Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

Objective:

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

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

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

Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score id above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

import re
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

from sklearn.model_selection  import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection  import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import model_selection
from sklearn.metrics import roc_auc_score
```

[1]. Reading Data

In [2]:

```
# using the SQLite Table to read data.
con = sqlite3.connect('database.sqlite')
# filtering only positive and negative reviews

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 2

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative
def partition(x):
    if x < 3:
        return 0
```

```python
    return 1

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

```
Number of data points in our data (20000, 10)
```

Out[2]:

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

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B005ZBZLT4	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ESG	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B005ZBZLT4	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
3	#oc-R11O5J5ZVQE25C	B005HG9ESG	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBEV0	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [5]:

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

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B001ATMQK2	undertheshrine "undertheshrine"	1296691200	5	I bought this 6 pack because for the price tha...	5

In [6]:

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

Out[6]:

```
393063
```

Exploratory Data Analysis

[2] Data Cleaning: Deduplication

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

In [7]:

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

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomina
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomina
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	

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

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

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

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

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

In [8]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=F
```

In [9]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, kee
final.shape
```

Out[9]:

```
(19354, 10)
```

In [10]:

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

Out[10]:

```
96.77
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [11]:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomina
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	

In [12]:

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

In [13]:

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

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

```
(19354, 10)
```

Out[13]:

```
1    16339
0     3015
Name: Score, dtype: int64
```

[3]. Text Preprocessing.

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

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

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric

4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

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

In [14]:

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

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

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

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

```
We have used the Victor fly bait for 3 seasons.  Can't beat it.  Great product!
==================================================
I received this box with great anticipation since they don't sell these on the west
coast. I got the package, opened the box and was EXTREMELY disappointed. The cookies
looked like a gorilla shook the box to death and left most of the box filled with cr
umbs. AND THERE WAS A RODENT SIZED HOLE ON THE SIDE OF THE BOX!!!!!!!! So, needless
to say I will not NOT be reordering these again.
==================================================
I have two cats. My big boy has eaten these and never had a problem...as a matter of
fact he has never vomited or had a hair ball since I adopted him at 2 months. My gir
l cat throws up every time she eats this particular flavor. Since I treat them equal
ly these are no longer purchased. I hate to see my girl sick so I just recommend you
watch your cats after you give them these treats. If not a problem...carry on.
==================================================
I was always a fan of Dave's, so I bought this at a local store to try Blair's and
I'm glad I did.  The jalepeno sause is very mild (for me) but one of the most delici
ous condiments I've ever tasted.  The Afterdeath is a bit painful, but still very ta
sty on rice & beans, burritos, or any chicken dish I've tried it on.  The Sudden Dea
th kicked my ass when I underestimated it, but now a few drops in a dish or pot are
just right if I want heat without changing flavor much.
==================================================
```

In [15]:

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

print(sent_0)
```

```
We have used the Victor fly bait for 3 seasons.  Can't beat it.  Great product!
```

In [16]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-al
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
```

```
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
We have used the Victor fly bait for 3 seasons.  Can't beat it.  Great product!
==================================================
I received this box with great anticipation since they don't sell these on the west
coast. I got the package, opened the box and was EXTREMELY disappointed. The cookies
looked like a gorilla shook the box to death and left most of the box filled with cr
umbs. AND THERE WAS A RODENT SIZED HOLE ON THE SIDE OF THE BOX!!!!!!!! So, needless
to say I will not NOT be reordering these again.
==================================================
I have two cats. My big boy has eaten these and never had a problem...as a matter of
fact he has never vomited or had a hair ball since I adopted him at 2 months. My gir
l cat throws up every time she eats this particular flavor. Since I treat them equal
ly these are no longer purchased. I hate to see my girl sick so I just recommend you
watch your cats after you give them these treats. If not a problem...carry on.
==================================================
I was always a fan of Dave's, so I bought this at a local store to try Blair's and
I'm glad I did.  The jalepeno sause is very mild (for me) but one of the most delici
ous condiments I've ever tasted.  The Afterdeath is a bit painful, but still very ta
sty on rice & beans, burritos, or any chicken dish I've tried it on.  The Sudden Dea
th kicked my ass when I underestimated it, but now a few drops in a dish or pot are
just right if I want heat without changing flavor much.
```

In [17]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [18]:

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

```
I have two cats. My big boy has eaten these and never had a problem...as a matter of
fact he has never vomited or had a hair ball since I adopted him at 2 months. My gir
l cat throws up every time she eats this particular flavor. Since I treat them equal
```

```
ly these are no longer purchased. I hate to see my girl sick so I just recommend you
watch your cats after you give them these treats. If not a problem...carry on.
==================================================
```

In [19]:

```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
We have used the Victor fly bait for  seasons.  Can't beat it.  Great product!
```

In [20]:

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

```
I have two cats My big boy has eaten these and never had a problem as a matter of fa
ct he has never vomited or had a hair ball since I adopted him at 2 months My girl c
at throws up every time she eats this particular flavor Since I treat them equally t
hese are no longer purchased I hate to see my girl sick so I just recommend you watc
h your cats after you give them these treats If not a problem carry on
```

In [21]:

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

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'oursel
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that',
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', '
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throu
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', '
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 't
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've",
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn'
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'm
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
            'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:

```
#using snowball stemmer for stemming
snowball_stemmer = SnowballStemmer(language='english')

def performStemming(text):
    for word in text.split():
        text = text.replace(word, snowball_stemmer.stem(word))
    return text
```

[3.1] Preprocess Reviews

In [23]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
```

```
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stop
    sentance = performStemming(sentance)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|████████████████████████████████████████████████████| 193
54/19354 [00:16<00:00, 1157.36it/s]
```

In [24]:

```
preprocessed_reviews[1500]
```

Out[24]:

```
'two cat big boy eaten never problem matter fact never vomit hair ball sinc adopt mo
nth girl cat throw everi time eat particular flavor sinc treat equal no longer purch
as hate see girl sick recommend watch cat give treat not problem carri'
```

[3.2] Preprocess Summary

In [25]:

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

```
100%|████████████████████████████████████████████████████| 193
54/19354 [00:05<00:00, 3717.66it/s]
```

In [26]:

```
preprocessed_summaries[1500]
```

Out[26]:

```
'temptat vomit'
```

[4] Featurization

[4.1] BAG OF WORDS

In [27]:

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

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

```
some feature names  ['aa', 'aaaa', 'aaaaa', 'aaaand', 'aafco', 'aah', 'aahhh', 'ab',
'aback', 'abandon']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
```

```
the shape of out text BOW vectorizer  (19354, 18259)
the number of unique words  18259
```

[4.2] Bi-Grams and n-Grams.

In [28]:

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

# removing stop words like "not" should be avoided before building n-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.org/stable/mo
# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigra
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (19354, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [29]:

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

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

```
some sample features(unique words in the corpus) ['abil', 'abl', 'abl buy', 'abl ea
t', 'abl find', 'abl get', 'abl give', 'abl make', 'abl order', 'abl purchas']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (19354, 12044)
the number of unique words including both unigrams and bigrams  12044
```

[4.4] Word2Vec

In [30]:

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

In [31]:

```
w2v_model=Word2Vec(list_of_sentance,min_count=5, vector_size=50, workers=4)
print(w2v_model.wv.most_similar('great'))
print('='*50)
print(w2v_model.wv.most_similar('worst'))
```

```
[('fantast', 0.7926201820373535), ('awesom', 0.7872790098190308), ('wonder', 0.76571
72083854675), ('excel', 0.759020268917083 7), ('amaz', 0.7559347748756409), ('good',
0.7552913427352905), ('perfect', 0.7181152701377869), ('especi', 0.699252903461456
3), ('nice', 0.6533973217010498), ('decent', 0.6212543845176697)]
==================================================
[('hate', 0.7069545388221741), ('heard', 0.7053143382072449), ('greatest', 0.7040787
935256958), ('aw', 0.6983534693717957), ('disgust', 0.6910261511802673), ('complic',
```

```
0.6813580393791199), ('closest', 0.6794140338897705), ('experienc', 0.67745351791381
84), ('best', 0.6735631227493286), ('sad', 0.670515775680542)]
```

In [32]:

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

```
number of words that occured minimum 5 times  6054
sample words  ['not', 'like', 'tast', 'good', 'flavor', 'love', 'one', 'product', 'g
reat', 'use', 'coffe', 'tri', 'would', 'food', 'dog', 'get', 'make', 'tea', 'no', 'b
uy', 'time', 'eat', 'realli', 'cup', 'price', 'order', 'much', 'treat', 'amazon', 'a
lso', 'best', 'littl', 'bag', 'find', 'drink', 'even', 'well', 'store', 'mix', 'bett
er', 'go', 'recommend', 'look', 'chocol', 'year', 'sugar', 'give', 'first', 'day',
'want']
```

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

[4.4.1.1] Avg W2v

In [33]:

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

```
100%|██████████████████████████████████████████████████| 193
54/19354 [00:11<00:00, 1659.87it/s]
19354
50
```

[4.4.1.2] TFIDF weighted W2v

In [34]:

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

In [35]:

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

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this
row=0;
for sent in tqdm(list_of_sentance[:10000]): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
```

```
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████████████████| 10
000/10000 [01:05<00:00, 152.33it/s]
```

[5] KNN Model with reviews data

[5.1] Dividing data into train, test & CV

In [36]:

```
X = preprocessed_reviews
Y = np.array(final['Score'])

X_1, X_test, y_1, y_test = train_test_split(X, Y, test_size=0.3, random_state=0)
X_tr, X_cv, y_tr, y_cv = train_test_split(X_1, y_1, test_size=0.3)
```

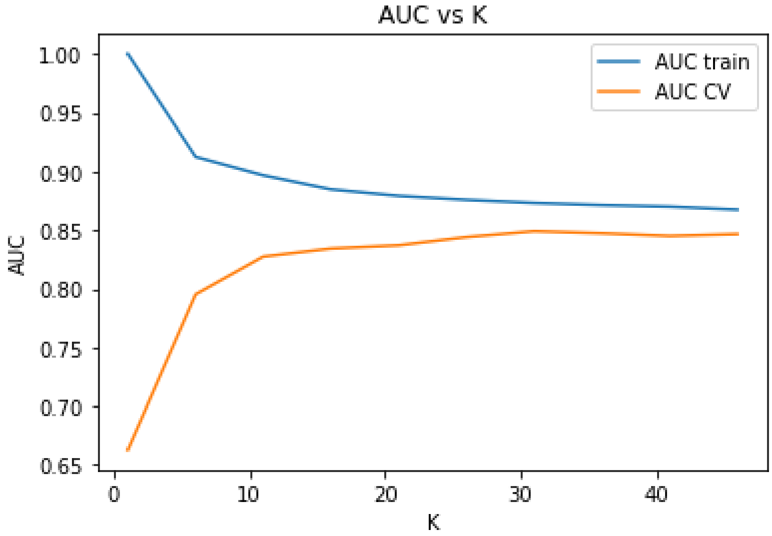
[5.2] KNN Model with BOW

In [37]:

```
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=8000)
final_bigram_counts = count_vect.fit(X_tr)

x_train = count_vect.transform(X_tr)
x_test = count_vect.transform(X_test)
x_cv = count_vect.transform(X_cv)

print("number of points in train data ", x_train.get_shape())
print("number of points in test data ", x_test.get_shape())
print("number of points in CV data ", x_cv.get_shape())

auc_cv = []
auc_train = []
K = []

for i in tqdm(range(1,50,5)):
    knn = KNeighborsClassifier(n_neighbors=i,weights='uniform',algorithm='brute',lea
    knn.fit(x_train, y_tr)
    pred = knn.predict_proba(x_cv)[:,1]
    pred1 = knn.predict_proba(x_train)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred))
    auc_train.append(roc_auc_score(y_tr,pred1))
    K.append(i)

fig = plt.figure()
ax = plt.subplot(111)
ax.plot(K, auc_train, label='AUC train')
ax.plot(K, auc_cv, label='AUC CV')
plt.title('AUC vs K')
plt.xlabel('K')
plt.ylabel('AUC')
ax.legend()
plt.show()
```

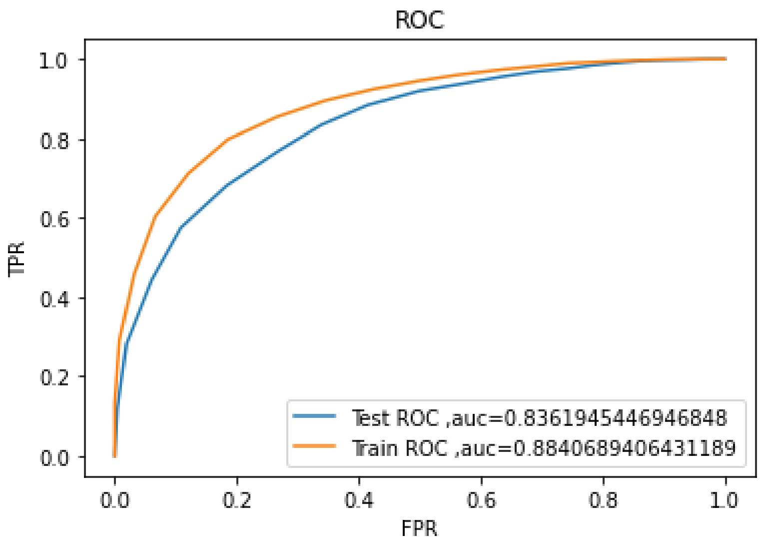
```
  0%|
```

```
| 0/10 [00:00<?, ?it/s]
number of points in train data  (9482, 5862)
number of points in test data  (5807, 5862)
number of points in CV data  (4065, 5862)
100%|████████████████████████████████████████
███| 10/10 [00:55<00:00,  5.52s/it]
```



In [38]:

```python
# KNN with optimal parameters
from sklearn.metrics import confusion_matrix

final_counts = count_vect.fit(X_1)

x_train = count_vect.transform(X_1)
x_test = count_vect.transform(X_test)

knn = KNeighborsClassifier(n_neighbors=26,weights='uniform',algorithm='brute',leaf_s
knn.fit(x_train, y_1)
predi = knn.predict_proba(x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, predi)
pred = knn.predict_proba(x_train)[:,1]
fpr2,tpr2,thresholds2 = metrics.roc_curve(y_1,pred)
fig = plt.figure()
ax = plt.subplot(111)
ax.plot(fpr1, tpr1, label='Test ROC ,auc='+str(roc_auc_score(y_test,predi)))
ax.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_1,pred)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
ax.legend()
plt.show()
```

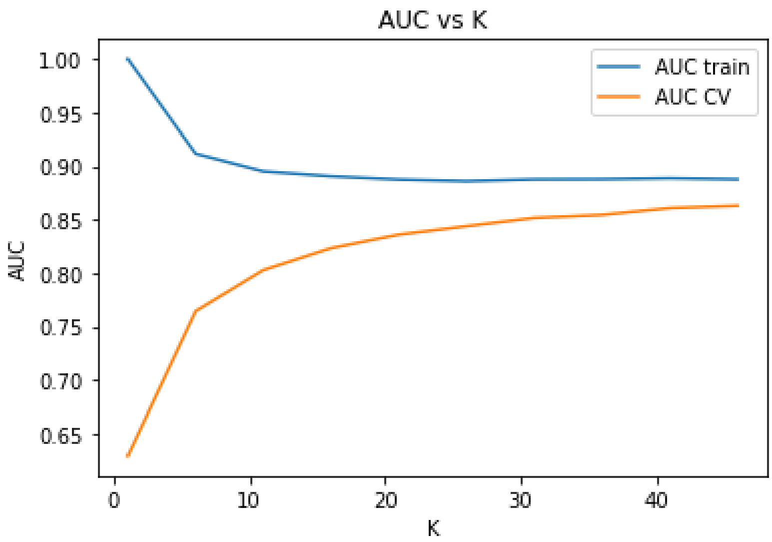


[5.3] KNN Model with TFIDF

In [39]:

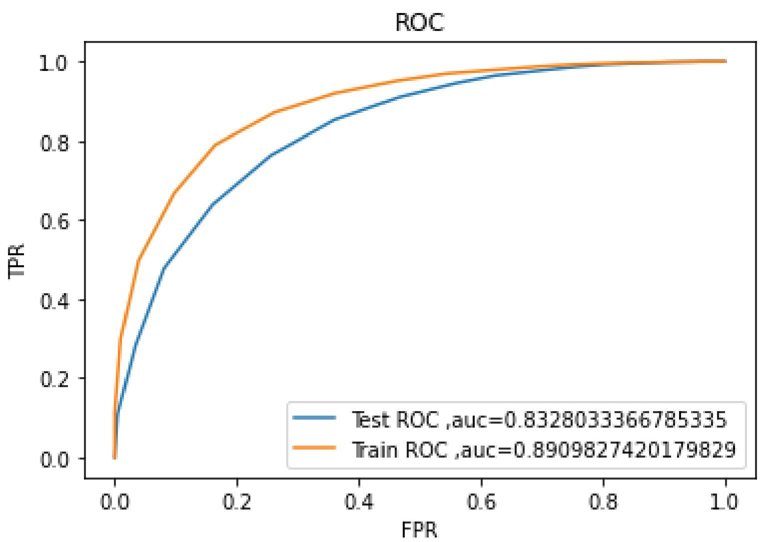
```
tfidf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features=8000)
final_bigram_counts = tfidf_vect.fit(X_tr)

x_train = tfidf_vect.transform(X_tr)
x_test = tfidf_vect.transform(X_test)
x_cv = tfidf_vect.transform(X_cv)

print("number of points in train data ", x_train.get_shape())
print("number of points in test data ", x_test.get_shape())
print("number of points in CV data ", x_cv.get_shape())

auc_cv = []
auc_train = []
K = []

for i in tqdm(range(1,50,5)):
    knn = KNeighborsClassifier(n_neighbors=i,weights='uniform',algorithm='brute',lea
    knn.fit(x_train, y_tr)
    pred = knn.predict_proba(x_cv)[:,1]
    pred1 = knn.predict_proba(x_train)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred))
    auc_train.append(roc_auc_score(y_tr,pred1))
    K.append(i)

fig = plt.figure()
ax = plt.subplot(111)
ax.plot(K, auc_train, label='AUC train')
ax.plot(K, auc_cv, label='AUC CV')
plt.title('AUC vs K')
plt.xlabel('K')
plt.ylabel('AUC')
ax.legend()
plt.show()
```

```
  0%|
| 0/10 [00:00<?, ?it/s]
number of points in train data  (9482, 5862)
number of points in test data  (5807, 5862)
number of points in CV data  (4065, 5862)
100%|████████████████████████████████████████
███| 10/10 [00:52<00:00,  5.22s/it]
```



In [40]:

```
# KNN with optimal parameters
from sklearn.metrics import confusion_matrix

final_vect = tfidf_vect.fit(X_1)

x_train = tfidf_vect.transform(X_1)
x_test = tfidf_vect.transform(X_test)

knn = KNeighborsClassifier(n_neighbors=26,weights='uniform',algorithm='brute',leaf_s
knn.fit(x_train, y_1)
predi = knn.predict_proba(x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, predi)
pred = knn.predict_proba(x_train)[:,1]
fpr2,tpr2,thresholds2 = metrics.roc_curve(y_1,pred)
fig = plt.figure()
ax = plt.subplot(111)
ax.plot(fpr1, tpr1, label='Test ROC ,auc='+str(roc_auc_score(y_test,predi)))
ax.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_1,pred)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
ax.legend()
plt.show()
```



[5.4] KNN Model with TFIDF weighted W2V

In [41]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
w2v_model=Word2Vec(list_of_sentance, min_count=5, vector_size=60, workers=4)
model = TfidfVectorizer()
model.fit(X_tr)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))

def getTfidfWeightedVector(reviews):
    i=0
    list_of_sentance=[]
    for sentance in reviews:
        list_of_sentance.append(sentance.split())
    tfidf_feat = model.get_feature_names()
    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in t
    row=0;
    for sent in tqdm(list_of_sentance): # for each review/sentence
        sent_vec = np.zeros(60) # as word vectors are of zero length
        weight_sum =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_words and word in tfidf_feat:
                vec = w2v_model.wv[word]
                tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                sent_vec += (vec * tf_idf)
                weight_sum += tf_idf
        if weight_sum != 0:
            sent_vec /= weight_sum
        tfidf_sent_vectors.append(sent_vec)
        row += 1
    return tfidf_sent_vectors
```

In [42]:

```
print("Vectorizing train data")
x_train = getTfidfWeightedVector(X_tr)
print("Vectorizing test data")
x_test = getTfidfWeightedVector(X_test)
print("Vectorizing CV data")
x_cv = getTfidfWeightedVector(X_cv)
```

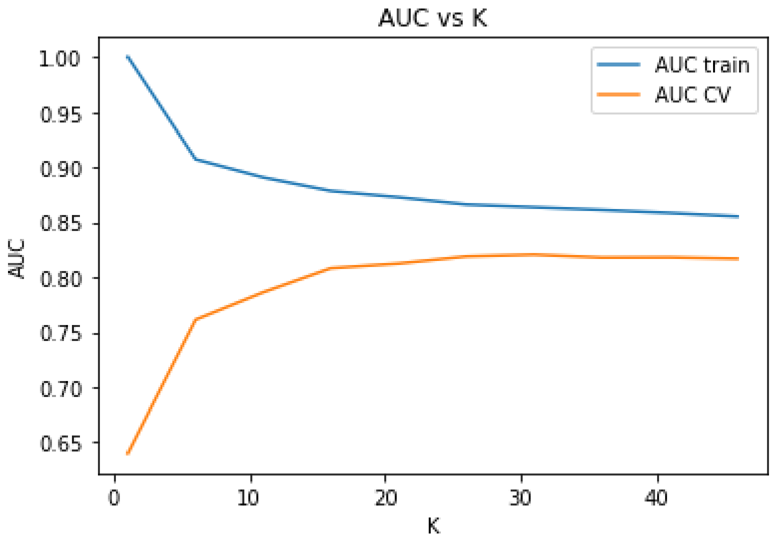
```
  0%||
| 21/9482 [00:00<00:45, 208.48it/s]
Vectorizing train data
100%|██████████████████████████████████████████████████|
9482/9482 [00:45<00:00, 208.07it/s]
  0%|▌
| 29/5807 [00:00<00:21, 271.83it/s]
Vectorizing test data
100%|██████████████████████████████████████████████████|
5807/5807 [00:28<00:00, 205.29it/s]
  1%|█
| 40/4065 [00:00<00:21, 189.28it/s]
Vectorizing CV data
100%|██████████████████████████████████████████████████|
4065/4065 [00:19<00:00, 208.56it/s]
```

In [43]:

```
print("number of points in train data ", len(x_train))
print("number of points in test data ", len(x_test))
print("number of points in CV data ", len(x_cv))

auc_cv = []
auc_train = []
K = []

for i in tqdm(range(1,50,5)):
```

```
    knn = KNeighborsClassifier(n_neighbors=i,weights='uniform',algorithm='brute',lea
    knn.fit(x_train, y_tr)
    pred = knn.predict_proba(x_cv)[:,1]
    pred1 = knn.predict_proba(x_train)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred))
    auc_train.append(roc_auc_score(y_tr,pred1))
    K.append(i)

fig = plt.figure()
ax = plt.subplot(111)
ax.plot(K, auc_train, label='AUC train')
ax.plot(K, auc_cv, label='AUC CV')
plt.title('AUC vs K')
plt.xlabel('K')
plt.ylabel('AUC')
ax.legend()
plt.show()
```

```
  0%|
| 0/10 [00:00<?, ?it/s]
number of points in train data  9482
number of points in test data  5807
number of points in CV data  4065
100%|██████████████████████████████████████████████████
███| 10/10 [00:31<00:00,  3.11s/it]
```

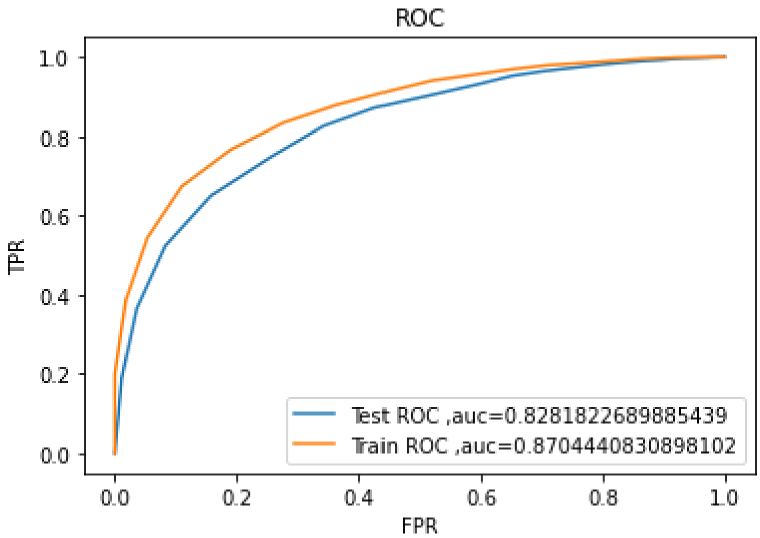


In [44]:

```
#KNN with optimal parameters
w2v_model = Word2Vec(list_of_sentance, min_count=5, vector_size=60, workers=4)
model = TfidfVectorizer()
model.fit(X_1)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))

print("Vectorizing train data")
x_train = getTfidfWeightedVector(X_1)
print("Vectorizing test data")
x_test = getTfidfWeightedVector(X_test)

knn = KNeighborsClassifier(n_neighbors=26,weights='uniform',algorithm='brute',leaf_s
knn.fit(x_train, y_1)
predi = knn.predict_proba(x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, predi)
pred = knn.predict_proba(x_train)[:,1]
fpr2,tpr2,thresholds2 = metrics.roc_curve(y_1,pred)
fig = plt.figure()
ax = plt.subplot(111)
```

```
ax.plot(fpr1, tpr1, label='Test ROC ,auc='+str(roc_auc_score(y_test,predi)))
ax.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_1,pred)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
ax.legend()
plt.show()
```

```
  0%|                                                                       |
21/13547 [00:00<01:06, 202.40it/s]
Vectorizing train data
100%|██████████████████████████████████████████████████████████████████████| 13
547/13547 [01:14<00:00, 181.66it/s]
  1%|█
| 50/5807 [00:00<00:29, 196.92it/s]
Vectorizing test data
100%|████████████████████████████████████████████████████████████████████████|
5807/5807 [00:32<00:00, 180.97it/s]
```



[6] KNN Model with summary data

[6.1] Dividing data into train, test & CV

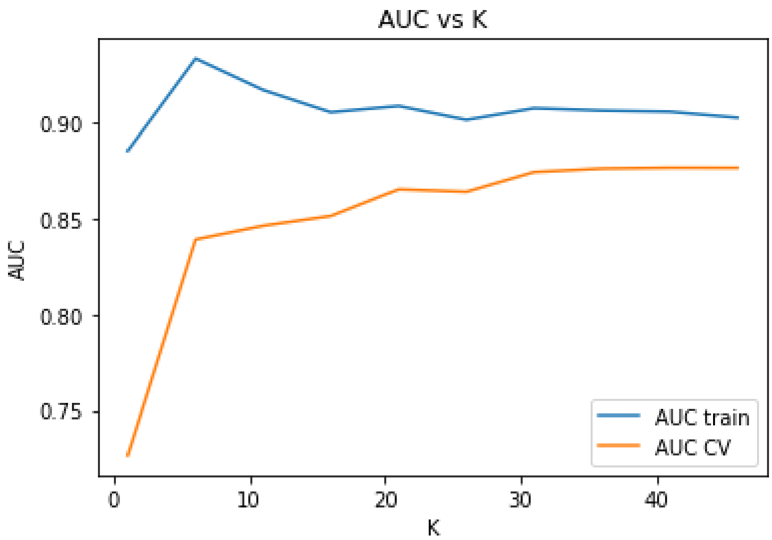
In [46]:
```
X = preprocessed_summaries
Y = np.array(final['Score'])

X_1, X_test, y_1, y_test = train_test_split(X, Y, test_size=0.3, random_state=0)
X_tr, X_cv, y_tr, y_cv = train_test_split(X_1, y_1, test_size=0.3)
```

[6.2] KNN Model with BOW

In [47]:
```
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=8000)
final_bigram_counts = count_vect.fit(X_tr)

x_train = count_vect.transform(X_tr)
x_test = count_vect.transform(X_test)
x_cv = count_vect.transform(X_cv)

print("number of points in train data ", x_train.get_shape())
print("number of points in test data ", x_test.get_shape())
print("number of points in CV data ", x_cv.get_shape())

auc_cv = []
```

```python
auc_train = []
K = []

for i in tqdm(range(1,50,5)):
    knn = KNeighborsClassifier(n_neighbors=i,weights='uniform',algorithm='brute',lea
    knn.fit(x_train, y_tr)
    pred = knn.predict_proba(x_cv)[:,1]
    pred1 = knn.predict_proba(x_train)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred))
    auc_train.append(roc_auc_score(y_tr,pred1))
    K.append(i)

fig = plt.figure()
ax = plt.subplot(111)
ax.plot(K, auc_train, label='AUC train')
ax.plot(K, auc_cv, label='AUC CV')
plt.title('AUC vs K')
plt.xlabel('K')
plt.ylabel('AUC')
ax.legend()
plt.show()
```

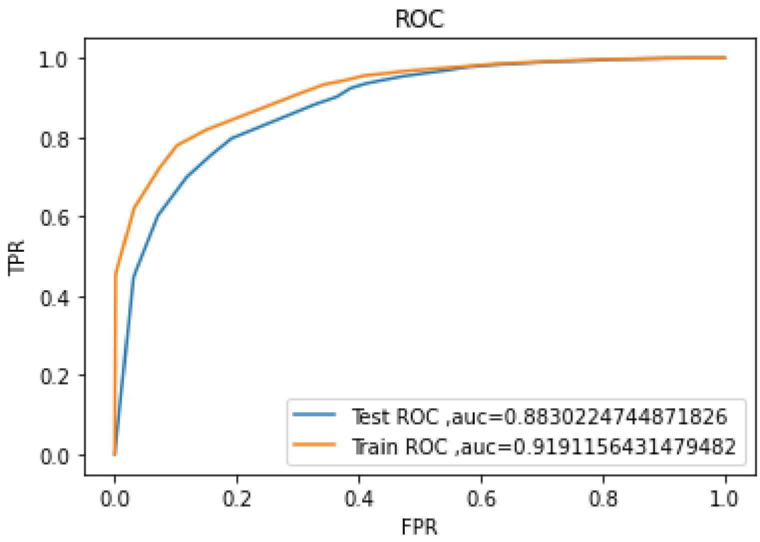
```
  0%|
| 0/10 [00:00<?, ?it/s]
number of points in train data  (9482, 602)
number of points in test data  (5807, 602)
number of points in CV data  (4065, 602)
100%|████████████████████████████████████████
███| 10/10 [00:35<00:00,  3.59s/it]
```



In [48]:

```python
# KNN with optimal parameters
from sklearn.metrics import confusion_matrix

final_counts = count_vect.fit(X_1)

x_train = count_vect.transform(X_1)
x_test = count_vect.transform(X_test)

knn = KNeighborsClassifier(n_neighbors=26,weights='uniform',algorithm='brute',leaf_s
knn.fit(x_train, y_1)
predi = knn.predict_proba(x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, predi)
pred = knn.predict_proba(x_train)[:,1]
fpr2,tpr2,thresholds2 = metrics.roc_curve(y_1,pred)
fig = plt.figure()
ax = plt.subplot(111)
```

```
ax.plot(fpr1, tpr1, label='Test ROC ,auc='+str(roc_auc_score(y_test,predi)))
ax.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_1,pred)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
ax.legend()
plt.show()
```



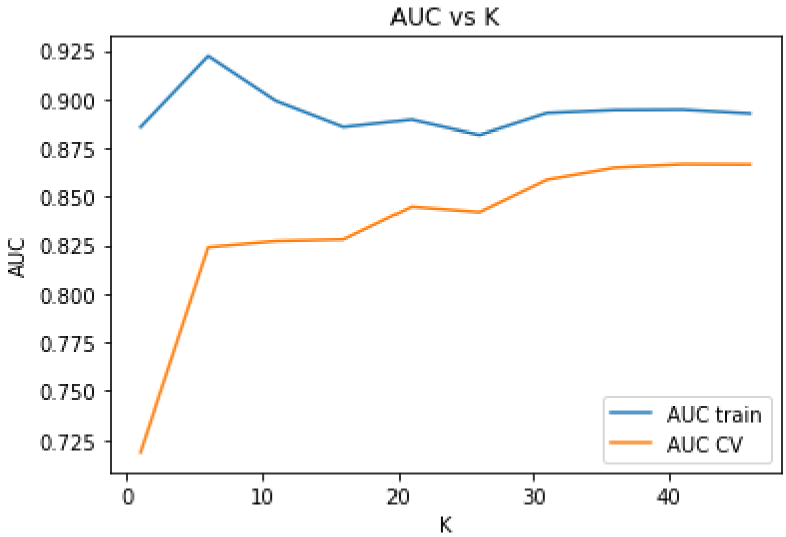
[6.3] KNN Model with TFIDF

In [49]:

```
tfidf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features=8000)
final_bigram_counts = tfidf_vect.fit(X_tr)

x_train = tfidf_vect.transform(X_tr)
x_test = tfidf_vect.transform(X_test)
x_cv = tfidf_vect.transform(X_cv)

print("number of points in train data ", x_train.get_shape())
print("number of points in test data ", x_test.get_shape())
print("number of points in CV data ", x_cv.get_shape())

auc_cv = []
auc_train = []
K = []

for i in tqdm(range(1,50,5)):
    knn = KNeighborsClassifier(n_neighbors=i,weights='uniform',algorithm='brute',lea
    knn.fit(x_train, y_tr)
    pred = knn.predict_proba(x_cv)[:,1]
    pred1 = knn.predict_proba(x_train)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred))
    auc_train.append(roc_auc_score(y_tr,pred1))
    K.append(i)

fig = plt.figure()
ax = plt.subplot(111)
ax.plot(K, auc_train, label='AUC train')
ax.plot(K, auc_cv, label='AUC CV')
plt.title('AUC vs K')
plt.xlabel('K')
plt.ylabel('AUC')
ax.legend()
plt.show()
```

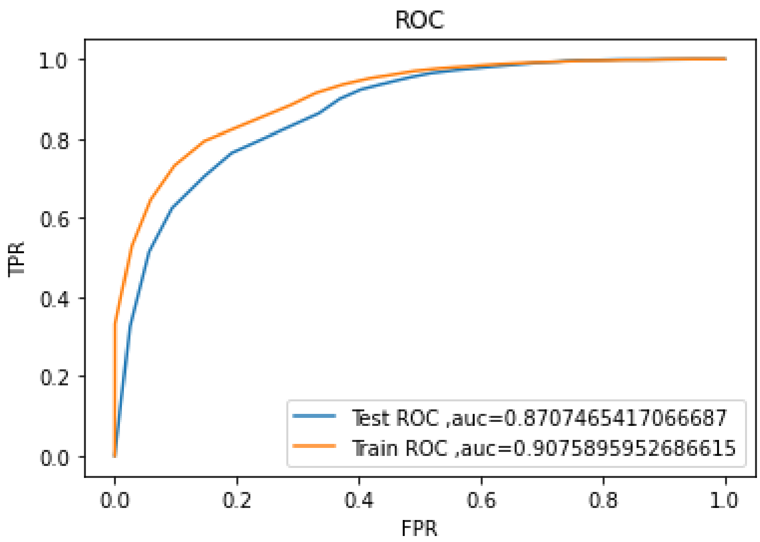
```
0%|
```

```
| 0/10 [00:00<?, ?it/s]
number of points in train data  (9482, 602)
number of points in test data  (5807, 602)
number of points in CV data  (4065, 602)
100%|██████████████████████████████████████████████████| 10/10 [00:36<00:00,  3.65s/it]
```



In [50]:

```python
# KNN with optimal parameters
from sklearn.metrics import confusion_matrix

final_vect = tfidf_vect.fit(X_1)

x_train = tfidf_vect.transform(X_1)
x_test = tfidf_vect.transform(X_test)

knn = KNeighborsClassifier(n_neighbors=26,weights='uniform',algorithm='brute',leaf_s
knn.fit(x_train, y_1)
predi = knn.predict_proba(x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, predi)
pred = knn.predict_proba(x_train)[:,1]
fpr2,tpr2,thresholds2 = metrics.roc_curve(y_1,pred)
fig = plt.figure()
ax = plt.subplot(111)
ax.plot(fpr1, tpr1, label='Test ROC ,auc='+str(roc_auc_score(y_test,predi)))
ax.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_1,pred)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
ax.legend()
plt.show()
```



[6.4] KNN Model with TFIDF weighted W2V

In [51]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
w2v_model=Word2Vec(list_of_sentance, min_count=5, vector_size=60, workers=4)
model = TfidfVectorizer()
model.fit(X_tr)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))

def getTfidfWeightedVector(reviews):
    i=0
    list_of_sentance=[]
    for sentance in reviews:
        list_of_sentance.append(sentance.split())
    tfidf_feat = model.get_feature_names()
    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in t
    row=0;
    for sent in tqdm(list_of_sentance): # for each review/sentence
        sent_vec = np.zeros(60) # as word vectors are of zero length
        weight_sum =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_words and word in tfidf_feat:
                vec = w2v_model.wv[word]
                tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                sent_vec += (vec * tf_idf)
                weight_sum += tf_idf
        if weight_sum != 0:
            sent_vec /= weight_sum
        tfidf_sent_vectors.append(sent_vec)
        row += 1
    return tfidf_sent_vectors
```

In [52]:

```
print("Vectorizing train data")
x_train = getTfidfWeightedVector(X_tr)
print("Vectorizing test data")
x_test = getTfidfWeightedVector(X_test)
print("Vectorizing CV data")
x_cv = getTfidfWeightedVector(X_cv)
```

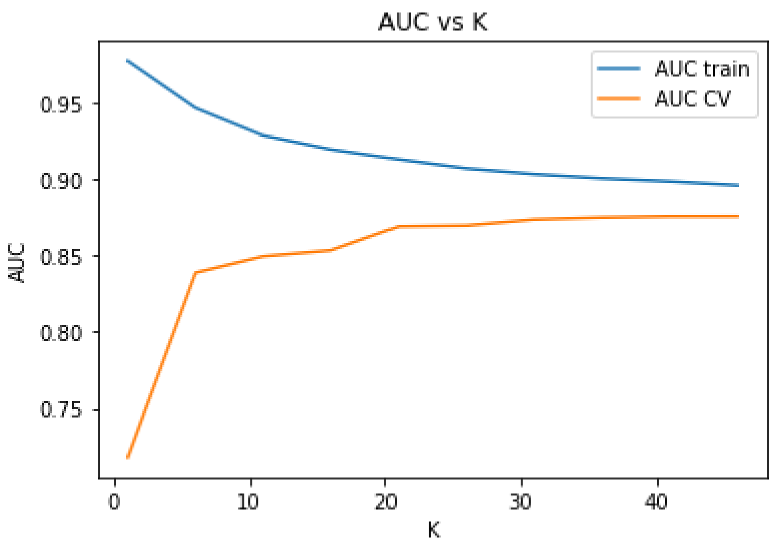
```
 11%|███████                                                                  |
997/9482 [00:00<00:01, 4964.19it/s]
Vectorizing train data
100%|█████████████████████████████████████████████████████████████████████| 9
```

```
482/9482 [00:01<00:00, 5212.95it/s]
 17%|█████████                                                             | 1
000/5807 [00:00<00:00, 5001.24it/s]
Vectorizing test data
100%|█████████████████████████████████████████████████████████████████████| 5
807/5807 [00:01<00:00, 5347.34it/s]
 28%|███████████████                                                       | 1
151/4065 [00:00<00:00, 5307.40it/s]
Vectorizing CV data
100%|█████████████████████████████████████████████████████████████████████| 4
065/4065 [00:00<00:00, 5131.27it/s]
```

In [53]:

```python
print("number of points in train data ", len(x_train))
print("number of points in test data ", len(x_test))
print("number of points in CV data ", len(x_cv))

auc_cv = []
auc_train = []
K = []

for i in tqdm(range(1,50,5)):
    knn = KNeighborsClassifier(n_neighbors=i,weights='uniform',algorithm='brute',lea
    knn.fit(x_train, y_tr)
    pred = knn.predict_proba(x_cv)[:,1]
    pred1 = knn.predict_proba(x_train)[:,1]
    auc_cv.append(roc_auc_score(y_cv,pred))
    auc_train.append(roc_auc_score(y_tr,pred1))
    K.append(i)

fig = plt.figure()
ax = plt.subplot(111)
ax.plot(K, auc_train, label='AUC train')
ax.plot(K, auc_cv, label='AUC CV')
plt.title('AUC vs K')
plt.xlabel('K')
plt.ylabel('AUC')
ax.legend()
plt.show()
```

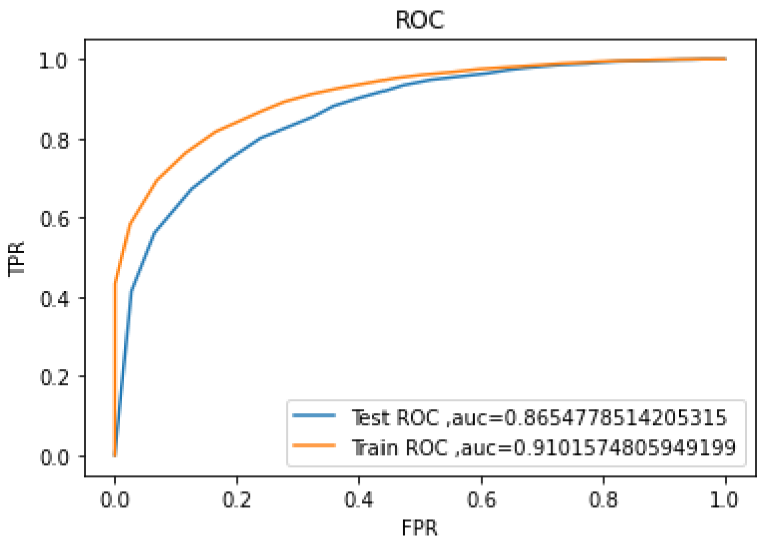
```
  0%|
| 0/10 [00:00<?, ?it/s]
number of points in train data  9482
number of points in test data  5807
number of points in CV data  4065
100%|████████████████████████████████████████████████████████████████████████
███| 10/10 [00:38<00:00,  3.84s/it]
```



In [54]:

```
#KNN with optimal parameters
w2v_model = Word2Vec(list_of_sentance, min_count=5, vector_size=60, workers=4)
model = TfidfVectorizer()
model.fit(X_1)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))

print("Vectorizing train data")
x_train = getTfidfWeightedVector(X_1)
print("Vectorizing test data")
x_test = getTfidfWeightedVector(X_test)

knn = KNeighborsClassifier(n_neighbors=26,weights='uniform',algorithm='brute',leaf_s
knn.fit(x_train, y_1)
predi = knn.predict_proba(x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, predi)
pred = knn.predict_proba(x_train)[:,1]
fpr2,tpr2,thresholds2 = metrics.roc_curve(y_1,pred)
fig = plt.figure()
ax = plt.subplot(111)
ax.plot(fpr1, tpr1, label='Test ROC ,auc='+str(roc_auc_score(y_test,predi)))
ax.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_1,pred)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
ax.legend()
plt.show()
```

```
  8%|███                                                                        | 1
020/13547 [00:00<00:02, 4473.56it/s]
Vectorizing train data
100%|█████████████████████████████████████████████████████████████████████| 135
47/13547 [00:02<00:00, 4674.06it/s]
  9%|███                                                                          |
512/5807 [00:00<00:01, 5075.04it/s]
Vectorizing test data
100%|██████████████████████████████████████████████████████████████████████| 5
807/5807 [00:01<00:00, 3936.21it/s]
```



[7.0] Overall Results (AUC Score)

KNN on Reviews

Vectorizer	Train	Test

BOW	0.88	0.83
TFIDF	0.89	0.83
TFIDF weighted W2V	0.87	0.82

KNN on Summary

Vectorizer	Train	Test
BOW	0.91	0.88
TFIDF	0.90	0.87
TFIDF weighted W2V	0.91	0.86