Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

Objective:

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

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

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

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


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

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

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

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

from tqdm import tqdm
```

In [2]:
```
# using SQLite Table to read data.
con = sqlite3.connect("amazon-fine-food-reviews\database.sqlite")

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

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

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

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rating(0).
def partition(x):
    if x < 3:
        return 0
    return 1

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

```
Number of data points in our data (525814, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all	This is a confection that has been around a fe...

In [3]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
```

In [4]:
```
print(display.shape)
```

```
(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
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]:

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]:

Out[6]: 393063

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

In [7]:

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

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...

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

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

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

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

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

In [8]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='l
```

In [9]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
```

Out[9]: (364173, 10)

In [10]:

```
#Checking to see how much % of data still remains
```

Out[10]: 69.25890143662969

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)
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son at College	My son loves spaghetti so I didn't hesitate or...
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside	It was almost a 'love at first bite' - the per...

In [12]:

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?
```

```
(364171, 10)
```

Out[13]:
```
1    307061
0     57110
Name: Score, dtype: int64
```

From the above result, we can see that data is imbalanced (Positive Reviews > Negative Reviews)

[3] Preprocessing

[3.1]. Preprocessing Review Text

Now that deduplication is finished for our data and 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]:
```python
# 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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along and he always can sing
the refrain. he's learned about whales, India, drooping roses:  i love all the new words this book  introduces and the
silliness of it all.  this is a classic book i am  willing to bet my son will STILL be able to recite from memory when
he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I prefer bolder taste.... imag
ine my surprise when I ordered 2 boxes - both were expired! One expired back in 2005 for gosh sakes.  I admit that Amaz
on agreed to credit me for cost plus part of shipping, but geez, 2 years expired!!!  I'm hoping to find local San Diego
area shoppe that carries pods so that I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing I do not think belon
gs in it is Canola oil. Canola or rapeseed is not someting a dog would ever find in nature and if it did find rapeseed
in nature and eat it, it would poison them. Today's Food industries have convinced the masses that Canola oil is a safe
and even better oil than olive or virgin coconut, facts though say otherwise. Until the late 70's it was poisonous unti
l they figured out a way to fix that. I still like it but it could be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of this product.<br /><br />Thi
ck, delicious.  Perfect.  3 ingredients: Water, Maltitol, Natural Maple Flavor.  PERIOD.  No chemicals.  No garbage.<br
/><br />Have numerous friends & family members hooked on this stuff.  My husband & son, who do NOT like "sugar free" pr
efer this over major label regular syrup.<br /><br />I use this as my SWEETENER in baking: cheesecakes, white brownies,
muffins, pumpkin pies, etc... Unbelievably delicious...<br /><br />Can you tell I like it? :)
==================================================
```

In [15]:
```python
# 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_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)
```

```
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of this product.<br /><br />Thi
ck, delicious.  Perfect.  3 ingredients: Water, Maltitol, Natural Maple Flavor.  PERIOD.  No chemicals.  No garbage.<br
/><br />Have numerous friends & family members hooked on this stuff.  My husband & son, who do NOT like "sugar free" pr
efer this over major label regular syrup.<br /><br />I use this as my SWEETENER in baking: cheesecakes, white brownies,
muffins, pumpkin pies, etc... Unbelievably delicious...<br /><br />Can you tell I like it? :)
```

In [16]:
```python
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-element
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

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

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

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

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along and he always can sing
the refrain. he's learned about whales, India, drooping roses:  i love all the new words this book  introduces and the
silliness of it all.  this is a classic book i am  willing to bet my son will STILL be able to recite from memory when
he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I prefer bolder taste.... imag
ine my surprise when I ordered 2 boxes - both were expired! One expired back in 2005 for gosh sakes.  I admit that Amaz
on agreed to credit me for cost plus part of shipping, but geez, 2 years expired!!!  I'm hoping to find local San Diego
area shoppe that carries pods so that I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing I do not think belon
gs in it is Canola oil. Canola or rapeseed is not someting a dog would ever find in nature and if it did find rapeseed
in nature and eat it, it would poison them. Today's Food industries have convinced the masses that Canola oil is a safe
and even better oil than olive or virgin coconut, facts though say otherwise. Until the late 70's it was poisonous unti
l they figured out a way to fix that. I still like it but it could be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of this product.Thick, deliciou
s.  Perfect.  3 ingredients: Water, Maltitol, Natural Maple Flavor.  PERIOD.  No chemicals.  No garbage.Have numerous f
riends & family members hooked on this stuff.  My husband & son, who do NOT like "sugar free" prefer this over major la
bel regular syrup.I use this as my SWEETENER in baking: cheesecakes, white brownies, muffins, pumpkin pies, etc... Unbe
lievably delicious...Can you tell I like it? :)
```

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)
```

In [18]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
```

```
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing I do not think belon
gs in it is Canola oil. Canola or rapeseed is not someting a dog would ever find in nature and if it did find rapeseed
in nature and eat it, it would poison them. Today is Food industries have convinced the masses that Canola oil is a saf
e and even better oil than olive or virgin coconut, facts though say otherwise. Until the late 70 is it was poisonous u
ntil they figured out a way to fix that. I still like it but it could be better.
==================================================
```

In [19]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_1500 = re.sub("\S*\d\S*", "", sent_1500).strip()
```

```
Great ingredients although, chicken should have been  rather than chicken broth, the only thing I do not think belongs
in it is Canola oil. Canola or rapeseed is not someting a dog would ever find in nature and if it did find rapeseed in
nature and eat it, it would poison them. Today is Food industries have convinced the masses that Canola oil is a safe a
nd even better oil than olive or virgin coconut, facts though say otherwise. Until the late  is it was poisonous until
they figured out a way to fix that. I still like it but it could be better.
```

In [20]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_4900 = re.sub('[^A-Za-z0-9]+', ' ', sent_4900)
```

```
Can t do sugar Have tried scores of SF Syrups NONE of them can touch the excellence of this product br br Thick delicio
us Perfect 3 ingredients Water Maltitol Natural Maple Flavor PERIOD No chemicals No garbage br br Have numerous friends
family members hooked on this stuff My husband son who do NOT like sugar free prefer this over major label regular syru
p br br I use this as my SWEETENER in baking cheesecakes white brownies muffins pumpkin pies etc Unbelievably delicious
br br Can you tell I like it
```

In [21]:
```
# https://gist.github.com/sebleier/554280
# removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, "br" is present in reviews
# 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', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't",
            'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:
```
# Combining all the above preprocessing steps
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
```

```
100%|██████████████████████████████| 364171/364171 [01:47<00:00, 3394.25it/s]
```

In [23]:

Out[23]:
```
'ca not sugar tried scores sf syrups none touch excellence product thick delicious perfect ingredients water maltitol n
atural maple flavor period no chemicals no garbage numerous friends family members hooked stuff husband son not like su
gar free prefer major label regular syrup use sweetener baking cheesecakes white brownies muffins pumpkin pies etc unbe
lievably delicious tell like'
```

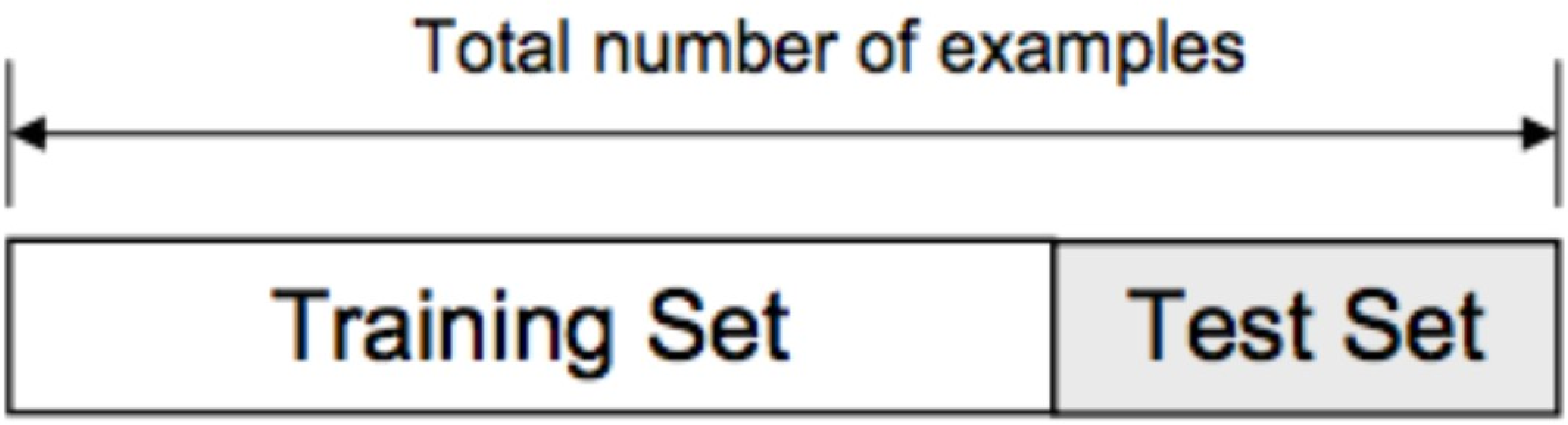
[3.2] Preprocessing Review Summary

In [24]:
```
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 stopwords)
```

```
100%|██████████████████████████████| 364171/364171 [01:11<00:00, 5121.12it/s]
```

Splitting Data - Train(70%) & Test(30%)

Source: [https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6 (https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6)](https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6)



In [25]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(preprocessed_reviews,final['Score'],test_size = 0.3, shuffle = False)

print("Total Reviews in Train set : ",len(x_train))
```

```
Total Reviews in Train set :  254919
Total Reviews in Test set :  109252
```

[4] Featurization

[4.1] BAG OF WORDS

Reference :

1. [https://en.wikipedia.org/wiki/Bag-of-words_model#Example_implementation (https://en.wikipedia.org/wiki/Bag-of-words_model#Example_implementation)](https://en.wikipedia.org/wiki/Bag-of-words_model#Example_implementation)
2. [http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html (http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html)](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html)

[4.2] Bi-Grams and n-Grams.

Reference : [https://en.wikipedia.org/wiki/Bag-of-words_model#n-gram_model (https://en.wikipedia.org/wiki/Bag-of-words_model#n-gram_model)](https://en.wikipedia.org/wiki/Bag-of-words_model#n-gram_model)

In [26]:
```
#bi-gram
#removing stop words like "not" should be avoided before building n-grams

count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
bigrams_train = (count_vect.fit_transform(x_train))
bigrams_test = (count_vect.transform(x_test))
print("some feature names ", count_vect.get_feature_names()[2564:2574])
print('='*50)

print("the type of count vectorizer ",type(bigrams_train))
print("the shape of out text BOW vectorizer for Train set ",bigrams_train.get_shape())
print("the number of unique words including both unigrams and bigrams in Train set ", bigrams_train.get_shape()[1])
print('='*50)
print("the shape of out text BOW vectorizer for Test set ",bigrams_test.get_shape())
print("the number of unique words including both unigrams and bigrams Test set ", bigrams_test.get_shape()[1])
```

```
some feature names  ['male', 'malt', 'malted', 'man', 'managed', 'mango', 'manner', 'manufactured', 'manufacturer', 'manufacturers']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer for Train set  (254919, 5000)
the number of unique words including both unigrams and bigrams in Train set   5000
==================================================
the shape of out text BOW vectorizer for Test set  (109252, 5000)
the number of unique words including both unigrams and bigrams Test set   5000
```

[4.3] TF-IDF

Reference :

1. https://en.wikipedia.org/wiki/Tf%E2%80%93idf#Definition (https://en.wikipedia.org/wiki/Tf%E2%80%93idf#Definition)
2. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)

In [27]:

```
tfidf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
tfidf_bigrams_train = (tfidf_vect.fit_transform(x_train))
tfidf_bigrams_test = (tfidf_vect.transform(x_test))
print("some feature names ", tfidf_vect.get_feature_names()[2564:2574])
print('='*50)

print("the type of count vectorizer ",type(tfidf_bigrams_train))
print("the shape of out text BOW vectorizer for Train set ",tfidf_bigrams_train.get_shape())
print("the number of unique words including both unigrams and bigrams in Train set ", tfidf_bigrams_train.get_shape()[1])
print('='*50)
print("the shape of out text BOW vectorizer for Test set ",tfidf_bigrams_test.get_shape())
print("the number of unique words including both unigrams and bigrams Test set ", tfidf_bigrams_test.get_shape()[1])
```

```
some feature names  ['male', 'malt', 'malted', 'man', 'managed', 'mango', 'manner', 'manufactured', 'manufacturer', 'ma
nufacturers']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer for Train set  (254919, 5000)
the number of unique words including both unigrams and bigrams in Train set  5000
==================================================
the shape of out text BOW vectorizer for Test set  (109252, 5000)
the number of unique words including both unigrams and bigrams Test set  5000
```

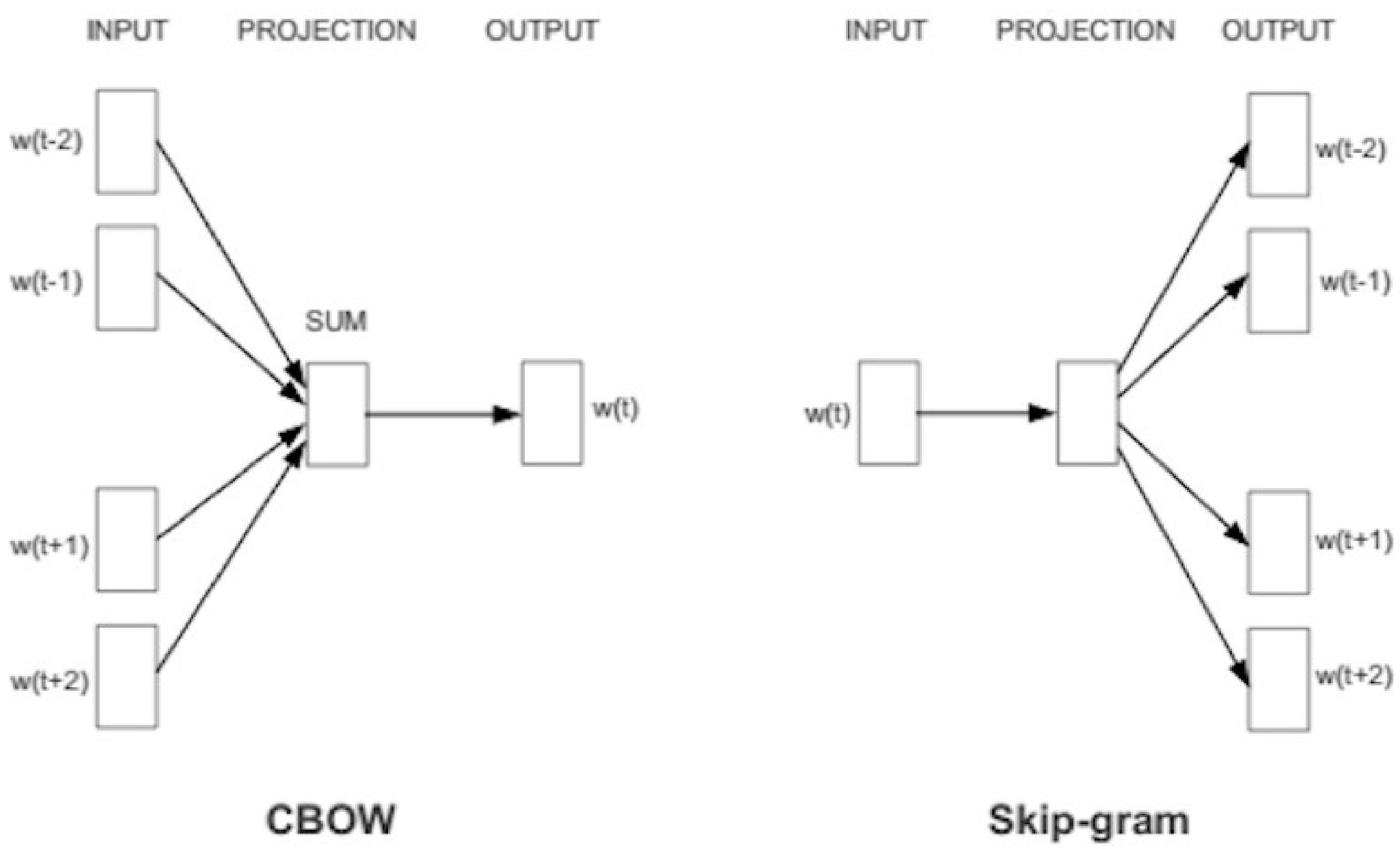
[4.4] Word2Vec

In [28]:

```
# Train our own Word2Vec model using preprocessed reviews
sentancesListTrain=[]
for eachSentance in x_train:
    sentancesListTrain.append(eachSentance.split())
sentancesListTest=[]
for eachSentance in x_test:
```

Reference:

1. https://towardsdatascience.com/a-beginners-guide-to-word-embedding-with-gensim-word2vec-model-5970fa56cc92 (https://towardsdatascience.com/a-beginners-guide-to-word-embedding-with-gensim-word2vec-model-5970fa56cc92)
2. https://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/ (https://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/)



In [29]:

```
w2v_model=Word2Vec(sentancesListTrain,min_count=5,size=50, workers=-1)
print(w2v_model.wv.most_similar('tasty'))
print('='*50)
```

```
[('diner', 0.571789026260376), ('gunky', 0.5230973958969116), ('itches', 0.5199359655380249), ('ranging', 0.51908731460
57129), ('parmesean', 0.502068817615509), ('distracts', 0.49597108364105225), ('roses', 0.48716795444488525), ('stassen
', 0.4850645065307617), ('slouch', 0.48224207758903503), ('isafe', 0.47921720147132874)]
==================================================
[('accross', 0.5421061515808105), ('attended', 0.5381463766098022), ('favorable', 0.5072031021118164), ('reside', 0.502
4747252464294), ('rubber', 0.487970411777749634), ('vacuum', 0.48725467920303345), ('decor', 0.48459041118621826), ('reg
rettably', 0.4845353066921234), ('cashew', 0.4783554971218109), ('subsist', 0.47698765993118286)]
```

In [30]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
```

```
number of words that occured minimum 5 times  28594
sample words :  ['account', 'couple', 'needs', 'value', 'loaded', 'chemical', 'fillers', 'irregular', 'drawback', 'surp
rising']
```

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

[4.4.1.1] Avg W2v

In [31]:
```
# average Word2Vec
# computing average word2vec for each review.
trainWord2Vectors = [] # the avg-w2v for each train sentence/review is stored in this list
for eachSentance in tqdm(sentancesListTrain):
    sentanceVector = np.zeros(50) # as word vectors are of zero length 50
    validWordCounts =0 # num of words with a valid vector in the sentence/review
    for eachWord in eachSentance:
        if eachWord in w2v_words:
            vector = w2v_model.wv[eachWord]
            sentanceVector += vector
            validWordCounts += 1
    if validWordCounts != 0:
        sentanceVector /= validWordCounts
    trainWord2Vectors.append(sentanceVector)
print(len(trainWord2Vectors))
```

```
100%|██████████████████████████████| 254919/254919 [09:16<00:00, 457.67it/s]

254919
50
```

In [32]:
```
testWord2Vectors = []; # the avg-w2v for each test sentence/review is stored in this list
for eachSentance in tqdm(sentancesListTest):
    sentanceVector = np.zeros(50)
    validWordCounts =0
    for eachWord in eachSentance:
        if eachWord in w2v_words:
            vector = w2v_model.wv[eachWord]
            sentanceVector += vector
            validWordCounts += 1
    if validWordCounts != 0:
        sentanceVector /= validWordCounts
    testWord2Vectors.append(sentanceVector)
print(len(testWord2Vectors))
```

```
100%|██████████████████████████████| 109252/109252 [04:23<00:00, 415.06it/s]

109252
50
```

[4.4.1.2] TFIDF weighted W2v

In [34]:
```
tfidfW2VModel = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
tfidfW2VModelVectors = tfidfW2VModel.fit_transform(x_train)
# creating hashmap with word as key and inverse document frequency as value
```

In [35]:
```
# TF-IDF weighted Word2Vec
tfidfWords = tfidfW2VModel.get_feature_names() # tfidf words

trainTfidfWord2Vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
for eachSentance in tqdm(sentancesListTrain):
    sentanceVector = np.zeros(50) # as word vectors are of zero length
    weightedSum =0; # num of words with a valid vector in the sentence/review
    for eachWord in eachSentance:
        if eachWord in w2v_words and eachWord in tfidfWords:
            vector = w2v_model.wv[eachWord]
            tf_idf = wordsHashMap[eachWord]*(eachSentance.count(eachWord)/len(eachSentance))
            sentanceVector += (vector * tf_idf)
            weightedSum += tf_idf
    if weightedSum != 0:
        sentanceVector /= weightedSum
```

```
100%|██████████████████████████████| 254919/254919 [17:36<00:00, 241.35it/s]
```

In [36]:
```
print(len(trainTfidfWord2Vectors))
```

```
254919
50
```

In [37]:
```
testTfidfWord2Vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
for eachSentance in tqdm(sentancesListTest):
    sentanceVector = np.zeros(50) # as word vectors are of zero length
    weightedSum =0; # num of words with a valid vector in the sentence/review
    for eachWord in eachSentance:
        if eachWord in w2v_words and eachWord in tfidfWords:
            vector = w2v_model.wv[eachWord]
            tf_idf = wordsHashMap[eachWord]*(eachSentance.count(eachWord)/len(eachSentance))
            sentanceVector += (vector * tf_idf)
            weightedSum += tf_idf
    if weightedSum != 0:
        sentanceVector /= weightedSum
    testTfidfWord2Vectors.append(sentanceVector)
print(len(testTfidfWord2Vectors))
```

```
100%|██████████████████████████████████████████| 109252/109252 [08:27<00:00, 215.30it/s]

109252
50
```

Decision Trees

Source: Wikipedia

A decision tree is a simple representation for classifying examples. For this section, assume that all of the input features have finite discrete domains, and there is a single target feature called the "classification". Each element of the domain of the classification is called a class. A decision tree or a classification tree is a tree in which each internal (non-leaf) node is labeled with an input feature. The arcs coming from a node labeled with an input feature are labeled with each of the possible values of the target or output feature or the arc leads to a subordinate decision node on a different input feature. Each leaf of the tree is labeled with a class or a probability distribution over the classes, signifying that the data set has been classified by the tree into either a specific class, or into a particular probability distribution (which, if the decision tree is well-constructed, is skewed towards certain subsets of classes).

A tree is built by splitting the source set, constituting the root node of the tree, into subsets - which constitute the successor children. The splitting is based on a set of splitting rules based on classification features. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node has all the same values of the target variable, or when splitting no longer adds value to the predictions. This process of top-down induction of decision trees (TDIDT) is an example of a greedy algorithm, and it is by far the most common strategy for learning decision trees from data.

Metrics used in Desicion Trees

- **Gini impurity** I_G

> Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. The Gini impurity can be computed by summing the probability p_i of an item with label i being chosen times the probability $\sum_{k\neq i} p_k = 1 - p_i$ of a mistake in categorizing that item. It reaches its minimum (zero) when all cases in the node fall into a single target category.

> To compute Gini impurity for a set of items with J classes, suppose $i \in 1, 2, \ldots, J$, and let p_i be the fraction of items labeled with class i in the set.
> Then Gini Impurity is $I_G(p) = 1 - \sum_{n=1}^{J} p_i^2$

- **Information gain**

> Information gain is based on the concept of entropy and information content from information theory.
> Entropy is defined as below $H(T) = -\sum_{n=1}^{J} p_i \log_2 p_i$
>
> where $p_1, p_2, \ldots$ are fractions that add up to 1 and represent the percentage of each class present in the child node that results from a split in the tree.
> $= -\sum_{n=1}^{J} p_i - \sum_a p(a) - \sum_{i=1}^{J} -Pr(i|a) \log_2 Pr(i|a)$

In [169]:
```
#source -
#    1. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
#    2. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
def gridSearchDecisionTrees(data,label):
    clf = GridSearchCV(DecisionTreeClassifier(criterion='gini',class_weight="balanced"),
                       param_grid = {"max_depth":[4,6, 8, 9,10,12,14,17],"min_samples_split":[2,10,20,30,40,50]},
                       return_train_score = True,
                       scoring='roc_auc',
                       n_jobs=-1,
                       cv=5)
    clf.fit(data,label)
```

In [170]:
```
#source - https://seaborn.pydata.org/generated/seaborn.heatmap.html
import seaborn as sns

def plotAUCvsHyperParam(model):
    plt.figure(figsize=(10,10))
    f,(ax1,ax2) = plt.subplots(1,2,figsize=(16,5))

    testScore = model.cv_results_["mean_test_score"]
    testScore = testScore.reshape(len(model.param_grid["max_depth"]),len(model.param_grid["min_samples_split"]))
    g1 = sns.heatmap(testScore,
                     annot = True,
                     fmt=".4f",
                     ax = ax1,
                     cmap = sns.color_palette("Paired"),
                     xticklabels=model.param_grid["min_samples_split"],
                     yticklabels=model.param_grid["max_depth"])
    g1.set_xlabel("min_samples_split")
    g1.set_ylabel("max_depth")
    title = "Best Cross Validation Score = "+\
            str(model.best_score_)+"\n"\
            " at "+\
            "max_depth "+str(model.best_params_["max_depth"])+\
            " , "+\
            "min_samples_split "+str(model.best_params_["min_samples_split"])
    ax1.title.set_text(title)
    ax1.title.set_fontsize(15)

    trainScore = model.cv_results_["mean_train_score"]
    trainScore = trainScore.reshape(len(model.param_grid["max_depth"]),len(model.param_grid["min_samples_split"]))
    indices = np.unravel_index(np.argmax(trainScore, axis=None), trainScore.shape)
    g2 = sns.heatmap(trainScore,
                     annot = True,
                     fmt=".4f",
                     ax = ax2,
                     cmap = sns.color_palette("Paired"),
                     xticklabels=model.param_grid["min_samples_split"],
                     yticklabels=model.param_grid["max_depth"])
    g2.set_xlabel("min_samples_split")
    g2.set_ylabel("max_depth")
    title = "Best Train Score = "+\
            str(trainScore.max())+"\n"\
            " at "+\
            "max_depth "+str(model.param_grid["max_depth"][indices[0]])+\
            " , "+\
            "min_samples_split "+str(model.param_grid["min_samples_split"][indices[1]])
    ax2.title.set_text(title)
```

In [204]:
```
#source - https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
from sklearn import metrics
def rocCurve(model,trainData,trainLabels,testData,testLabels):
    predictedProbabilities = model.predict_proba(testData)
    fpr, tpr, thresholds = metrics.roc_curve(testLabels, predictedProbabilities[:,1])
    plt.plot(fpr,tpr,label='Test AUC is %0.3f' %(metrics.auc(fpr,tpr)))
    predictedProbabilities = model.predict_proba(trainData)
    fpr, tpr, thresholds = metrics.roc_curve(trainLabels, predictedProbabilities[:,1])
    plt.plot(fpr,tpr,label='Train AUC is %0.3f' %(metrics.auc(fpr,tpr)))
    plt.legend()
    plt.show()
```

In [172]:
```
#source - https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
from sklearn.metrics import confusion_matrix
def confusionMatrix(model,testData,testLabels):
    tn, fp, fn, tp = confusion_matrix(testLabels,model.predict(testData)).ravel()
    sns.heatmap([[fn,tn],[fp,tp]],yticklabels=["Actual 0","Actual 1"],\
```

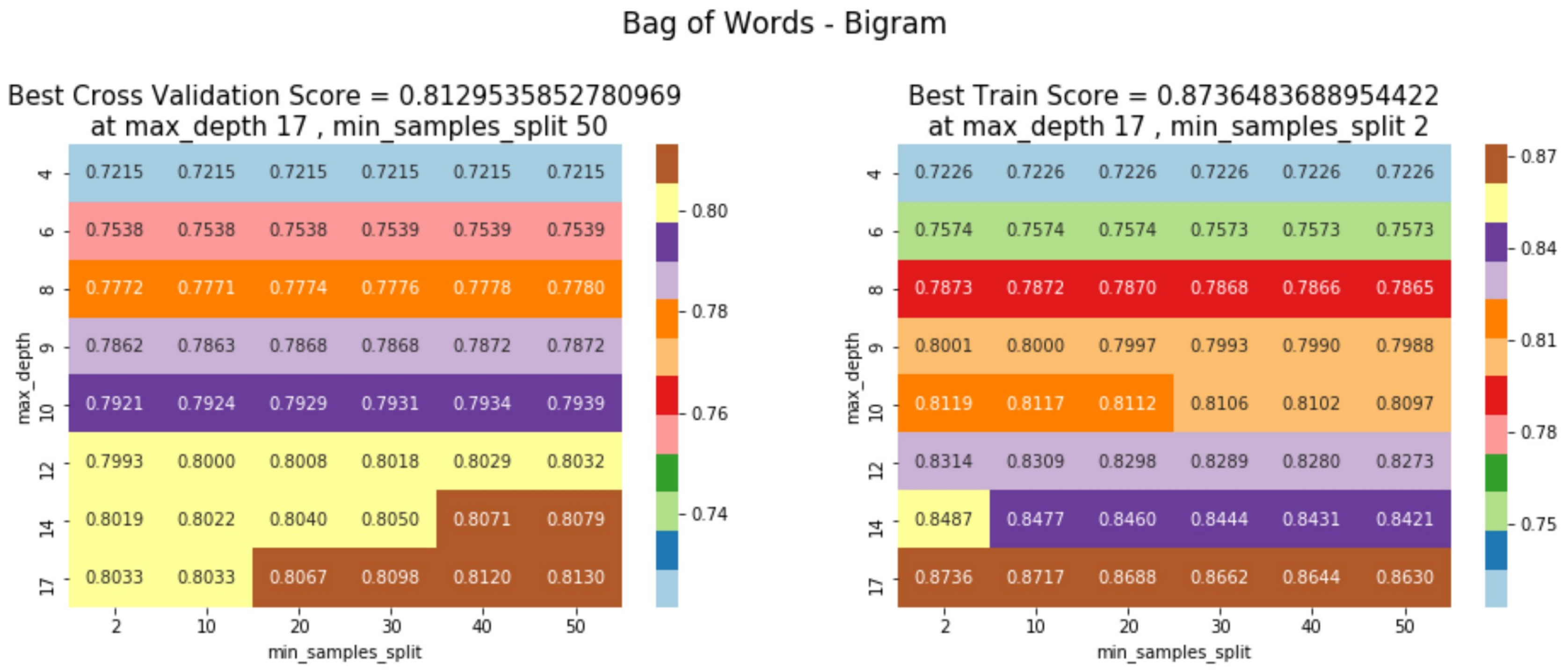
Applying Decision Trees

[5.1] Applying Decision Trees on BOW

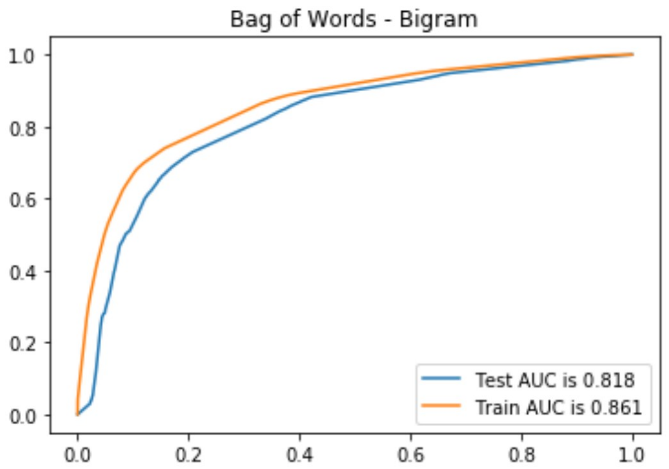
In [173]:

In [174]:
```
plotAUCvsHyperParam(bigram_model)
```

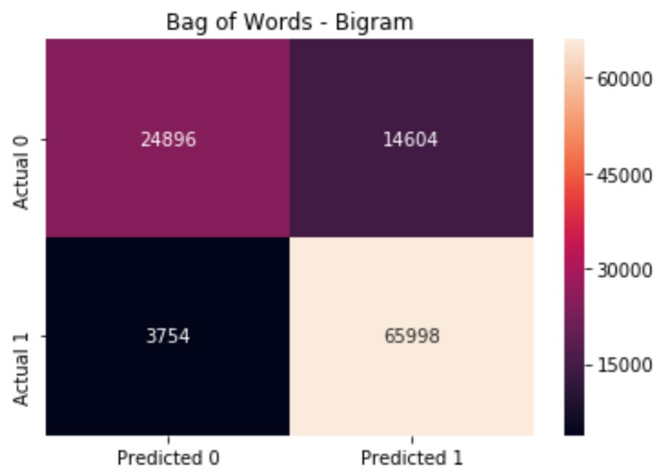
Out[174]:
```
Text(0.49, 1.1, 'Bag of Words - Bigram')

<Figure size 720x720 with 0 Axes>
```



In [205]:
```
plt.title("Bag of Words - Bigram")
```



In [176]:
```
plt.title("Bag of Words - Bigram")
```



[5.1.1] Top 20 important features from

In [177]:
```
print(pd.DataFrame(data = bigram_model.best_estimator_.feature_importances_,index=count_vect.get_feature_names()).sort_val
```

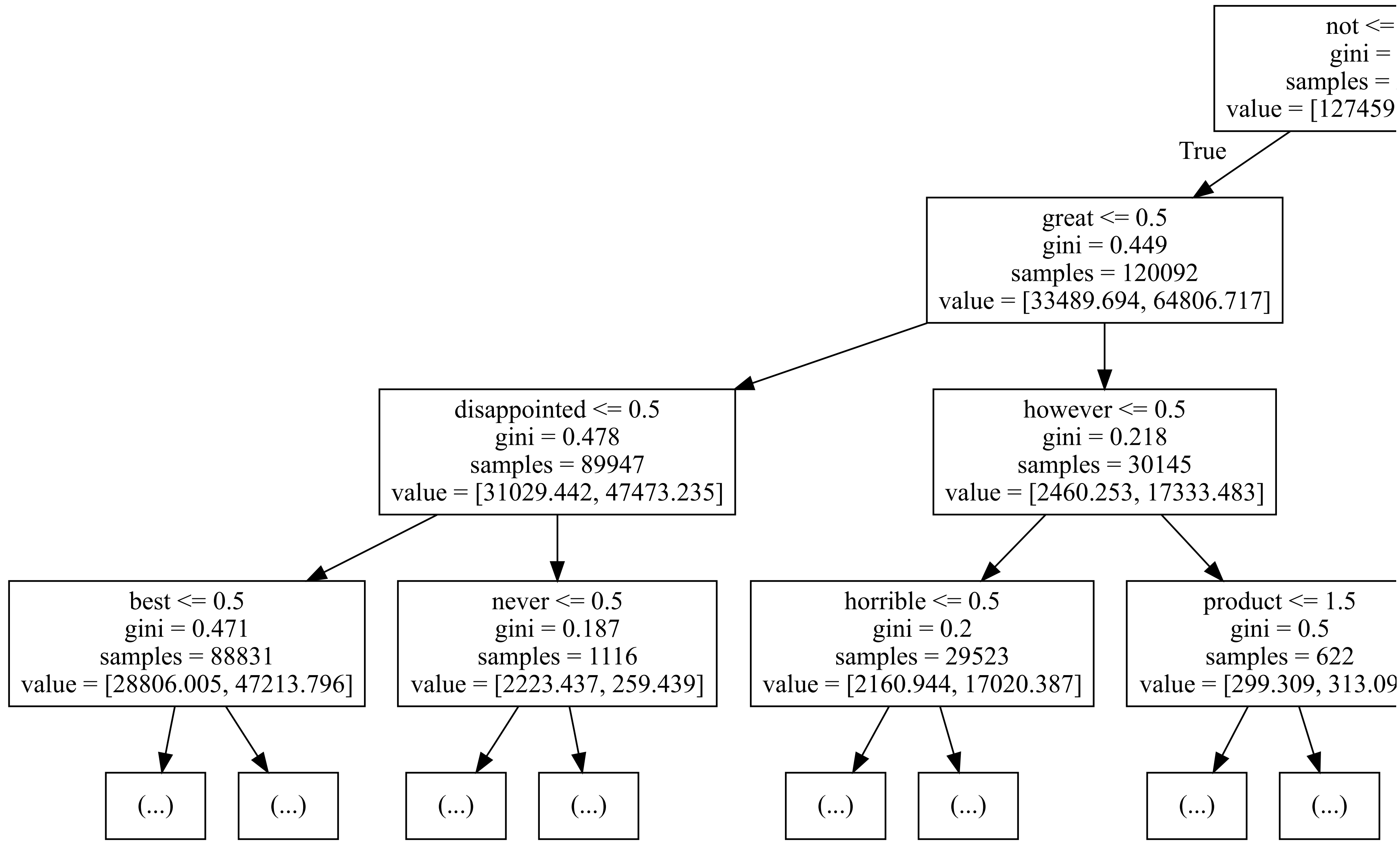
```
                    0
not              0.188817
great            0.122561
best             0.062953
delicious        0.046635
love             0.040331
disappointed     0.037506
perfect          0.032831
good             0.031984
loves            0.027037
excellent        0.018881
favorite         0.018052
bad              0.017709
wonderful        0.014723
thought          0.013582
money            0.012877
easy             0.012655
not good         0.010911
horrible         0.009508
unfortunately    0.009038
awful            0.008364
```

[5.1.2] Graphviz visualization of Decision Tree on BOW

In [178]:
```
from sklearn import tree
import graphviz
```

In [179]:

Out[179]:



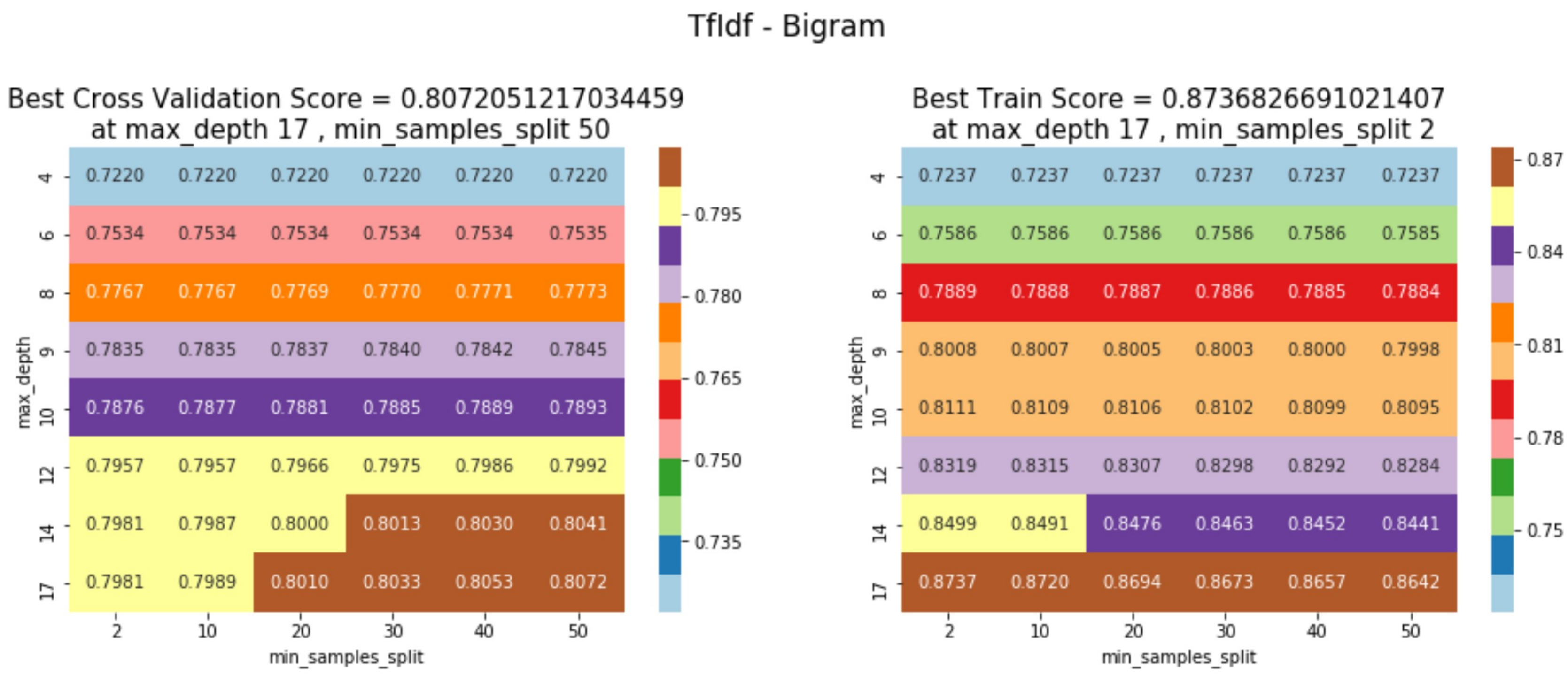
[5.2] Applying Decision Trees on TFIDF

In [180]:

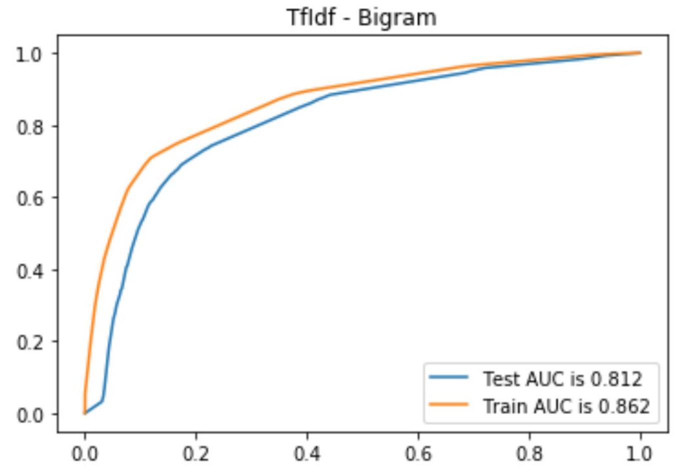
In [181]:
```
plotAUCvsHyperParam(tfidf_bigram_model)
```

Out[181]:
```
Text(0.49, 1.1, 'TfIdf - Bigram')
```

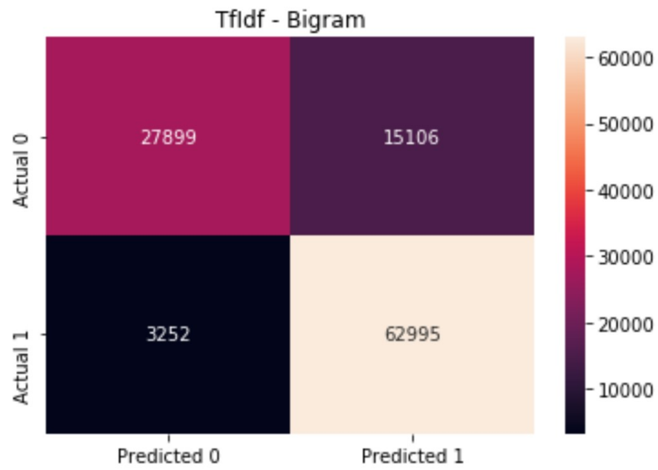
```
<Figure size 720x720 with 0 Axes>
```



In [206]:
```
plt.title("TfIdf - Bigram")
```



In [183]:
```
plt.title("TfIdf - Bigram")
```



[5.2.1] Top 20 important features from

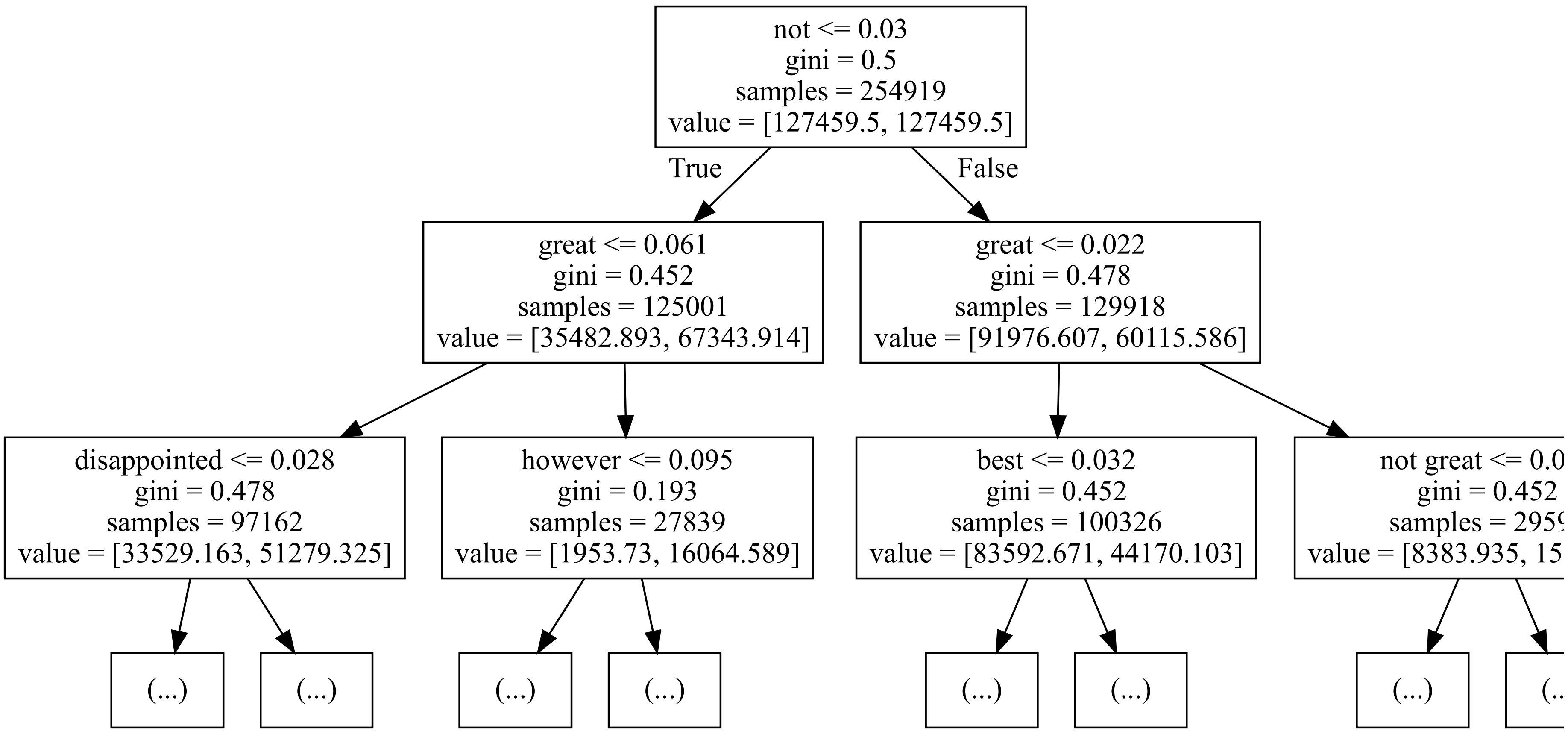
In [184]:
```
print(pd.DataFrame(data = tfidf_bigram_model.best_estimator_.feature_importances_,index=tfidf_vect.get_feature_names()).sc
```

```
                     0
not           0.178170
great         0.125379
best          0.061648
delicious     0.047113
love          0.042591
disappointed  0.039842
good          0.033851
perfect       0.033623
loves         0.029414
favorite      0.018456
excellent     0.017943
bad           0.017919
wonderful     0.016071
nice          0.013852
thought       0.013502
easy          0.011549
worst         0.009214
reviews       0.009062
however       0.008093
horrible      0.008051
```

[5.2.2] Graphviz visualization of Decision Tree on TFIDF

In [185]:
```
graphviz.Source(tree.export_graphviz(tfidf_bigram_model.best_estimator_,feature_names=tfidf_vect.get_feature_names(),max_(
```

Out[185]:



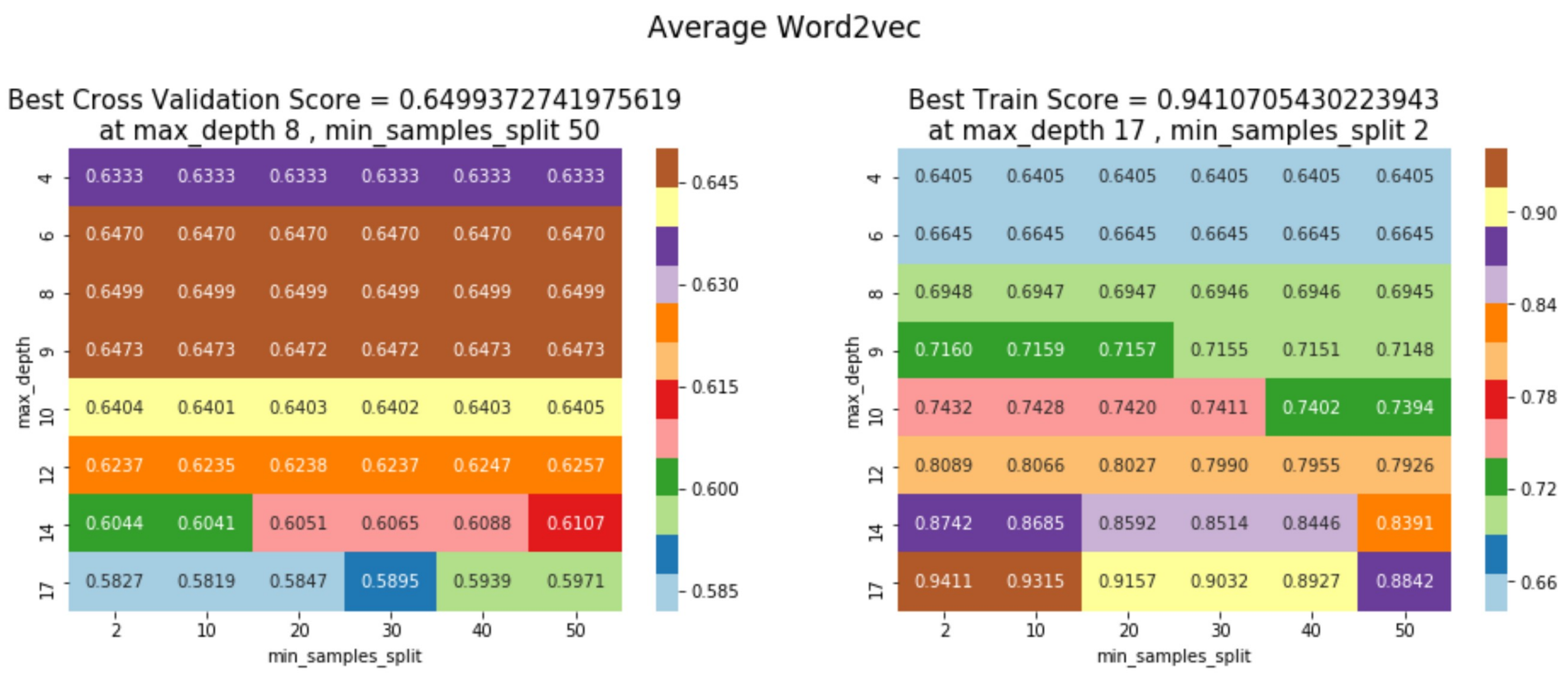
[5.3] Applying Decision Trees on AVG W2V

In [186]:

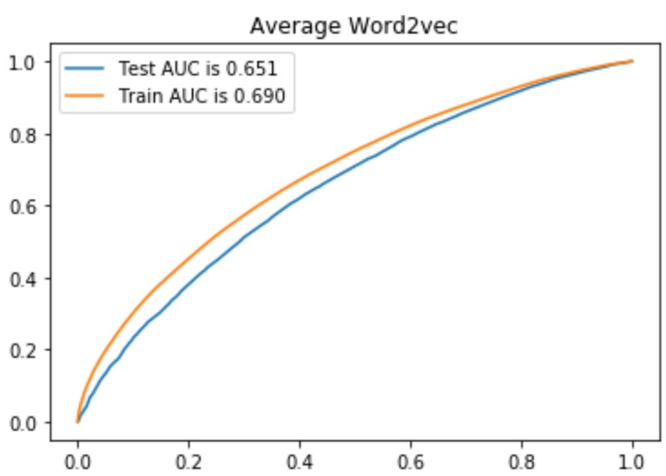
In [187]:
```
plotAUCvsHyperParam(avgW2v_model)
```

Out[187]: Text(0.49, 1.1, 'Average Word2vec')

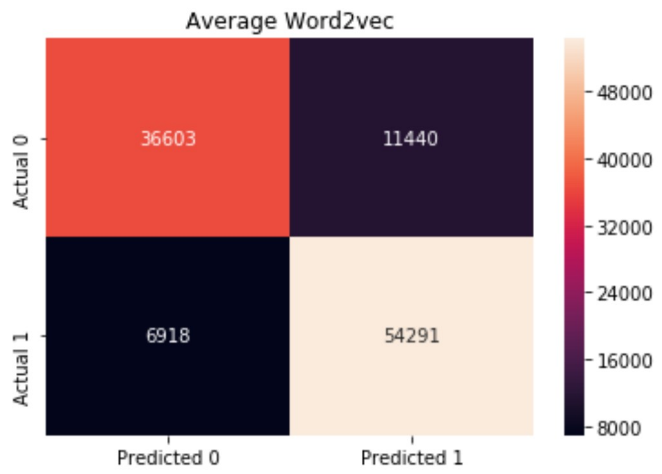
<Figure size 720x720 with 0 Axes>



In [207]:
```
plt.title("Average Word2vec")
```



In [189]:
```
plt.title("Average Word2vec")
```



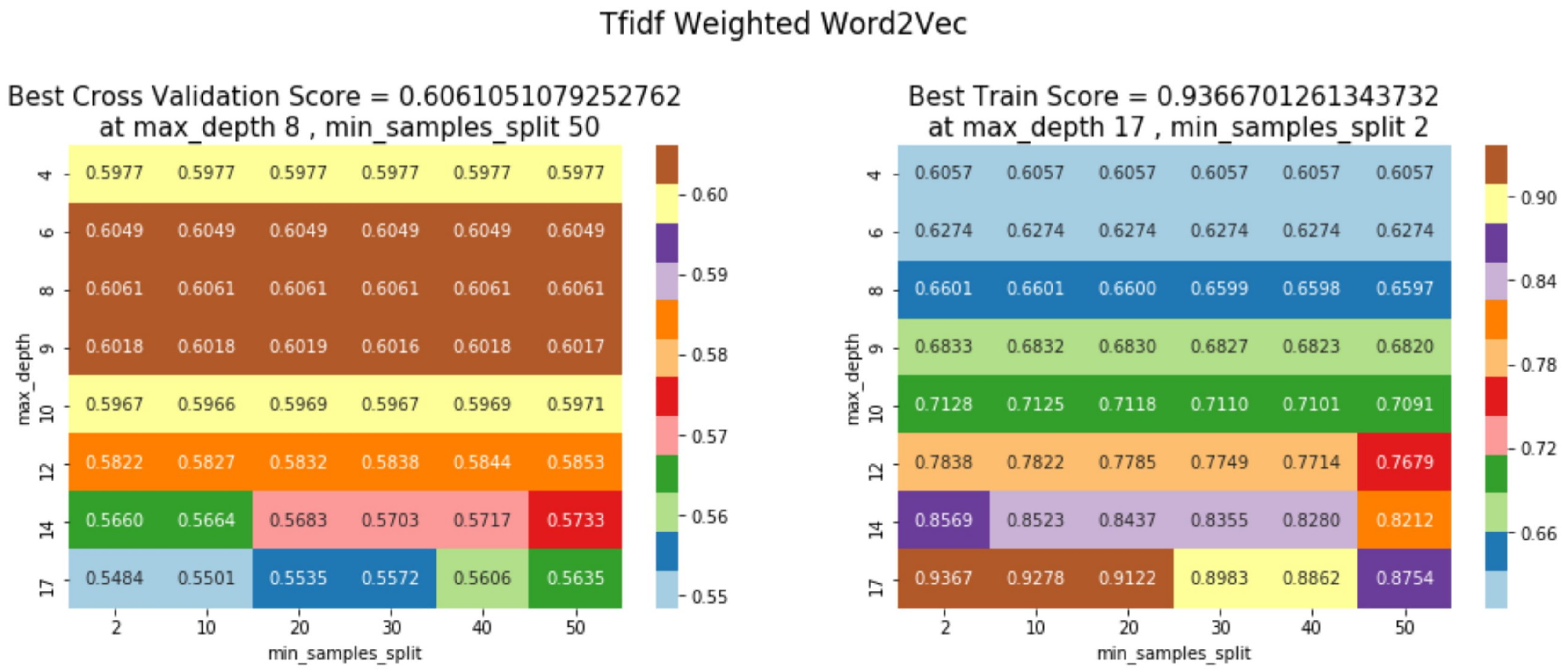
[5.4] Applying Decision Trees on TFIDF W2V

In [190]:

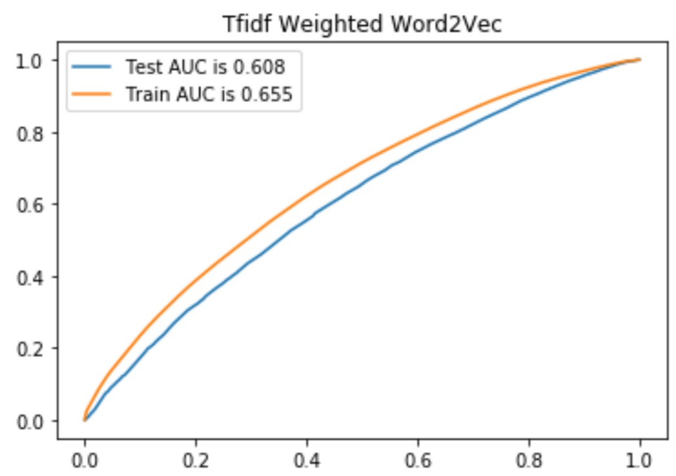
In [191]:
```
plotAUCvsHyperParam(tfidfW2v_model)
```

Out[191]:
```
Text(0.49, 1.1, 'Tfidf Weighted Word2Vec')
```

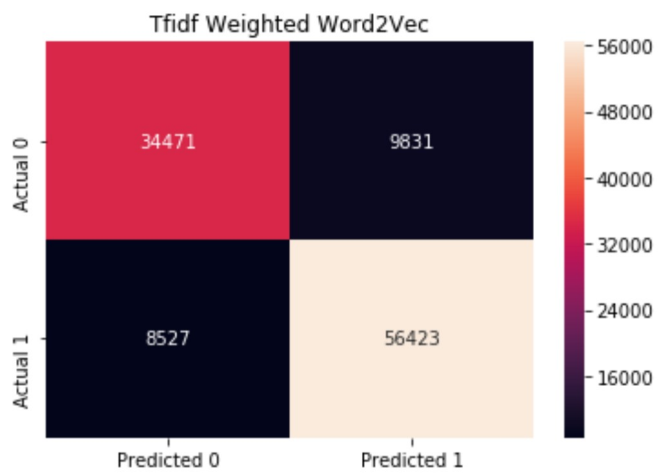
```
<Figure size 720x720 with 0 Axes>
```



In [208]:
```
plt.title("Tfidf Weighted Word2Vec")
```



In [193]:
```
plt.title("Tfidf Weighted Word2Vec")
```



[6] Feature Engineering

[6.1] TfIdf Weighted W2V Vectorization

In [155]:
```
#appending each summary with each review text
reviewWithSummary = []
for eachReview,eachSummary in tqdm(zip(preprocessed_reviews,preprocessed_summaries)):
```

```
0it [00:00, ?it/s]

17678it [00:00, 175551.73it/s]

49281it [00:00, 202223.01it/s]

81380it [00:00, 227386.91it/s]

116352it [00:00, 253720.32it/s]

153213it [00:00, 279399.11it/s]

203228it [00:00, 321547.87it/s]

249843it [00:00, 354505.34it/s]

288350it [00:00, 362545.47it/s]

364171it [00:01, 361655.80it/s]
```

In [156]:

```
length of feature engineered reviews :364171
```

In [157]:

In [159]:
```
tfidfW2VModel_FE = TfidfVectorizer(ngram_range=(1,1), min_df=10, max_features=5000)
tfidfW2VModelVectors_FE = tfidfW2VModel_FE.fit_transform(x_train)
# creating hashmap with word as key and inverse document frequency as value
```

In [160]:
```
# Train our own Word2Vec model using preprocessed reviews
sentancesListTrain_FE=[]
for eachSentance in x_train:
    sentancesListTrain_FE.append(eachSentance.split())
sentancesListTest_FE=[]
for eachSentance in x_test:
```

In [161]:
```
# TF-IDF weighted Word2Vec
tfidfWords_FE = tfidfW2VModel_FE.get_feature_names() # tfidf words

trainTfidfWord2Vectors_FE = []; # the tfidf-w2v for each sentence/review is stored in this list
for eachSentance in tqdm(sentancesListTrain_FE):
    sentanceVector = np.zeros(50) # as word vectors are of zero length
    weightedSum =0; # num of words with a valid vector in the sentence/review
    for eachWord in eachSentance:
        if eachWord in w2v_words and eachWord in tfidfWords:
            vector = w2v_model.wv[eachWord]
            tf_idf = wordsHashMap_FE[eachWord]*(eachSentance.count(eachWord)/len(eachSentance))
            sentanceVector += (vector * tf_idf)
            weightedSum += tf_idf
    if weightedSum != 0:
        sentanceVector /= weightedSum
```

```
0%|                                                  | 0/254919 [00:00<?, ?it/s]

0%|                                         | 2/254919 [00:00<3:40:20, 19.28it/s]

0%|                                        | 19/254919 [00:00<2:42:09, 26.20it/s]

0%|                                        | 30/254919 [00:00<2:06:40, 33.54it/s]

0%|                                        | 37/254919 [00:00<1:49:43, 38.71it/s]

0%|                                        | 59/254919 [00:00<1:22:45, 51.33it/s]

0%|                                        | 90/254919 [00:00<1:02:03, 68.43it/s]

0%|                                        | 114/254919 [00:00<48:50, 86.95it/s]

0%|                                       | 133/254919 [00:00<41:12, 103.05it/s]
```

In [162]:
```
print(len(trainTfidfWord2Vectors_FE))
```

```
254919
50
```

In [163]:

```
testTfidfWord2Vectors_FE = []; # the tfidf-w2v for each sentence/review is stored in this list
for eachSentance in tqdm(sentancesListTest_FE):
    sentanceVector = np.zeros(50) # as word vectors are of zero length
    weightedSum =0; # num of words with a valid vector in the sentence/review
    for eachWord in eachSentance:
        if eachWord in w2v_words and eachWord in tfidfWords:
            vector = w2v_model.wv[eachWord]
            tf_idf = wordsHashMap_FE[eachWord]*(eachSentance.count(eachWord)/len(eachSentance))
            sentanceVector += (vector * tf_idf)
            weightedSum += tf_idf
    if weightedSum != 0:
        sentanceVector /= weightedSum
    testTfidfWord2Vectors_FE.append(sentanceVector)
print(len(testTfidfWord2Vectors_FE))
```

```
  0%|                                                              | 0/109252 [00:00<?, ?it/s]
  0%|                                                    | 21/109252 [00:00<08:43, 208.48it/s]
  0%|                                                    | 38/109252 [00:00<09:22, 194.07it/s]
  0%|                                                    | 64/109252 [00:00<08:43, 208.65it/s]
  0%|                                                    | 82/109252 [00:00<09:10, 198.30it/s]
  0%|                                                    | 98/109252 [00:00<10:13, 177.82it/s]
  0%|                                                   | 129/109252 [00:00<09:02, 201.22it/s]
  0%|                                                   | 153/109252 [00:00<08:36, 211.08it/s]
  0%|                                                   | 174/109252 [00:00<09:05, 200.08it/s]
```

[6.2] Applying Decision Tree on Feature Engineered Reviews

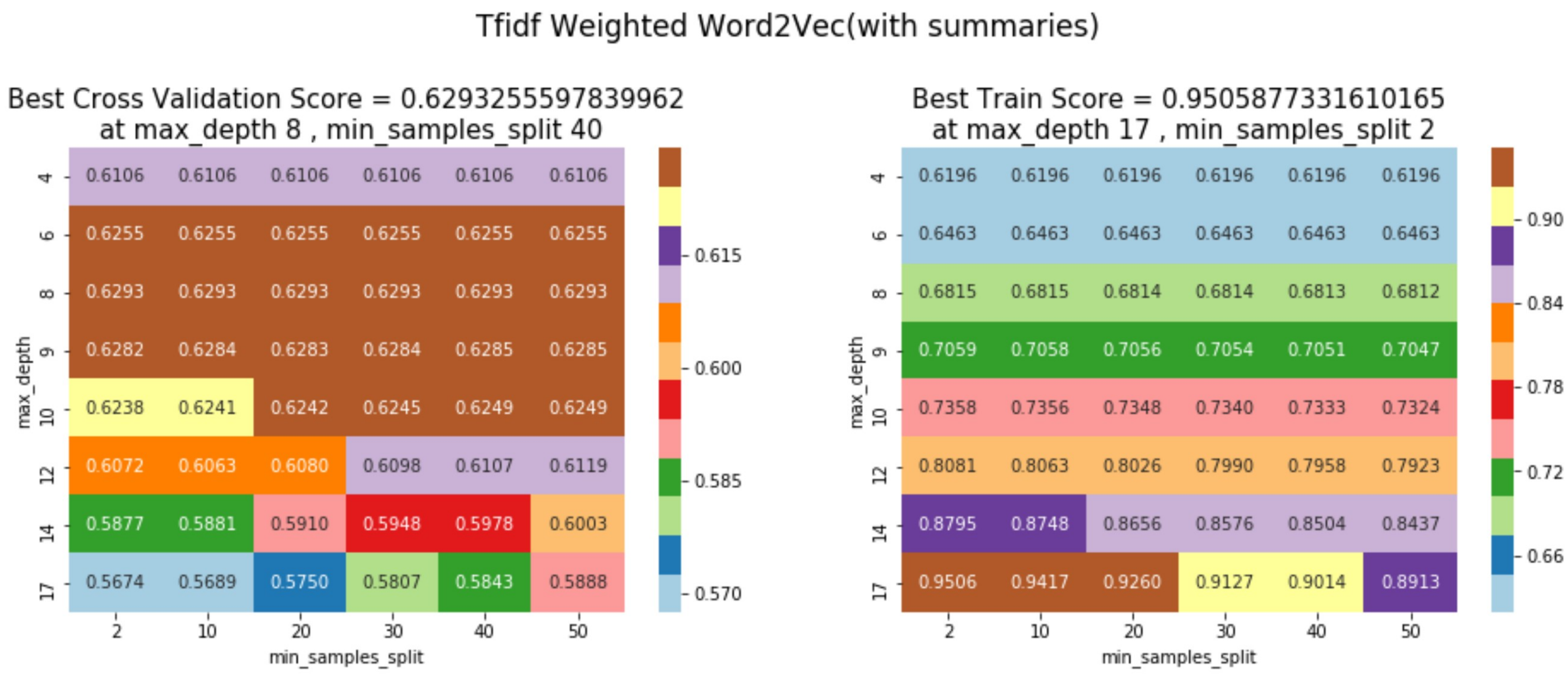
In [194]:

In [195]:

```
plotAUCvsHyperParam(tfidfW2v_FE_model)
```

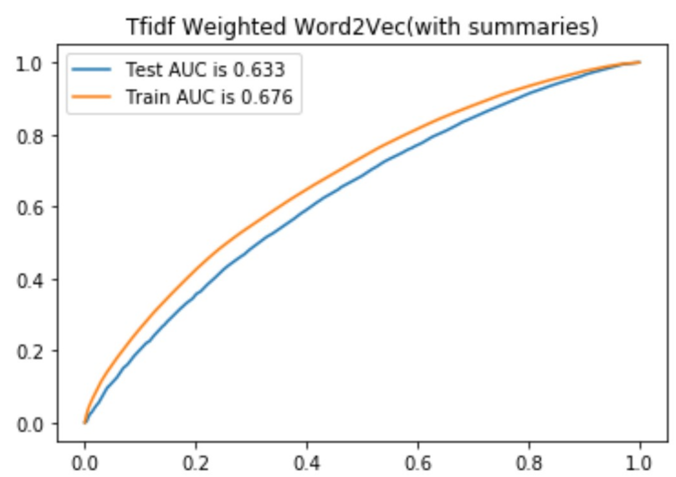
Out[195]:

```
Text(0.49, 1.1, 'Tfidf Weighted Word2Vec(with summaries)')

<Figure size 720x720 with 0 Axes>
```

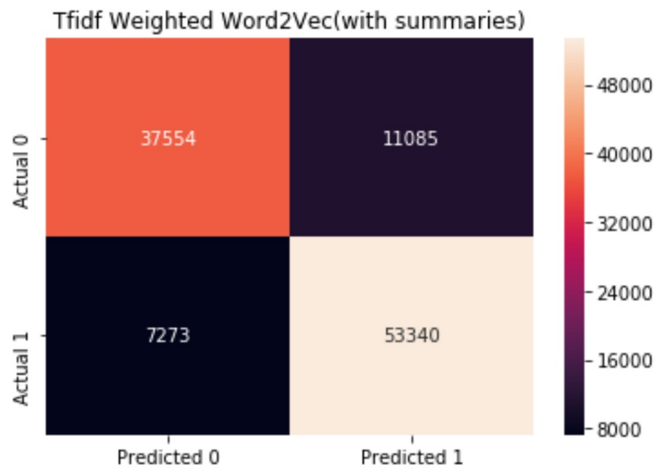


In [209]:

```
plt.title("Tfidf Weighted Word2Vec(with summaries)")
```



In [197]:
```
plt.title("Tfidf Weighted Word2Vec(with summaries)")
```



[7] Conclusions

In [214]:
```
from prettytable import PrettyTable
table = PrettyTable()
table.field_names = ["Vectoriser","max_depth || min_samples_split","Train AUC Score","Test AUC score"]
table.add_row(["Bag of Words - Bigram",
               bigram_model.best_params_,
               bigram_model.score(bigrams_train,y_train),
               bigram_model.score(bigrams_test,y_test)])
table.add_row(["TfIdf - Bigram",
               tfidf_bigram_model.best_params_,
               tfidf_bigram_model.score(tfidf_bigrams_train,y_train),
               tfidf_bigram_model.score(tfidf_bigrams_test,y_test)])
table.add_row(["Average Word2Vec",
               avgW2v_model.best_params_,
               avgW2v_model.score(trainWord2Vectors,y_train),
               avgW2v_model.score(testWord2Vectors,y_test)])
table.add_row(["Tfidf Weighted Word2Vec",
               tfidfW2v_model.best_params_,
               tfidfW2v_model.score(trainTfidfWord2Vectors,y_train),
               tfidfW2v_model.score(testTfidfWord2Vectors,y_test)])
table.add_row(["Tfidf Weighted Word2Vec"+"\n"+"(With Summary)",
               tfidfW2v_FE_model.best_params_,
               tfidfW2v_FE_model.score(trainTfidfWord2Vectors_FE,y_train),
```

In [215]:

```
+-------------------------+-----------------------------------------------+---------------------+--------------------+
|        Vectoriser       |        max_depth || min_samples_split        |   Train AUC Score   |   Test AUC score   |
+-------------------------+-----------------------------------------------+---------------------+--------------------+
|  Bag of Words - Bigram  | {'max_depth': 17, 'min_samples_split': 50} | 0.8613885055669276 | 0.817776747644543  |
|      TfIdf - Bigram     | {'max_depth': 17, 'min_samples_split': 50} | 0.862488828187043  | 0.8115546853944768 |
|     Average Word2Vec    | {'max_depth': 8, 'min_samples_split': 50}  | 0.6902326896233738 | 0.6506618584358824 |
| Tfidf Weighted Word2Vec | {'max_depth': 8, 'min_samples_split': 50}  | 0.6551315749869804 | 0.6075787138242026 |
| Tfidf Weighted Word2Vec | {'max_depth': 8, 'min_samples_split': 40}  | 0.6763657770661049 | 0.6334233228548819 |
|      (With Summary)     |                                               |                     |                    |
+-------------------------+-----------------------------------------------+---------------------+--------------------+
```

From the above table we can observe that if we include summary text as features in our data set, accuracy is improving marginally