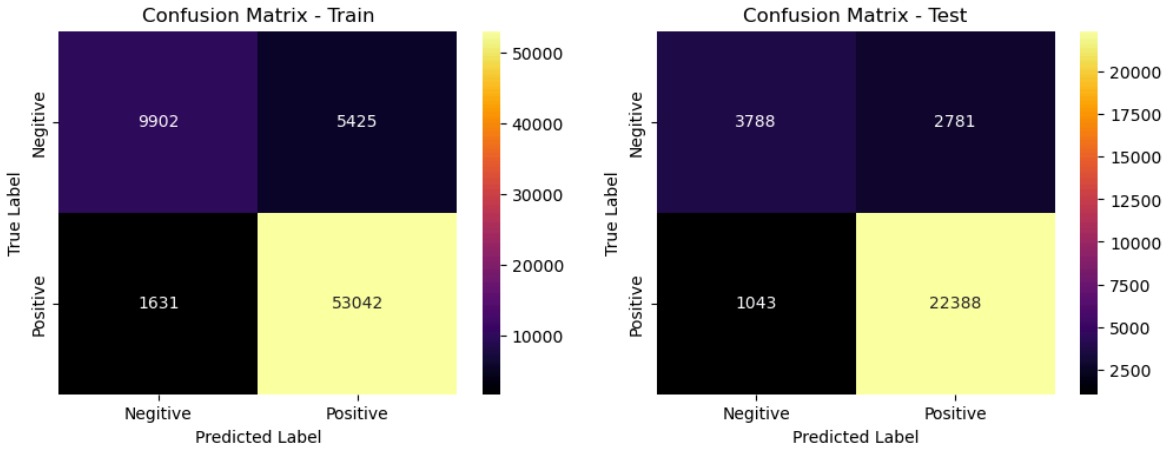
In [162…
```
plt.grid(True)
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, tra
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tp
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC CURVE FOR OPTIMAL K")
plt.show()
```



In [164…
```
from sklearn.metrics import confusion_matrix
# confusion matrix
def confusion_matrixes(model, X_train, y_train, X_test, y_test):
    train_cm = confusion_matrix(y_train,model.predict(X_train))
    test_cm = confusion_matrix(y_test, model.predict(X_test))
    col_n = ['Negitive','Positive']
    df_train = pd.DataFrame(train_cm,index = col_n, columns= col_n)
    df_test = pd.DataFrame(test_cm,index = col_n, columns= col_n)
    f, axes = plt.subplots(1,2,figsize= (12,4))

    for i in range(2):
        df = df_train if i == 0 else df_test
        sns.heatmap(df,annot = True, cmap = 'inferno', ax = axes[i], fmt
```

```
        axes[i].set_title(f"Confusion Matrix - {'Train' if i==0 else 'Tes
        axes[i].set_xlabel("Predicted Label")
        axes[i].set_ylabel("True Label")
    plt.show()
```

In [166…
```
confusion_matrixes(log_r, X_train_bow, y_train, X_test_bow, y_test)
```



In [168…
```
coff = log_r.coef_.reshape(-1,1)
df_imp = pd.DataFrame(coff,columns = ['coef'],index = bow_features)
print('positive words')
positive = df_imp.sort_values(by = 'coef', ascending= False).head(10)
print(positive)
print("="*50)
print('Negitive words')
negitive = df_imp.sort_values(by = 'coef').head(10)
print(negitive)
```

```
positive words
                coef
excellent   1.383414
delicious   1.266162
perfect     1.261638
highly      1.259622
beat        1.093022
hook        1.084167
wonderful   1.070364
amazing     1.064675
awesome     1.063231
yummy       1.032007
==================================================
Negitive words
                    coef
terrible       -1.586960
disappointing  -1.427603
awful          -1.400700
disappointment -1.395226
refund         -1.366344
horrible       -1.348827
disappointed   -1.305698
unfortunately  -1.263590
return         -1.134702
disappoint     -1.128305
```

we can see that logestic regression performing better then naive_Bayes

In [ ]:

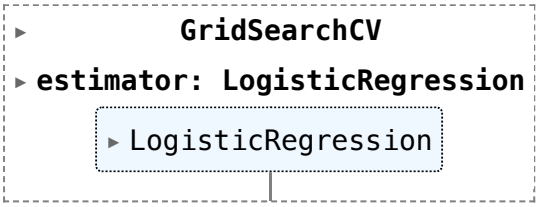
2. TF-IDF

In [200…
```
#loding data
```

In [142…
```
X_train_tfidf = pickle.load(open('tfidf_vectors/X_train_tfidf','rb'))
X_test_tfidf = pickle.load(open('tfidf_vectors/X_test_tfidf','rb'))
tfidf_features = pickle.load(open('tfidf_vectors/tfidf_features','rb'))
```

In [144…
```
penalty = ['l1', 'l2']
C = [0.0001,0.001,0.01,0.1,1,10,100,1000]
grid = {'penalty': penalty, 'C': C}
log_reg1 = LogisticRegression(solver = 'liblinear')
gridcv2 = GridSearchCV(log_reg1, grid, cv =5,return_train_score = True,n_
gridcv2.fit(X_train_tfidf,y_train)
```

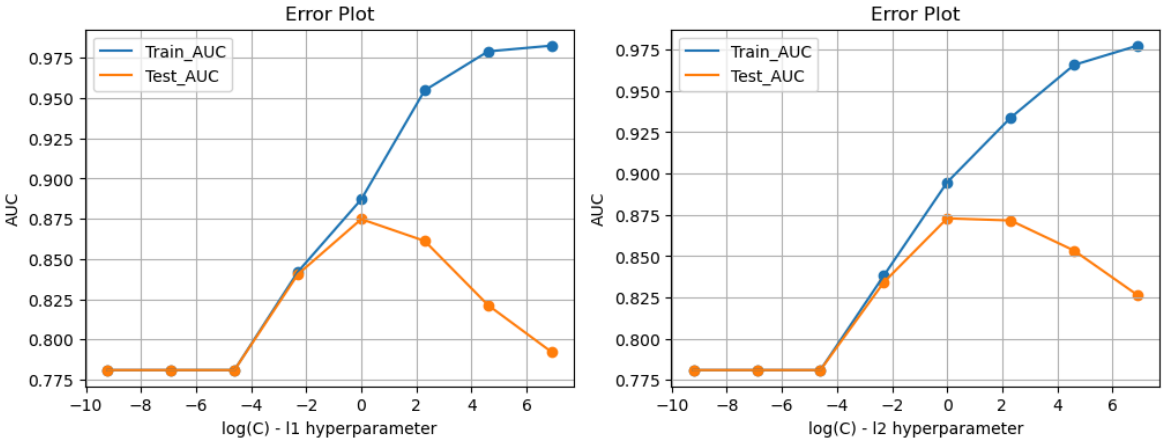
Out[144…



In [216…
```
print(gridcv2.best_params_)
```

```
{'C': 1, 'penalty': 'l1'}
```

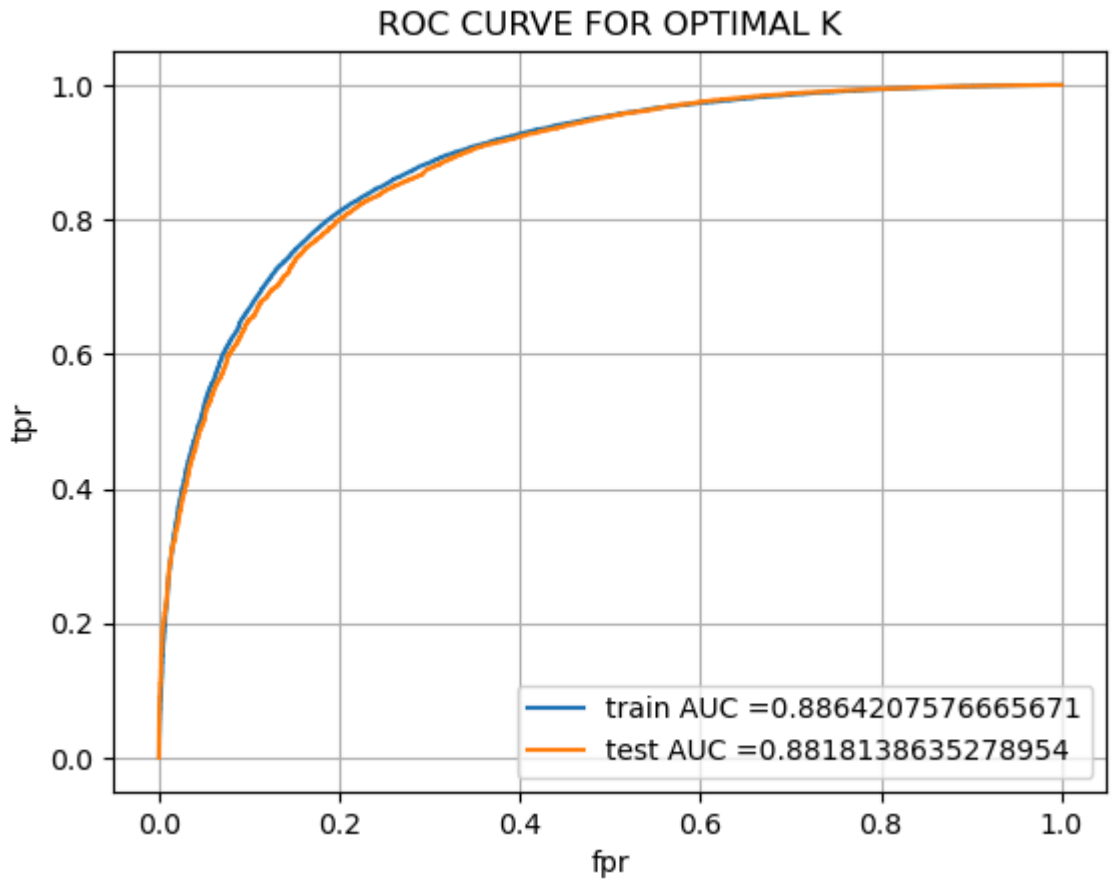
In [148…
```
f, axes = plt.subplots(1, 2 , figsize=(12,4))
for i in range(2):
    title ='log(C) - l1 hyperparameter' if i== 0 else 'log(C) - l2 hyperp
    train_auc_1 = gridcv2.cv_results_['mean_train_score'][0+i:16+i:2]
    test_auc_1 = gridcv2.cv_results_['mean_test_score'][0+i:16+i:2]
    axes[i].grid(True)
    axes[i].plot(np.log(C), train_auc_1, label = 'Train_AUC')
    axes[i].plot(np.log(C), test_auc_1, label = 'Test_AUC')
    axes[i].scatter(np.log(C), train_auc_1)
    axes[i].scatter(np.log(C), test_auc_1)
    axes[i].set_xlabel(title)
    axes[i].set_ylabel('AUC')
    axes[i].set_title('Error Plot')
    axes[i].legend()
plt.show()
```



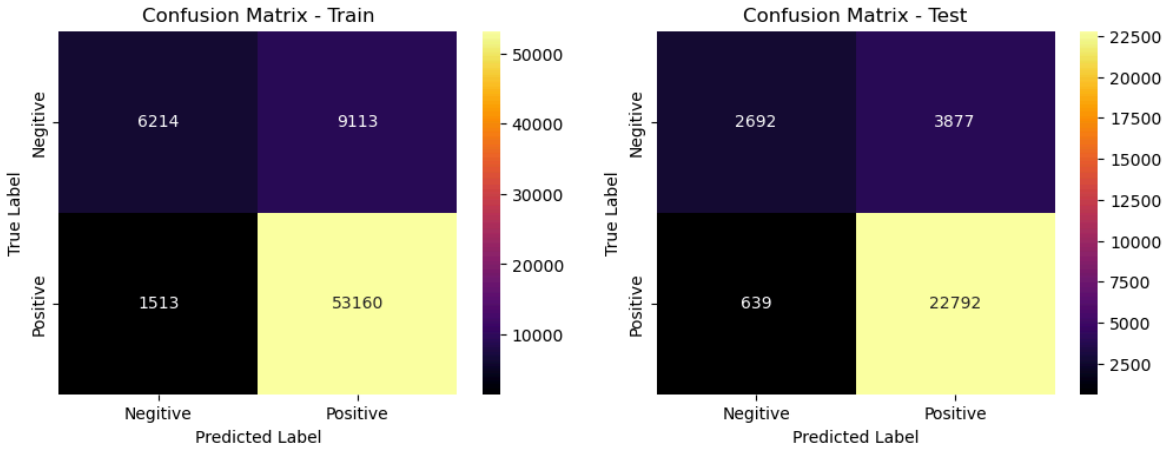
by both the graph we can see that L2 regularization is performing well by L1 may be has point age

In [251…

```
# roc curve

log_r1 = LogisticRegression(solver = 'liblinear', C = 0.1, penalty = 'l1'

log_r1 = log_r1.fit(X_train_tfidf, y_train)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, log_r1.predic
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, log_r1.predict_pr

plt.grid(True)
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, tra
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tp
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC CURVE FOR OPTIMAL K")
plt.show()
```



In [194…
```
confusion_matrixes(log_r1, X_train_tfidf, y_train, X_test_tfidf, y_test)
```



In [196…
```
coff = log_r1.coef_.reshape(-1,1)
df_imp = pd.DataFrame(coff,columns = ['coef'],index = tfidf_features)
print('positive words')
positive = df_imp.sort_values(by = 'coef', ascending= False).head(10)
print(positive)
print("="*50)
print('Negitive words')
negitive = df_imp.sort_values(by = 'coef').head(10)
print(negitive)
```

```
positive words
                coef
great      10.819319
delicious   8.866541
perfect     8.473113
love        8.393851
excellent   7.175173
highly      6.619821
wonderful   6.092787
good        5.918099
favorite    5.039743
easy        4.558367
==================================================
Negitive words
                   coef
bad           -6.538950
terrible      -6.138030
return        -6.068764
awful         -5.844612
unfortunately -5.522237
ok            -5.404544
horrible      -5.374753
disappointed  -5.235962
disappoint    -5.093102
waste         -4.622711
```

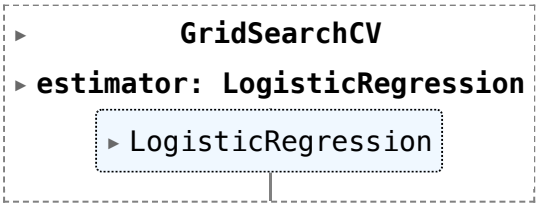
3. Avg Word2Vector

In [204…
```
# loading dataset
```

In [206…
```
X_train_avgw2v = pickle.load(open('w2v/avgw2v_train','rb'))
X_test_avgw2v = pickle.load(open('w2v/avgw2v_test','rb'))
w2v_words = pickle.load(open('w2v/w2v_words','rb'))
```

In [208…
```
penalty = ['l1', 'l2']
C = [0.0001,0.001,0.01,0.1,1,10,100,1000]
grid = {'penalty': penalty, 'C': C}
log_reg2 = LogisticRegression(solver = 'liblinear')
gridcv3 = GridSearchCV(log_reg2, grid, cv =5,return_train_score = True,n_
gridcv3.fit(X_train_avgw2v,y_train)
```

Out[208…

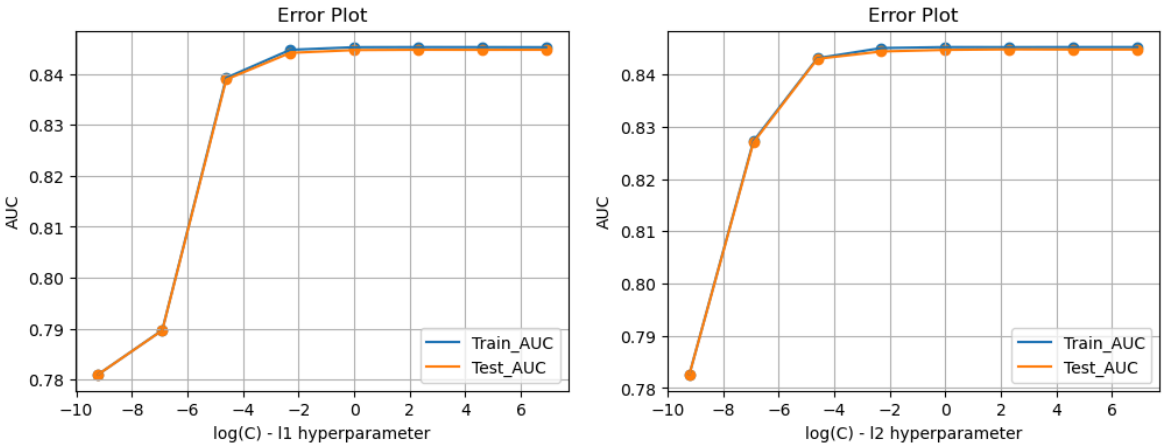


In [212…
```
print(gridcv3.best_params_)
```

```
{'C': 10, 'penalty': 'l2'}
```

In [218…
```
f, axes = plt.subplots(1, 2 , figsize=(12,4))
for i in range(2):
    title ='log(C) - l1 hyperparameter' if i== 0 else 'log(C) - l2 hyperp
    train_auc_2 = gridcv3.cv_results_['mean_train_score'][0+i:16+i:2]
    test_auc_2 = gridcv3.cv_results_['mean_test_score'][0+i:16+i:2]
    axes[i].grid(True)
```

```
    axes[i].plot(np.log(C), train_auc_2, label = 'Train_AUC')
    axes[i].plot(np.log(C), test_auc_2, label = 'Test_AUC')
    axes[i].scatter(np.log(C), train_auc_2)
    axes[i].scatter(np.log(C), test_auc_2)
    axes[i].set_xlabel(title)
    axes[i].set_ylabel('AUC')
    axes[i].set_title('Error Plot')
    axes[i].legend()
plt.show()
```

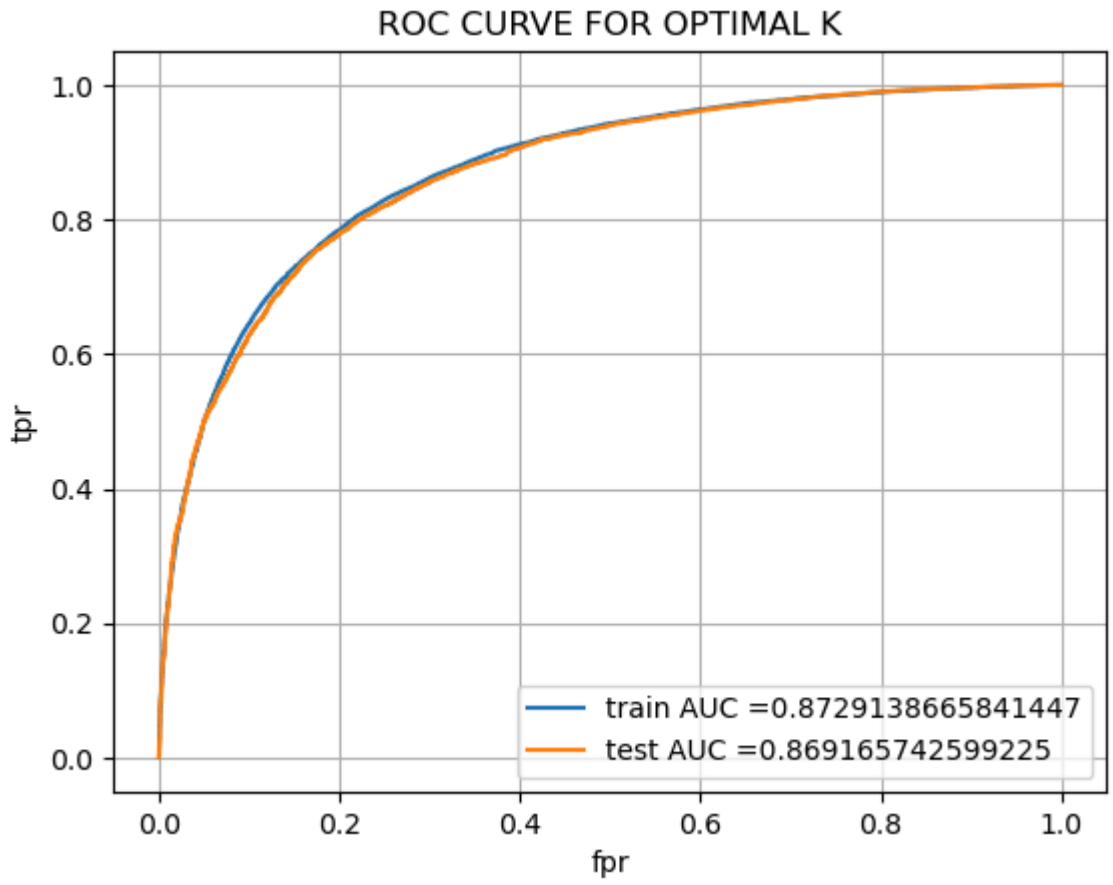


In [229…
```
math.exp(2.3)
```

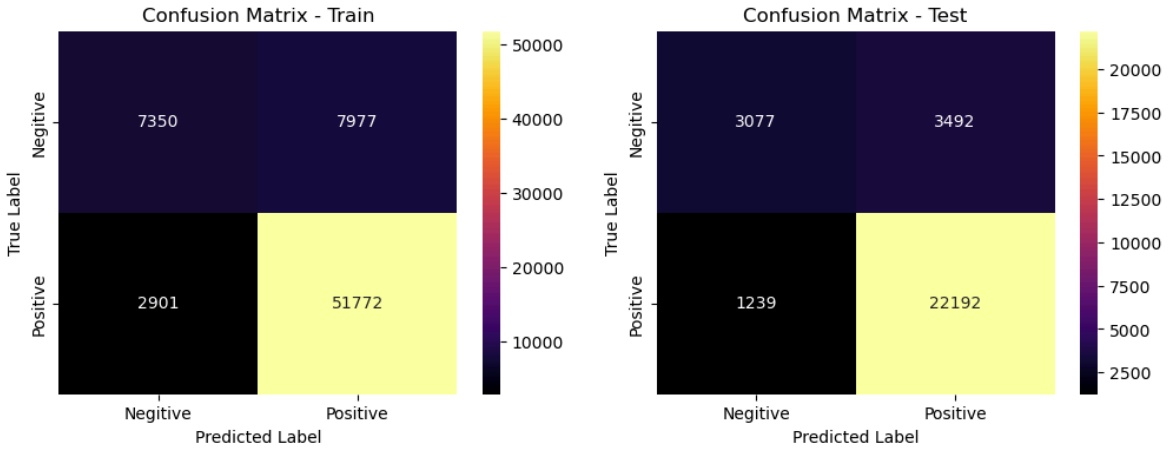
Out[229… 9.974182454814718

here we see that model with hyperparameter of L2 reg have good AUC for both traing and testing data

In [240…
```
# roc curve

log_r2 = LogisticRegression(solver = 'liblinear', C = 10, penalty = 'l2'

log_r2 = log_r1.fit(X_train_avgw2v, y_train)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, log_r2.predic
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, log_r2.predict_pr

plt.grid(True)
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, tra
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tp
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC CURVE FOR OPTIMAL K")
plt.show()
```



In [242…
```
confusion_matrixes(log_r2, X_train_avgw2v, y_train, X_test_avgw2v, y_test
```



Observation and Calculation

In [249…
```
from prettytable import PrettyTable
```

In [255…
```
z = PrettyTable()
z.field_names = ["vector","Algorithm","regulariazation","Hyperparameter(C
z.add_row(['BOW','Logistic Regression','l2',0.1,0.94,0.90])
z.add_row(['TFIDF','Logistic Regression','l1',1,0.886,0.881])
z.add_row(['Average W2v','Logistic Regression','l2',10,0.87,0.86])
```

In [257…
```
print(z)
```

```
+-------------+---------------------+-----------------+-------------------
+-----------+----------+
|    vector   |      Algorithm      | regulariazation | Hyperparameter(C)
| Train-AUC | Test-AUC |
+-------------+---------------------+-----------------+-------------------
+-----------+----------+
|     BOW     | Logistic Regression |        l2       |        0.1
|    0.94   |   0.9    |
|    TFIDF    | Logistic Regression |        l1       |         1
|   0.886   |  0.881   |
| Average W2v | Logistic Regression |        l2       |         10
|    0.87   |   0.86   |
+-------------+---------------------+-----------------+-------------------
+-----------+----------+
```

In [ ]:

3. Decision Tree

In [4]:
```
import numpy as np
import pandas as pd
import sklearn
import pickle
import sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve,auc
```

In [6]:
```
# loading Vectors
```

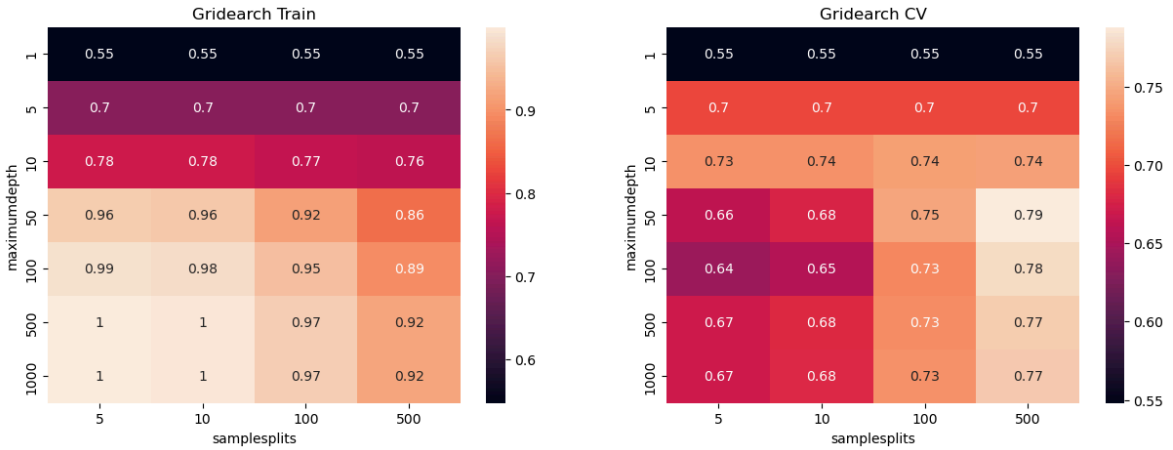
In [ ]:
```
# bow vectors
```

In [8]:
```
X_train_bow = pickle.load(open('bow_vectors/X_train_bow','rb'))
X_test_bow = pickle.load(open('bow_vectors/X_test_bow','rb'))
bow_features = pickle.load(open('bow_vectors/bow_features','rb'))
y_train = np.load('split dataset/y_train.npy',allow_pickle = True)
y_test = np.load('split dataset/y_test.npy',allow_pickle = True)
```

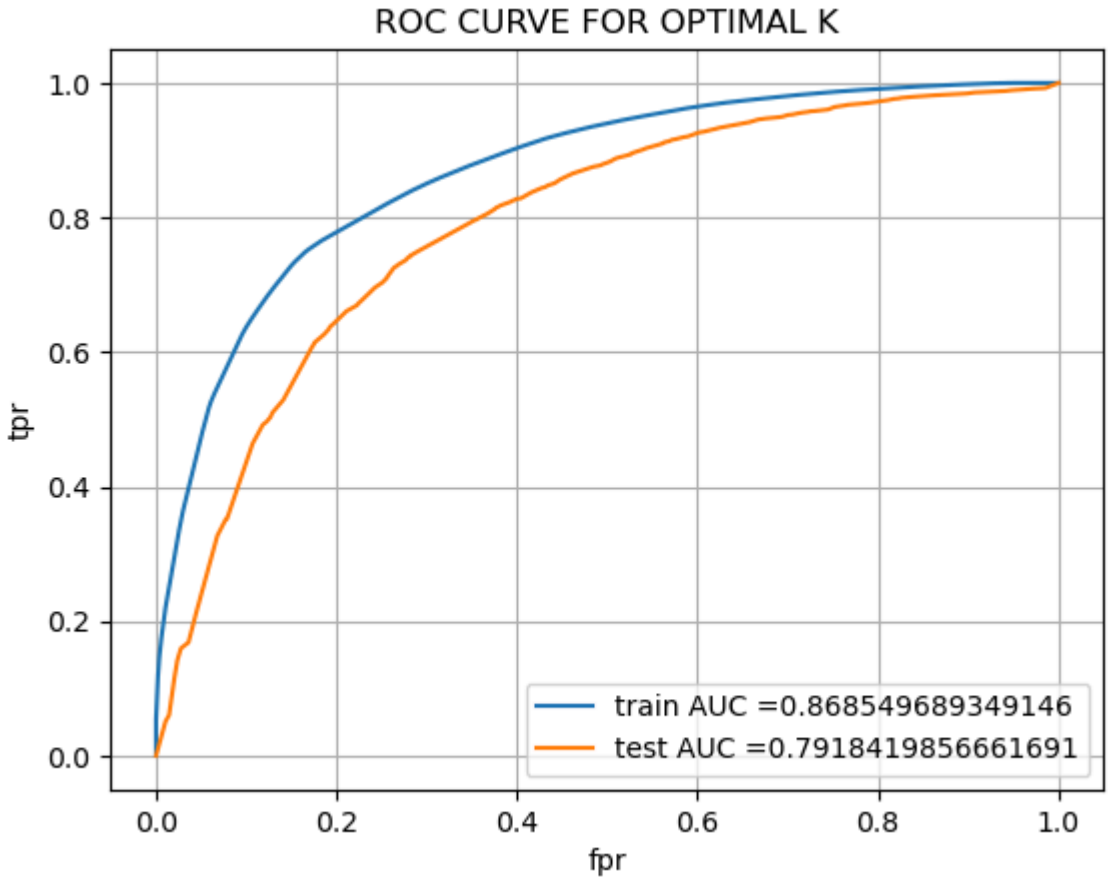
1. BoW (bag of Words)

In [15]:
```
dtr = DecisionTreeClassifier()
samplesplits = [5, 10, 100, 500]
maximumdepth = [1, 5, 10, 50, 100, 500, 1000]
parameters = {'min_samples_split':samplesplits ,'max_depth':maximumdepth}
model = GridSearchCV(estimator=dtr, param_grid=parameters, cv=3, n_jobs=-
model.fit(X_train_bow,y_train)
print("Model with best parameters :\n",model.best_params_)
train_auc = model.cv_results_['mean_train_score'].reshape(7,4)
cv_auc = model.cv_results_['mean_test_score'].reshape(7,4)

f, axes = plt.subplots(1, 2,figsize=(15,5))

for i in range(2):
  title = train_auc if i == 0 else cv_auc
  sns.heatmap(title,xticklabels=samplesplits,yticklabels=maximumdepth,ann
  axes[i].set_title(f"Gridearch {'Train' if i==0 else 'CV'}")
  axes[i].set_ylabel("maximumdepth")
  axes[i].set_xlabel("samplesplits")


plt.show()
```

```
Model with best parameters :
 {'max_depth': 50, 'min_samples_split': 500}
```



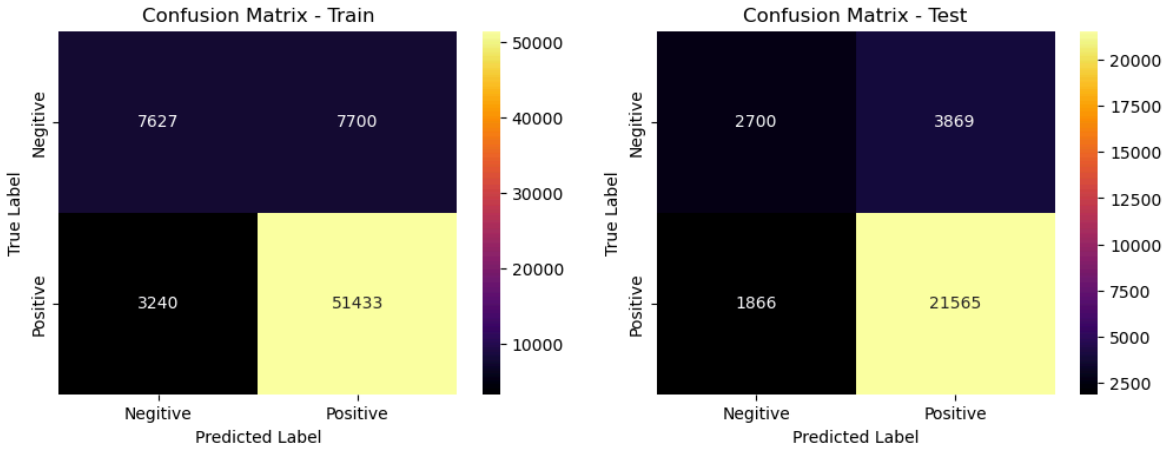
In [21]:
```
dtr_bp = DecisionTreeClassifier(max_depth = 50, min_samples_split = 500)
dtr_bp = dtr_bp.fit(X_train_bow,y_train)
train_fpr, train_tpr, thresholds = roc_curve(y_train, dtr_bp.predict_prob
test_fpr, test_tpr, thresholds = roc_curve(y_test, dtr_bp.predict_proba(X

plt.grid(True)
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, tra
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tp
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC CURVE FOR OPTIMAL K")
plt.show()

#Area under ROC curve
print('Area under train roc {}'.format(auc(train_fpr, train_tpr)))
print('Area under test roc {}'.format(auc(test_fpr, test_tpr)))
```



```
Area under train roc 0.868549689349146
Area under test roc 0.7918419856661691
```

In [23]:
```
from sklearn.metrics import confusion_matrix
# confusion matrix
def confusion_matrixes(model, X_train, y_train, X_test, y_test):
    train_cm = confusion_matrix(y_train,model.predict(X_train))
    test_cm = confusion_matrix(y_test, model.predict(X_test))
    col_n = ['Negitive','Positive']
    df_train = pd.DataFrame(train_cm,index = col_n, columns= col_n)
    df_test = pd.DataFrame(test_cm,index = col_n, columns= col_n)
    f, axes = plt.subplots(1,2,figsize= (12,4))

    for i in range(2):
        df = df_train if i == 0 else df_test
        sns.heatmap(df,annot = True, cmap = 'inferno', ax = axes[i], fmt
        axes[i].set_title(f"Confusion Matrix - {'Train' if i==0 else 'Tes
        axes[i].set_xlabel("Predicted Label")
        axes[i].set_ylabel("True Label")
    plt.show()
```

In [27]:
```
confusion_matrixes(dtr_bp, X_train_bow, y_train, X_test_bow, y_test)
```



In [ ]:
```
```

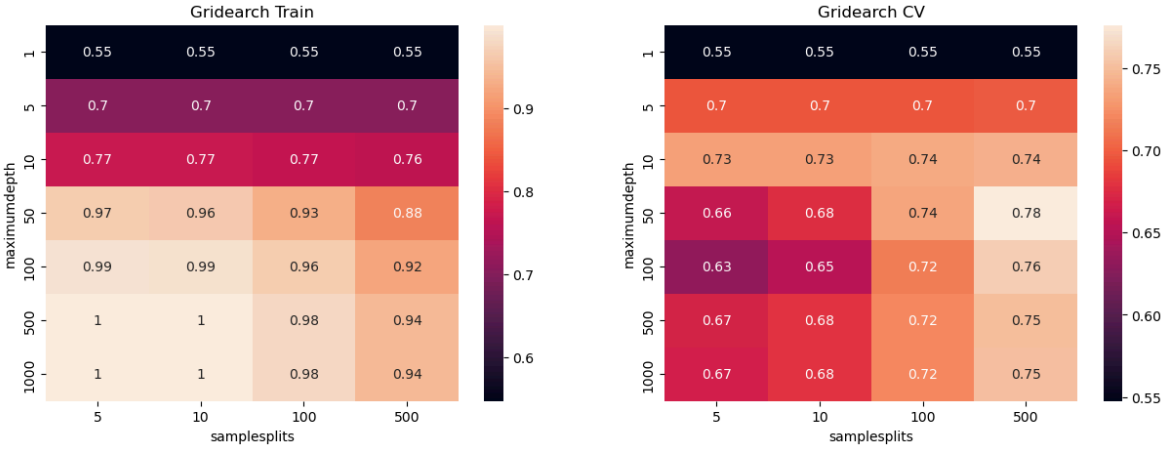
2. TF-IDF

In [30]:
```
X_train_tfidf = pickle.load(open('tfidf_vectors/X_train_tfidf','rb'))
X_test_tfidf = pickle.load(open('tfidf_vectors/X_test_tfidf','rb'))
tfidf_features = pickle.load(open('tfidf_vectors/tfidf_features','rb'))
```

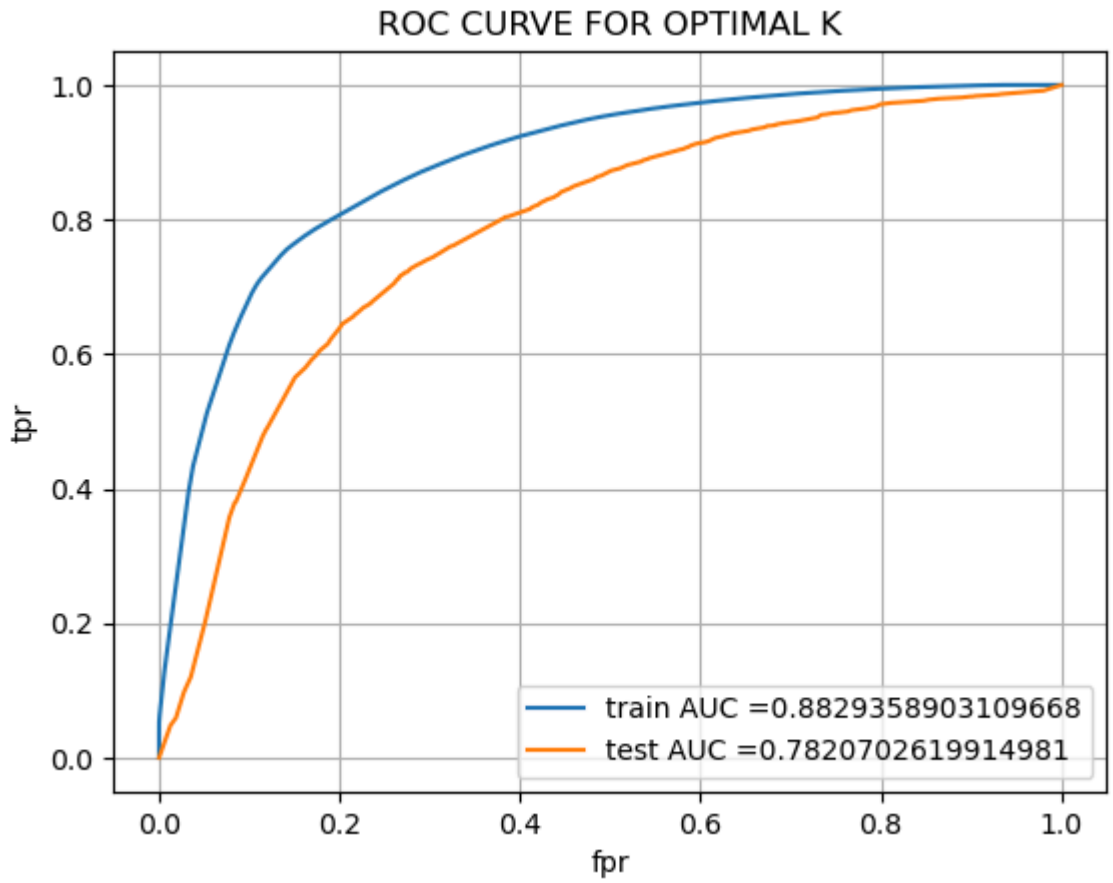
In [32]:
```
dtr2 = DecisionTreeClassifier()
samplesplits = [5, 10, 100, 500]
maximumdepth = [1, 5, 10, 50, 100, 500, 1000]
parameters = {'min_samples_split':samplesplits ,'max_depth':maximumdepth}
model = GridSearchCV(estimator=dtr2, param_grid=parameters, cv=3, n_jobs=
model.fit(X_train_tfidf,y_train)
print("Model with best parameters :\n",model.best_params_)
train_auc = model.cv_results_['mean_train_score'].reshape(7,4)
cv_auc = model.cv_results_['mean_test_score'].reshape(7,4)

f, axes = plt.subplots(1, 2,figsize=(15,5))
```

```
for i in range(2):
  title = train_auc if i == 0 else cv_auc
  sns.heatmap(title,xticklabels=samplesplits,yticklabels=maximumdepth,ann
  axes[i].set_title(f"Gridearch {'Train' if i==0 else 'CV'}")
  axes[i].set_ylabel("maximumdepth")
  axes[i].set_xlabel("samplesplits")


plt.show()
```

```
Model with best parameters :
 {'max_depth': 50, 'min_samples_split': 500}
```

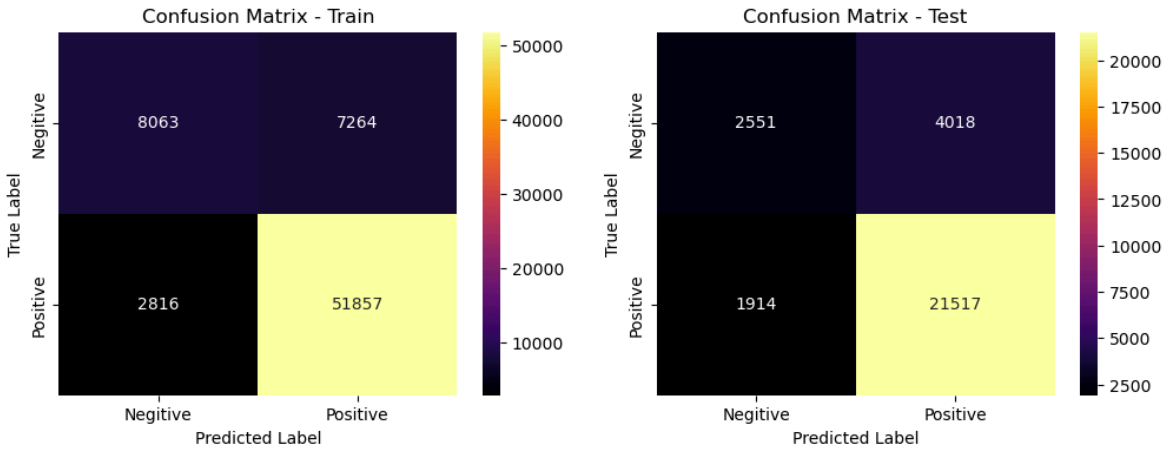


In [39]:
```
dtr_bp2 = DecisionTreeClassifier(max_depth = 50, min_samples_split = 500)
dtr_bp2 = dtr_bp2.fit(X_train_tfidf,y_train)
train_fpr, train_tpr, thresholds = roc_curve(y_train, dtr_bp2.predict_pro
test_fpr, test_tpr, thresholds = roc_curve(y_test, dtr_bp2.predict_proba(

plt.grid(True)
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, tra
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tp
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC CURVE FOR OPTIMAL K")
plt.show()

#Area under ROC curve
print('Area under train roc {}'.format(auc(train_fpr, train_tpr)))
print('Area under test roc {}'.format(auc(test_fpr, test_tpr)))
```




```
Area under train roc 0.8829358903109668
Area under test roc 0.7820702619914981
```

In [41]:
```
confusion_matrixes(dtr_bp2, X_train_tfidf, y_train, X_test_tfidf, y_test)
```



In [ ]:
```
```

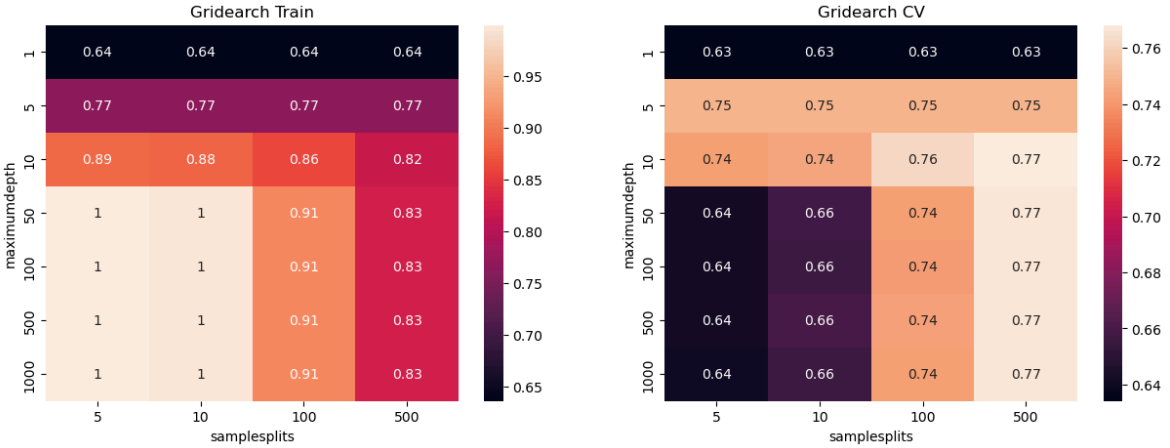
3. Average Word2Vector

In [44]:
```
X_train_avgw2v = pickle.load(open('w2v/avgw2v_train','rb'))
X_test_avgw2v = pickle.load(open('w2v/avgw2v_test','rb'))
w2v_words = pickle.load(open('w2v/w2v_words','rb'))
```

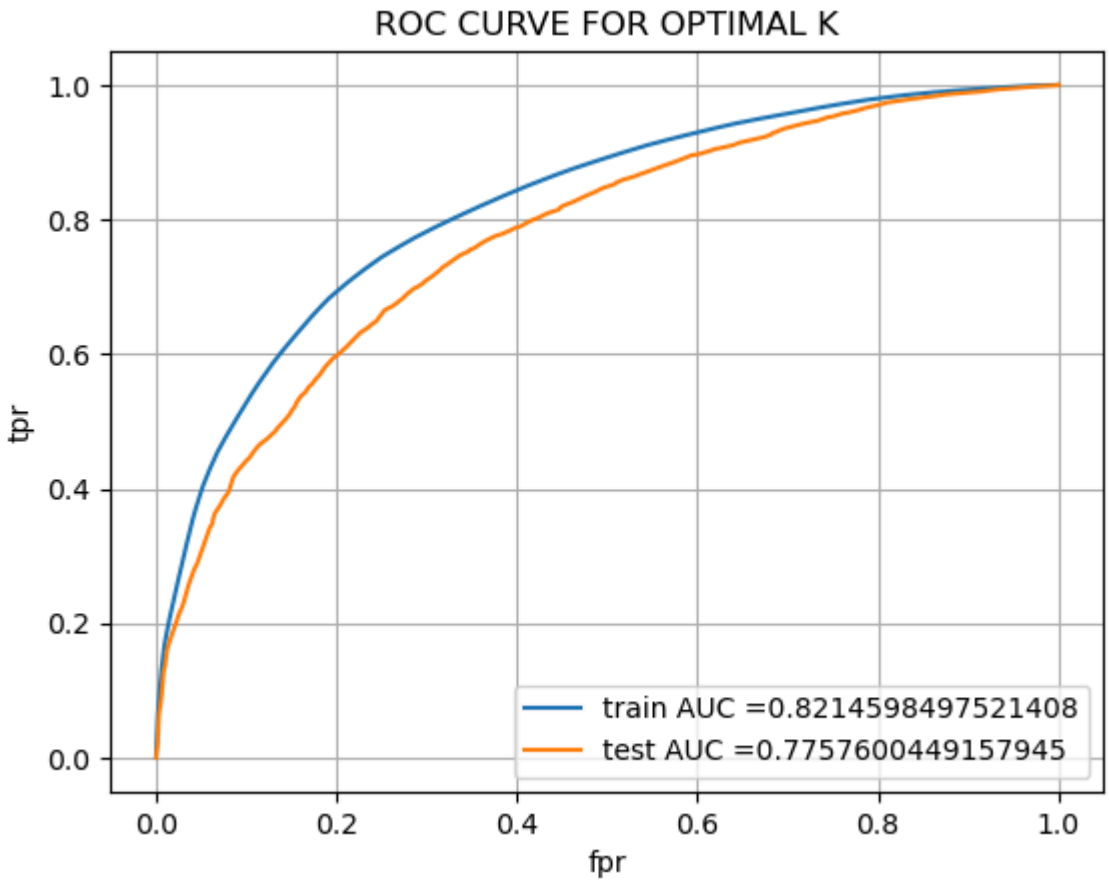
In [46]:
```
dtr3 = DecisionTreeClassifier()
samplesplits = [5, 10, 100, 500]
maximumdepth = [1, 5, 10, 50, 100, 500, 1000]
parameters = {'min_samples_split':samplesplits ,'max_depth':maximumdepth}
```

```
model = GridSearchCV(estimator=clf, param_grid=parameters, cv=3, n_jobs=-
model.fit(X_train_avgw2v,y_train)
print("Model with best parameters :\n",model.best_params_)
train_auc = model.cv_results_['mean_train_score'].reshape(7,4)
cv_auc = model.cv_results_['mean_test_score'].reshape(7,4)

f, axes = plt.subplots(1, 2,figsize=(15,5))

for i in range(2):
  title = train_auc if i == 0 else cv_auc
  sns.heatmap(title,xticklabels=samplesplits,yticklabels=maximumdepth,ann
  axes[i].set_title(f"Gridearch {'Train' if i==0 else 'CV'}")
  axes[i].set_ylabel("maximumdepth")
  axes[i].set_xlabel("samplesplits")


plt.show()
```

```
Model with best parameters :
 {'max_depth': 10, 'min_samples_split': 500}
```



In [50]:
```
dtr_bp3 = DecisionTreeClassifier(**model.best_params_)
dtr_bp3 = dtr_bp3.fit(X_train_avgw2v,y_train)
train_fpr, train_tpr, thresholds = roc_curve(y_train, dtr_bp3.predict_pro
test_fpr, test_tpr, thresholds = roc_curve(y_test, dtr_bp3.predict_proba(

plt.grid(True)
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, tra
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tp
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC CURVE FOR OPTIMAL K")
plt.show()

#Area under ROC curve
print('Area under train roc {}'.format(auc(train_fpr, train_tpr)))
print('Area under test roc {}'.format(auc(test_fpr, test_tpr)))
```




```
Area under train roc 0.8214598497521408
Area under test roc 0.7757600449157945
```

In [ ]:

Observation

In [55]:
```
from prettytable import PrettyTable
```

In [57]:
```
z = PrettyTable()

z.field_names = ["Vector","Algorithm","Hyperparam-min_sample_splits","Hyp
z.add_row(["bow","decision_tree",500,50, 0.86,0.79])
z.add_row(["tfidf","decision_tree",500,50,0.88, 0.78 ])
z.add_row(["avgw2v","decision_tree",500,10, 0.82,0.77])
print(z)
```

```
+--------+---------------+-----------------------------+-----------------
-----+-----------+----------+
| Vector |   Algorithm   | Hyperparam-min_sample_splits | Hyperparam-max_D
epth | Train AUC | Test AUC |
+--------+---------------+-----------------------------+-----------------
-----+-----------+----------+
|  bow   | decision_tree |             500             |          50
|    0.86   |   0.79   |
| tfidf  | decision_tree |             500             |          50
|    0.88   |   0.78   |
| avgw2v | decision_tree |             500             |          10
|    0.82   |   0.77   |
+--------+---------------+-----------------------------+-----------------
-----+-----------+----------+
```

we can see that all are performing same

Random Forest Classifier

In [86]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

import sklearn
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import roc_auc_score,roc_curve,auc
from sklearn.metrics import confusion_matrix


import pickle
```

1. BOW

In [15]:
```
# loading BOW vectors

X_train_bow = pickle.load(open('bow_vectors/X_train_bow','rb'))
X_test_bow = pickle.load(open('bow_vectors/X_test_bow','rb'))
bow_features = pickle.load(open('bow_vectors/bow_features','rb'))

y_train = np.load('split dataset/y_train.npy',allow_pickle = True)
y_test = np.load('split dataset/y_test.npy',allow_pickle = True)
```

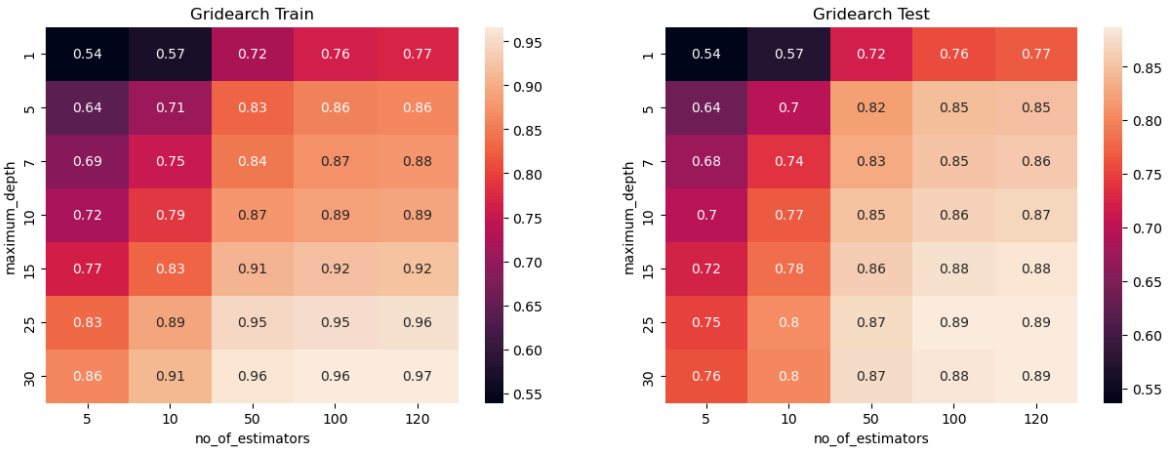
In [21]:
```
rfr = RandomForestClassifier(n_jobs = -1, class_weight = 'balanced')
num_estimator = [5,10,50,100,120]
maximum_depth = [1, 5, 7, 10, 15, 25, 30]
parameter = {
    'n_estimators': num_estimator,
    'max_depth': maximum_depth
}
grid = GridSearchCV(estimator = rfr , param_grid = parameter , cv =5, n_j
grid.fit(X_train_bow,y_train)

print('Best Parameter:',grid.best_params_)
```

```
Best Parameter: {'max_depth': 25, 'n_estimators': 120}
```

In [49]:
```
train_auc = grid.cv_results_['mean_train_score'].reshape(7,5)
test_auc = grid.cv_results_['mean_test_score'].reshape(7,5)
```

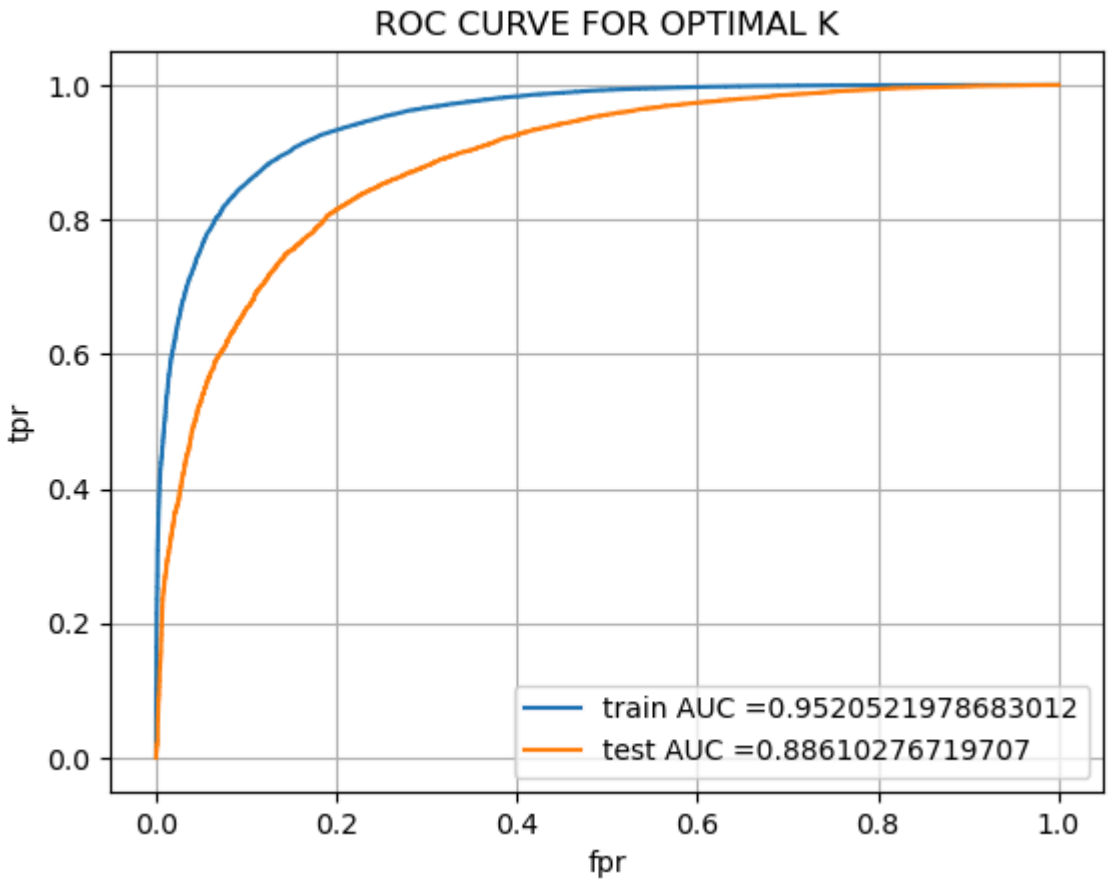
In [59]:
```
f, axes = plt.subplots(1, 2,figsize=(15,5))

for i in range(2):
    title = train_auc if i ==0 else test_auc
    sns.heatmap(title,xticklabels = num_estimator, yticklabels = maximum_
    axes[i].set_title(f"Gridearch {'Train' if i==0 else 'Test'}")
```

```
    axes[i].set_xlabel("no_of_estimators")
    axes[i].set_ylabel("maximum_depth")
```



we can see here at maximum depth = 15 and num of estimator = 120 has optimal score on both train and test

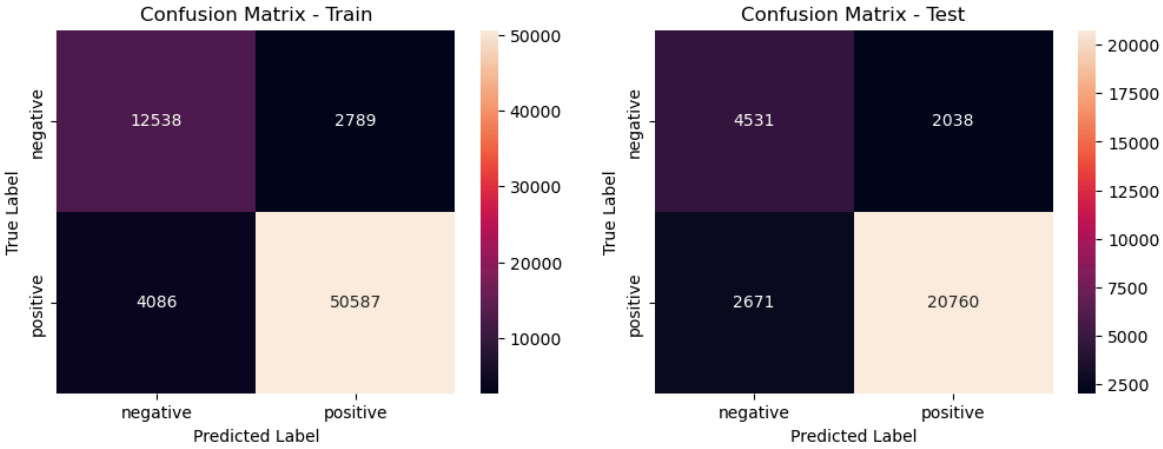
In [74]:
```
rfr_bp = RandomForestClassifier(**grid.best_params_,class_weight = 'balan
rfr_bp = rfr_bp.fit(X_train_bow,y_train)
train_fpr, train_tpr, thresholds = roc_curve(y_train, rfr_bp.predict_prob
test_fpr, test_tpr, thresholds = roc_curve(y_test, rfr_bp.predict_proba(X
plt.grid(True)
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, tra
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tp
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC CURVE FOR OPTIMAL K")
plt.show()

#Area under ROC curve
print('Area under train roc {}'.format(auc(train_fpr, train_tpr)))
print('Area under test roc {}'.format(auc(test_fpr, test_tpr)))
```




```
Area under train roc 0.9520521978683012
Area under test roc 0.88610276719707
```

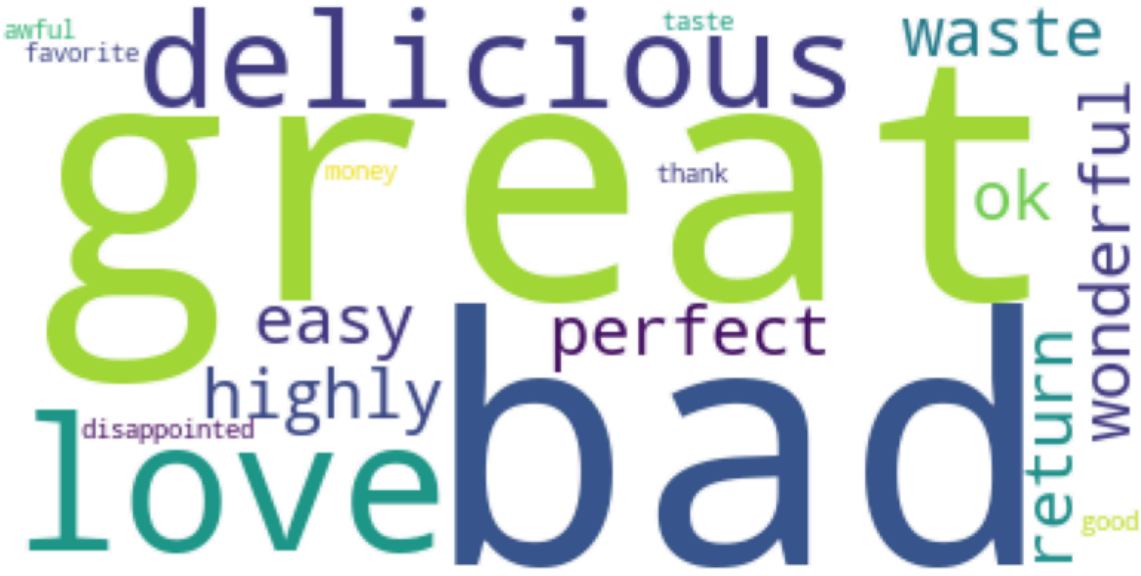
In [76]:
```
# helper function to plot confusion matrix
def plot_confusion_matrixes(model,x_train,y_train,x_test,y_test):
    cm_train = confusion_matrix(y_train,model.predict(x_train))
    cm_test =  confusion_matrix(y_test,model.predict(x_test))
    class_label = ["negative", "positive"]
    df_train = pd.DataFrame(cm_train, index = class_label, columns = clas
    df_test = pd.DataFrame(cm_test, index = class_label, columns = class_
    f, axes = plt.subplots(1, 2,figsize=(12,4))

    for i in range(2):
      df = df_train if i==0 else df_test
      sns.heatmap(df, annot = True, fmt = "d",ax=axes[i])
      axes[i].set_title(f"Confusion Matrix - {'Train' if i==0 else 'Test'
      axes[i].set_xlabel("Predicted Label")
      axes[i].set_ylabel("True Label")
    plt.show()
```

In [88]:
```
plot_confusion_matrixes(rfr_bp,X_train_bow,y_train,X_test_bow,y_test)
```



In [98]:
```
from wordcloud import WordCloud
df = rfr_bp.feature_importances_
features = bow_features
df = pd.DataFrame(df,columns=['coef'],index=features)
top = df.sort_values(by='coef',ascending=False).head(20)
print('Top 20 features are: \n {}'.format(top))
top['words'] = top.index
top.reset_index(drop=True)
sent = top.words.str.cat(sep=' ')
#word cloud representation
wordcloud = WordCloud(background_color='white').generate(sent)
plt.figure(figsize=(10,10))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```

```
Top 20 features are:
                   coef
great          0.030292
bad            0.023931
love           0.021625
would          0.016574
delicious      0.015791
perfect        0.013016
highly         0.009805
waste          0.008276
return         0.008178
ok             0.007946
wonderful      0.007701
easy           0.007651
disappointed   0.007308
however        0.007187
money          0.006822
thank          0.006370
good           0.006229
taste          0.005988
awful          0.005961
favorite       0.005865
```



2. TFIDF

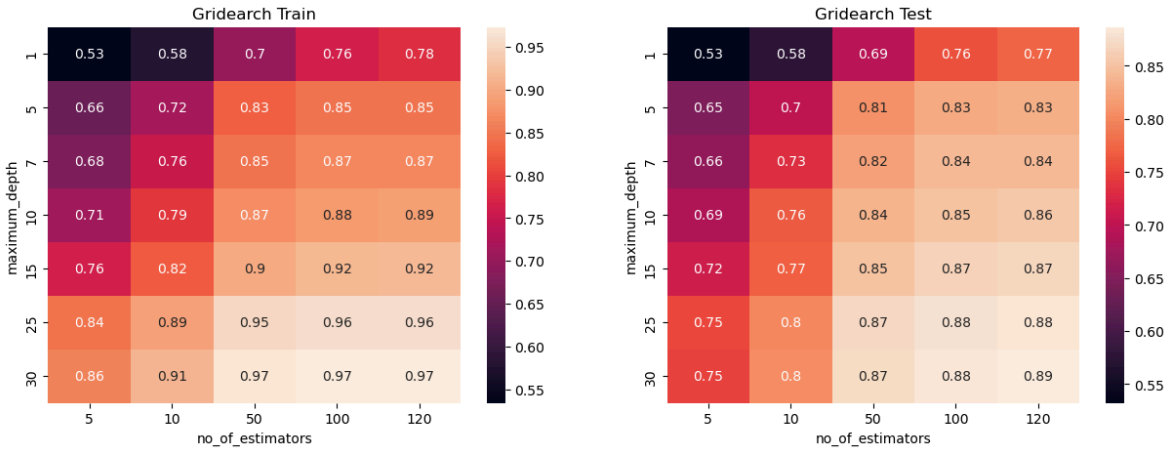
In [101…
```
X_train_tfidf = pickle.load(open('tfidf_vectors/X_train_tfidf','rb'))
X_test_tfidf = pickle.load(open('tfidf_vectors/X_test_tfidf','rb'))
tfidf_features = pickle.load(open('tfidf_vectors/tfidf_features','rb'))
```

In [103…
```
rfr2 = RandomForestClassifier(n_jobs = -1, class_weight = 'balanced')
num_estimator = [5,10,50,100,120]
maximum_depth = [1, 5, 7, 10, 15, 25, 30]
parameter = {
    'n_estimators': num_estimator,
    'max_depth': maximum_depth
}
grid2 = GridSearchCV(estimator = rfr2, param_grid = parameter , cv =5, n_
grid2.fit(X_train_tfidf,y_train)

print('Best Parameter:',grid.best_params_)
```

```
Best Parameter: {'max_depth': 25, 'n_estimators': 120}
```

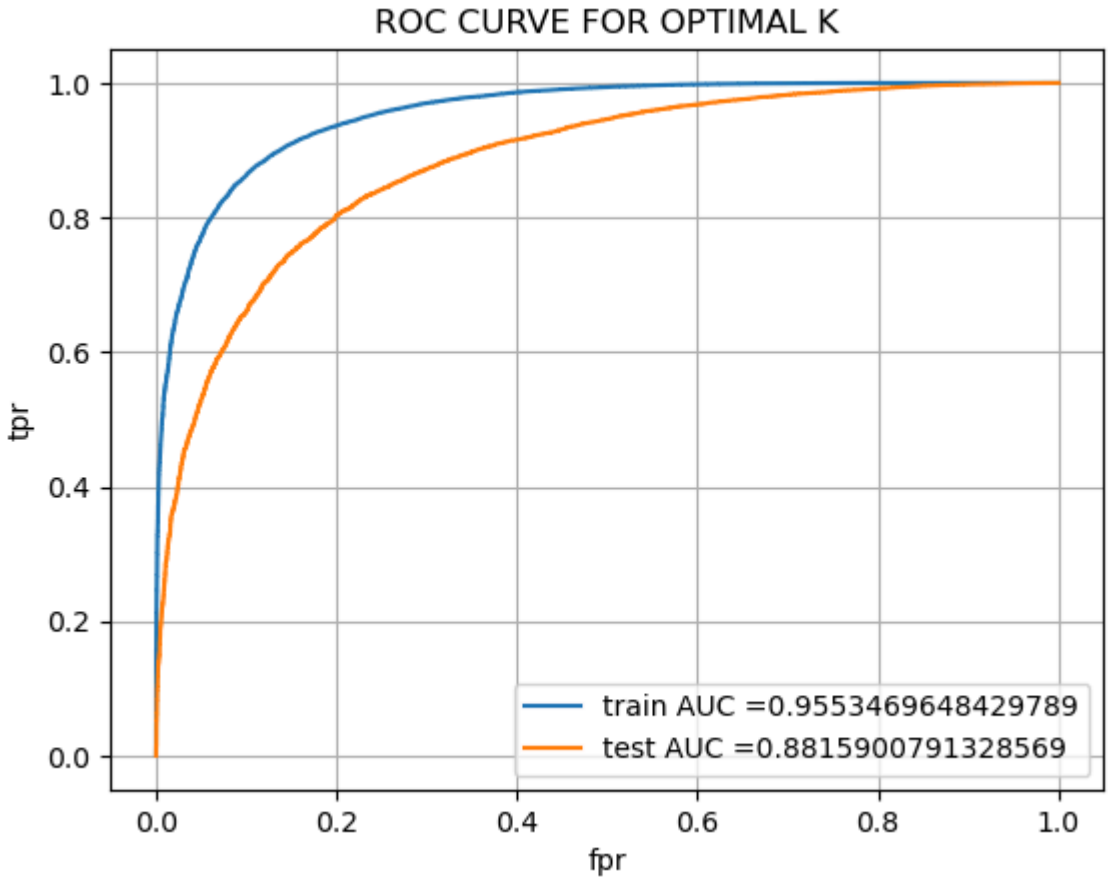
In [107…
```
train_auc = grid2.cv_results_['mean_train_score'].reshape(7,5)
test_auc = grid2.cv_results_['mean_test_score'].reshape(7,5)
```

In [109…
```
f, axes = plt.subplots(1, 2,figsize=(15,5))

for i in range(2):
    title = train_auc if i ==0 else test_auc
    sns.heatmap(title,xticklabels = num_estimator, yticklabels = maximum_
    axes[i].set_title(f"Gridearch {'Train' if i==0 else 'Test'}")
    axes[i].set_xlabel("no_of_estimators")
    axes[i].set_ylabel("maximum_depth")
```



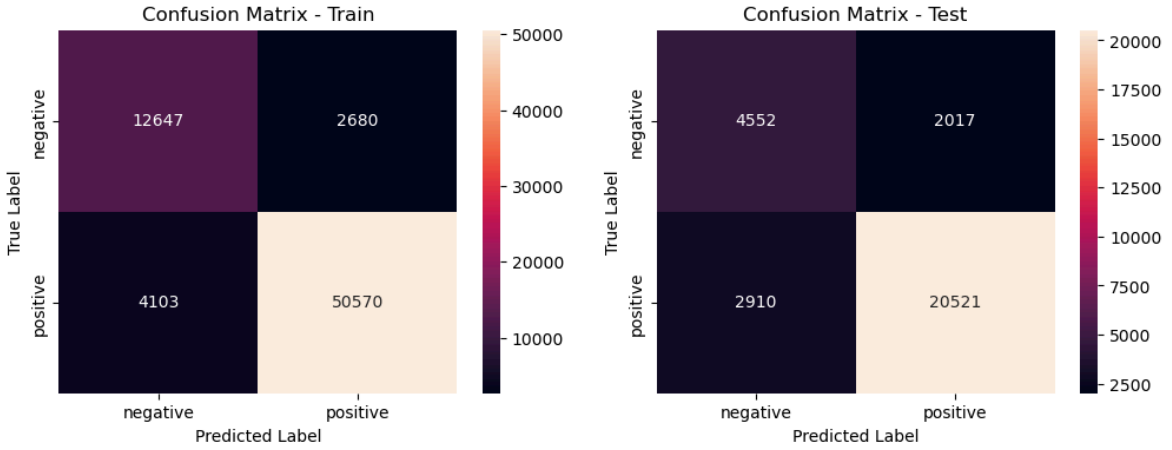
In [115…

```
rfr_bp2 = RandomForestClassifier(**grid.best_params_,class_weight = 'bala
rfr_bp2 = rfr_bp2.fit(X_train_tfidf,y_train)
train_fpr, train_tpr, thresholds = roc_curve(y_train, rfr_bp2.predict_pro
test_fpr, test_tpr, thresholds = roc_curve(y_test, rfr_bp2.predict_proba(
plt.grid(True)
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, tra
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tp
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC CURVE FOR OPTIMAL K")
plt.show()

#Area under ROC curve
print('Area under train roc {}'.format(auc(train_fpr, train_tpr)))
print('Area under test roc {}'.format(auc(test_fpr, test_tpr)))
```




```
Area under train roc 0.9553469648429789
Area under test roc 0.8815900791328569
```

In [117…
```
plot_confusion_matrixes(rfr_bp2,X_train_tfidf,y_train,X_test_tfidf,y_test
```



In [121…
```
from wordcloud import WordCloud
df = rfr_bp2.feature_importances_
features = bow_features
df = pd.DataFrame(df,columns=['coef'],index=features)
top = df.sort_values(by='coef',ascending=False).head(20)
print('Top 20 features are: \n {}'.format(top))
top['words'] = top.index
top.reset_index(drop=True)
sent = top.words.str.cat(sep=' ')
#word cloud representation
wordcloud = WordCloud(background_color='white').generate(sent)
plt.figure(figsize=(10,10))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```

```
Top 20 features are:
                coef
great       0.035407
bad         0.018747
love        0.018561
delicious   0.016621
perfect     0.015956
would       0.015457
easy        0.011828
favorite    0.011240
throw       0.009744
good        0.009161
think       0.008896
highly      0.007962
wonderful   0.007707
money       0.007506
excellent   0.007076
waste       0.006958
terrible    0.006447
return      0.006421
maybe       0.006285
review      0.006256
```

In [ ]:

3. Average Word2Vec

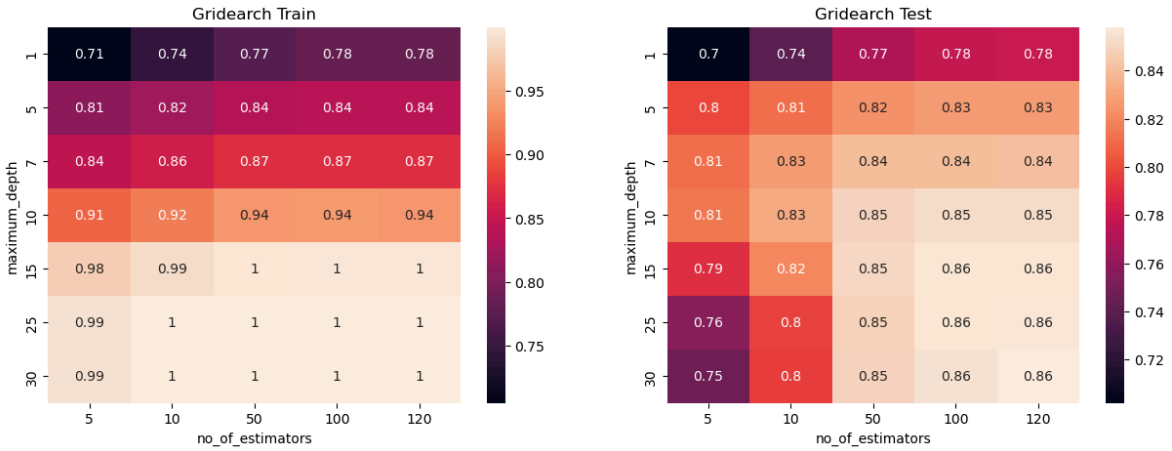
In [126…
```
X_train_avgw2v = pickle.load(open('w2v/avgw2v_train','rb'))
X_test_avgw2v = pickle.load(open('w2v/avgw2v_test','rb'))
w2v_words = pickle.load(open('w2v/w2v_words','rb'))
```

In [130…
```
rfr3 = RandomForestClassifier(n_jobs = -1, class_weight = 'balanced')
num_estimator = [5,10,50,100,120]
maximum_depth = [1, 5, 7, 10, 15, 25, 30]
parameter = {
    'n_estimators': num_estimator,
    'max_depth': maximum_depth
}
grid3 = GridSearchCV(estimator = rfr3, param_grid = parameter , cv =5, n_
grid3.fit(X_train_avgw2v,y_train)

print('Best Parameter:',grid.best_params_)
```

```
Best Parameter: {'max_depth': 25, 'n_estimators': 120}
```

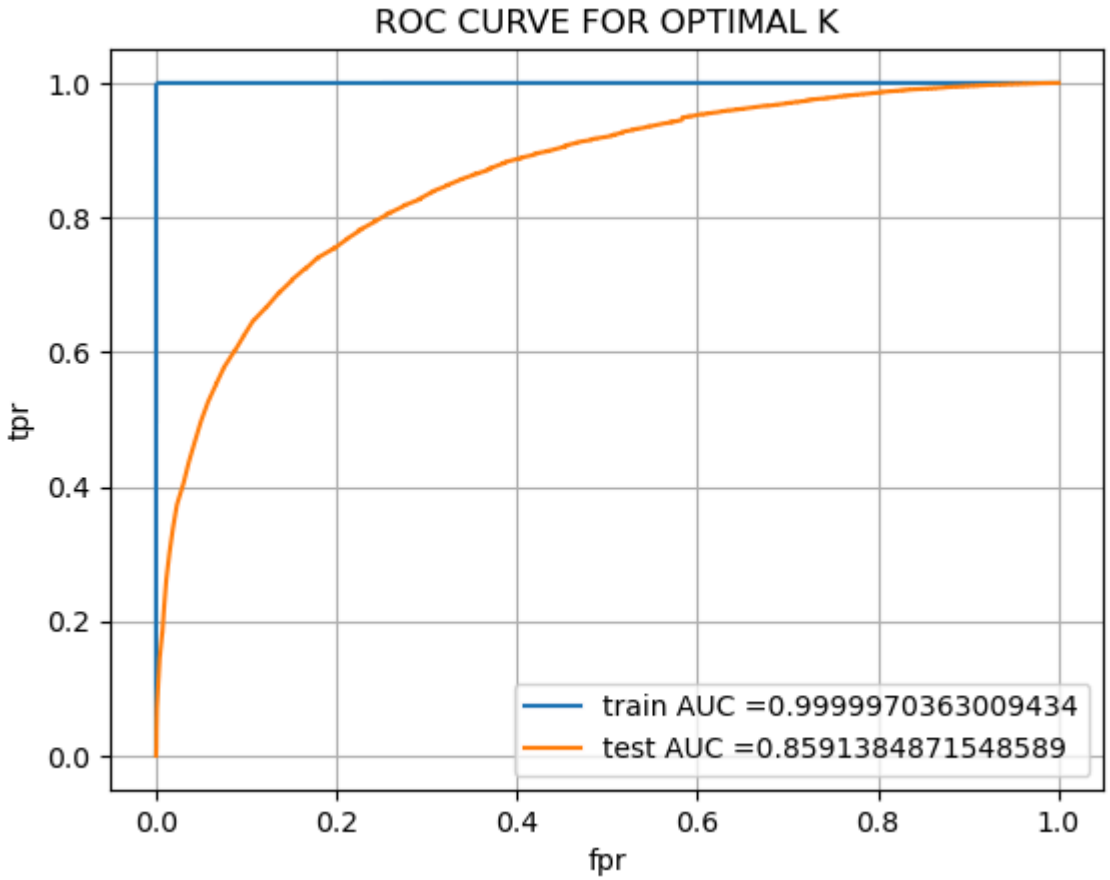
In [132…
```
train_auc = grid3.cv_results_['mean_train_score'].reshape(7,5)
test_auc = grid3.cv_results_['mean_test_score'].reshape(7,5)
```

In [134…
```
f, axes = plt.subplots(1, 2,figsize=(15,5))

for i in range(2):
    title = train_auc if i ==0 else test_auc
    sns.heatmap(title,xticklabels = num_estimator, yticklabels = maximum_
    axes[i].set_title(f"Gridearch {'Train' if i==0 else 'Test'}")
    axes[i].set_xlabel("no_of_estimators")
    axes[i].set_ylabel("maximum_depth")
```



In [136…

```
rfr_bp3 = RandomForestClassifier(**grid3.best_params_,class_weight = 'bal
rfr_bp3 = rfr_bp3.fit(X_train_avgw2v,y_train)
train_fpr, train_tpr, thresholds = roc_curve(y_train, rfr_bp3.predict_pro
test_fpr, test_tpr, thresholds = roc_curve(y_test, rfr_bp3.predict_proba(
plt.grid(True)
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, tra
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tp
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC CURVE FOR OPTIMAL K")
plt.show()

#Area under ROC curve
print('Area under train roc {}'.format(auc(train_fpr, train_tpr)))
print('Area under test roc {}'.format(auc(test_fpr, test_tpr)))
```




```
Area under train roc 0.9999970363009434
Area under test roc 0.8591384871548589
```

in avgW2v model is overfitted in Traning data , but performing good in test data

Observation and Calculation

In [142…
```
from prettytable import PrettyTable
```

In [144…
```
z = PrettyTable()

z.field_names = ["Vector","Algorithm","Hyperparam-min_sample_splits","Hyp
z.add_row(["bow","Random Forest",120,25, 0.95,0.88])
z.add_row(["tfidf","Random Forest",120,25,0.95, 0.88 ])
z.add_row(["avgw2v","Random Forest",120,25, 0.99,0.86])
print(z)
```

```
+--------+---------------+------------------------------+-----------------
-----+-----------+----------+
| Vector |   Algorithm   | Hyperparam-min_sample_splits | Hyperparam-max_D
epth | Train AUC | Test AUC |
+--------+---------------+------------------------------+-----------------
-----+-----------+----------+
|  bow   | Random Forest |             120              |          25
|    0.95   |   0.88   |
| tfidf  | Random Forest |             120              |          25
|    0.95   |   0.88   |
| avgw2v | Random Forest |             120              |          25
|    0.99   |   0.86   |
+--------+---------------+------------------------------+-----------------
-----+-----------+----------+
```

In [ ]:

XGBoost

In [2]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

import sklearn
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import roc_auc_score,roc_curve,auc
from sklearn.metrics import confusion_matrix

import pickle
```

1.BOW

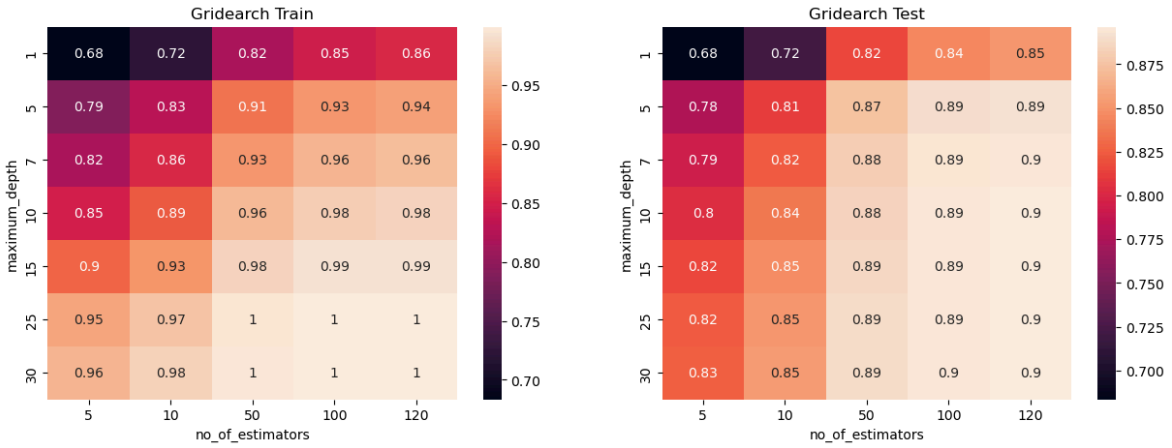
In [4]:
```
# loading BOW vectors

X_train_bow = pickle.load(open('bow_vectors/X_train_bow','rb'))
X_test_bow = pickle.load(open('bow_vectors/X_test_bow','rb'))
bow_features = pickle.load(open('bow_vectors/bow_features','rb'))

y_train = np.load('split dataset/y_train.npy',allow_pickle = True)
y_test = np.load('split dataset/y_test.npy',allow_pickle = True)
```

In [9]:
```
xgbc = XGBClassifier(n_jobs = -1, class_weight = 'balanced')
num_estimator = [5,10,50,100,120]
maximum_depth = [1, 5, 7, 10, 15, 25, 30]
parameter = {
    'n_estimators': num_estimator,
    'max_depth': maximum_depth
}
grid = GridSearchCV(estimator = xgbc , param_grid = parameter , cv =5, n_
grid.fit(X_train_bow,y_train)

print('Best Parameter:',grid.best_params_)
```

```
ng: [17:17:45] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:
740:
Parameters: { "class_weight" } are not used.

  warnings.warn(smsg, UserWarning)
/opt/anaconda3/lib/python3.11/site-packages/xgboost/core.py:158: UserWarni
ng: [17:17:45] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:
740:
Parameters: { "class_weight" } are not used.

  warnings.warn(smsg, UserWarning)
/opt/anaconda3/lib/python3.11/site-packages/xgboost/core.py:158: UserWarni
ng: [17:17:48] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:
740:
Parameters: { "class_weight" } are not used.

  warnings.warn(smsg, UserWarning)
/opt/anaconda3/lib/python3.11/site-packages/xgboost/core.py:158: UserWarni
ng: [17:17:51] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:
740:
Parameters: { "class_weight" } are not used.

  warnings.warn(smsg, UserWarning)
/opt/anaconda3/lib/python3.11/site-packages/xgboost/core.py:158: UserWarni
ng: [17:18:24] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:
740:
Parameters: { "class_weight" } are not used.

  warnings.warn(smsg, UserWarning)
/opt/anaconda3/lib/python3.11/site-packages/xgboost/core.py:158: UserWarni
ng: [17:18:25] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:
740:
Parameters: { "class_weight" } are not used.

  warnings.warn(smsg, UserWarning)
/opt/anaconda3/lib/python3.11/site-packages/xgboost/core.py:158: UserWarni
ng: [17:20:03] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:
740:
Parameters: { "class_weight" } are not used.

  warnings.warn(smsg, UserWarning)
Best Parameter: {'max_depth': 10, 'n_estimators': 120}
```

In [11]:
```
train_auc = grid.cv_results_['mean_train_score'].reshape(7,5)
test_auc = grid.cv_results_['mean_test_score'].reshape(7,5)
```

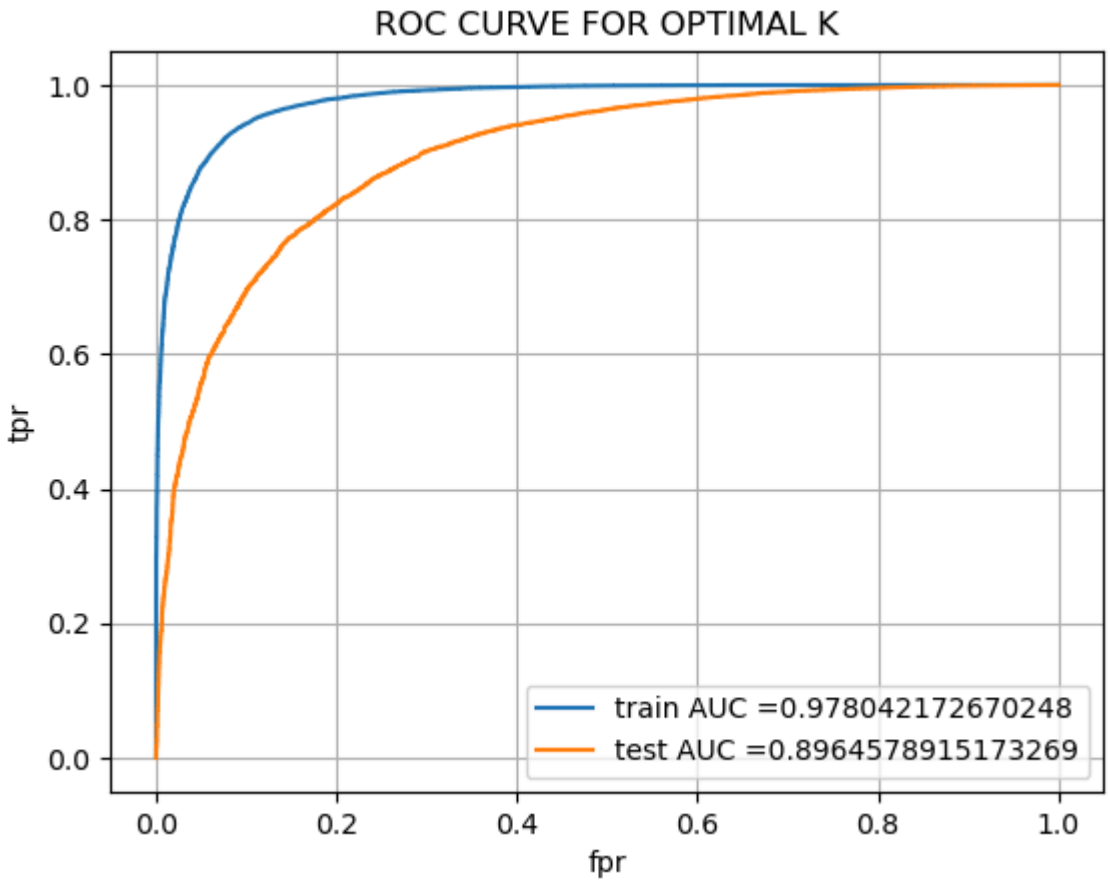
In [13]:
```
f, axes = plt.subplots(1, 2,figsize=(15,5))

for i in range(2):
    title = train_auc if i ==0 else test_auc
    sns.heatmap(title,xticklabels = num_estimator, yticklabels = maximum_
    axes[i].set_title(f"Gridearch {'Train' if i==0 else 'Test'}")
    axes[i].set_xlabel("no_of_estimators")
    axes[i].set_ylabel("maximum_depth")
```



here we can see that at max_dept = 10 and number of estimator =120 model have good score

In [24]:
```
xgbc_bp = XGBClassifier(**grid.best_params_,class_weight = 'balanced')
xgbc_bp = xgbc_bp.fit(X_train_bow,y_train)
train_fpr, train_tpr, thresholds = roc_curve(y_train, xgbc_bp.predict_pro
test_fpr, test_tpr, thresholds = roc_curve(y_test, xgbc_bp.predict_proba(
plt.grid(True)
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, tra
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tp
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC CURVE FOR OPTIMAL K")
plt.show()

#Area under ROC curve
print('Area under train roc {}'.format(auc(train_fpr, train_tpr)))
print('Area under test roc {}'.format(auc(test_fpr, test_tpr)))
```

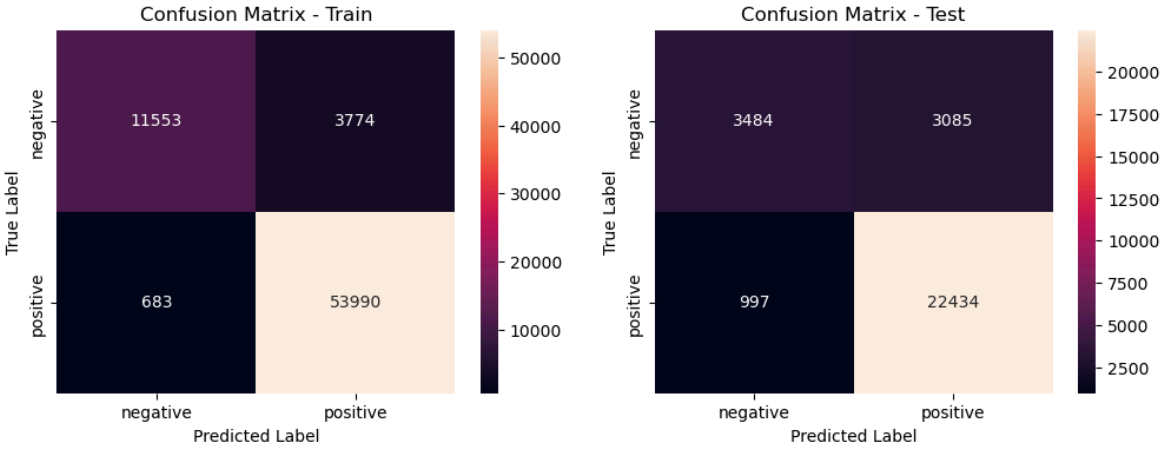
```
/opt/anaconda3/lib/python3.11/site-packages/xgboost/core.py:158: UserWarni
ng: [17:24:58] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:
740:
Parameters: { "class_weight" } are not used.

  warnings.warn(smsg, UserWarning)
```




```
Area under train roc 0.978042172670248
Area under test roc 0.8964578915173269
```

In [26]:
```
# helper function to plot confusion matrix
def plot_confusion_matrixes(model,x_train,y_train,x_test,y_test):
    cm_train = confusion_matrix(y_train,model.predict(x_train))
    cm_test =  confusion_matrix(y_test,model.predict(x_test))
    class_label = ["negative", "positive"]
    df_train = pd.DataFrame(cm_train, index = class_label, columns = clas
    df_test = pd.DataFrame(cm_test, index = class_label, columns = class_
    f, axes = plt.subplots(1, 2,figsize=(12,4))

    for i in range(2):
      df = df_train if i==0 else df_test
      sns.heatmap(df, annot = True, fmt = "d",ax=axes[i])
      axes[i].set_title(f"Confusion Matrix - {'Train' if i==0 else 'Test'
      axes[i].set_xlabel("Predicted Label")
      axes[i].set_ylabel("True Label")
    plt.show()
```

In [28]:
```
plot_confusion_matrixes(xgbc_bp,X_train_bow,y_train,X_test_bow,y_test)
```



In [30]:
```
from wordcloud import WordCloud
df = xgbc_bp.feature_importances_
features = bow_features
df = pd.DataFrame(df,columns=['coef'],index=features)
top = df.sort_values(by='coef',ascending=False).head(20)
print('Top 20 features are: \n {}'.format(top))
top['words'] = top.index
top.reset_index(drop=True)
sent = top.words.str.cat(sep=' ')
#word cloud representation
wordcloud = WordCloud(background_color='white').generate(sent)
plt.figure(figsize=(10,10))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```

```
Top 20 features are:
                     coef
return           0.011402
perfect          0.009996
delicious        0.007763
excellent        0.007539
highly           0.007023
throw            0.006674
waste            0.006631
disappointed     0.006295
awful            0.006190
horrible         0.005539
wonderful        0.005461
favorite         0.005461
description      0.005397
refund           0.005352
terrible         0.005324
bad              0.005134
stale            0.005030
great            0.005015
disappointment   0.004693
amazing          0.004535
```

2. TFIDF

In [10]:
```
# X_train_tfidf = pickle.load(open('tfidf_vectors/X_train_tfidf','rb'))
# X_test_tfidf = pickle.load(open('tfidf_vectors/X_test_tfidf','rb'))
# tfidf_features = pickle.load(open('tfidf_vectors/tfidf_features','rb'))
```

In [12]:
```
# xgbc2 = XGBClassifier()
# num_estimator = [5,10,50,100,120]
# maximum_depth = [1, 5, 7, 10, 15, 25, 30]
# parameter = {
#     'n_estimators': num_estimator,
#     'max_depth': maximum_depth
# }
# grid2 = GridSearchCV(estimator = xgbc2, param_grid = parameter , cv = 2
# grid2.fit(X_train_tfidf,y_train)

# print('Best Parameter:',grid2.best_params_)
```

Taking too much time in TIDF

In [3]:
```
from prettytable import PrettyTable
```

In [5]:
```
z = PrettyTable()

z.field_names = ["Vector","Algorithm","Hyperparam-min_sample_splits","Hyp
z.add_row(["bow","XGBoost",120,10, 0.97,0.89])
print(z)
```

```
+--------+-----------+------------------------------+--------------------
-+-----------+----------+
| Vector | Algorithm | Hyperparam-min_sample_splits | Hyperparam-max_Depth
| Train AUC | Test AUC |
+--------+-----------+------------------------------+--------------------
-+-----------+----------+
|  bow   |  XGBoost  |             120              |          10
|    0.97   |   0.89   |
+--------+-----------+------------------------------+--------------------
-+-----------+----------+
```

In Traning XGboostClassifier showing overfitting

In [ ]:

Insights into sentiment analysis on Amazon fine food reviews, covering aspects such as:

1. **Performance of different models like Naiv bayes, Logistic Regression, Decision Tree, and Random Forest Classifier.**
2. **Best parameters, area under ROC curve, and key words affecting sentiment analysis.**
3. **Target variable analysis and categorization of helpfulness percentage.**
4. **Data cleaning, exploratory data analysis, and preparation for sentiment analysis.**
5. **Techniques like stemming, lemmatization, and vectorization used in sentiment analysis.**
6. **Review sentiment categorization based on the Score feature.**
7. **Calculation of the percentage of helpfulness for reviews.**
8. **Dataset attributes and cleanliness issues.**
9. **Process steps for sentiment analysis, including loading vectors, model creation, and evaluation.**
10. **Visualization of categories and word clouds for sentiment analysis.**
11. **Handling of text data and rectifying cleanliness issues.**
12. **Comparison of models like BOW, TFIDF, and Average Word2Vec.**
13. **Exploration of Logistic Regression for sentiment analysis.**
14. **Analysis of reviews over time and helpfulness metrics.**
15. **Preparation for sentiment analysis by addressing cleanliness issues and text data cleaning.**
16. **XGBoost Classifier performing very well**

Challenges :-

1. **Cleaning the Data**
2. **Converting and choosing the right vectorization(BOW,TFIDF,AVG W2V)**
3. **Analyzing the model**