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='
```

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]:
```
# 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]:
```
# 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]:
```
# 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)
```

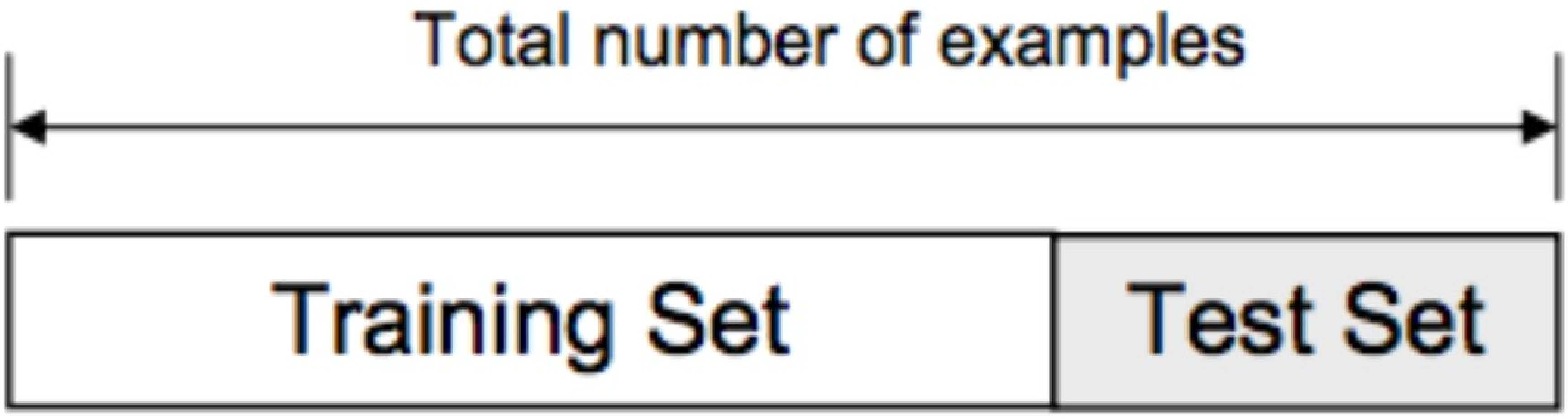
```
100%|██████████████████████████████| 364171/364171 [02:34<00:00, 2359.49it/s]
```

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'
```

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 [24]:
```
from sklearn.model_selection import train_test_split
```

[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 [28]:
```
#bi-gram
#removing stop words like "not" should be avoided before building n-grams
#CountVectorizer documentation http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVect
from sklearn.preprocessing import StandardScaler
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10)
standardizer = StandardScaler(with_mean=False)
bigrams_train = standardizer.fit_transform(count_vect.fit_transform(x_train))
bigrams_test = standardizer.transform(count_vect.transform(x_test))
print("some feature names ", count_vect.get_feature_names()[489:499])
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])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtyp
e int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtyp
e int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtyp
e int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)

some feature names  ['actually bought', 'actually break', 'actually brought', 'actually buy', 'actually buying', 'actua
lly called', 'actually came', 'actually cheaper', 'actually chew', 'actually coffee']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer for Train set  (254919, 141856)
the number of unique words including both unigrams and bigrams in Train set  141856
==================================================
the shape of out text BOW vectorizer for Test set  (109252, 141856)
the number of unique words including both unigrams and bigrams Test set  141856
```

[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)](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)](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)

In [29]:
```
tfidf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
standardizer = StandardScaler(with_mean=False)
tfidf_bigrams_train = standardizer.fit_transform(tfidf_vect.fit_transform(x_train))
tfidf_bigrams_test = standardizer.transform(tfidf_vect.transform(x_test))
print("some feature names ", tfidf_vect.get_feature_names()[5000:5010])
print('='*50)

print("the type of count vectorizer ",type(tfidf_bigrams_train))
print("the shape of out text Tfidf 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 Tfidf 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])
```

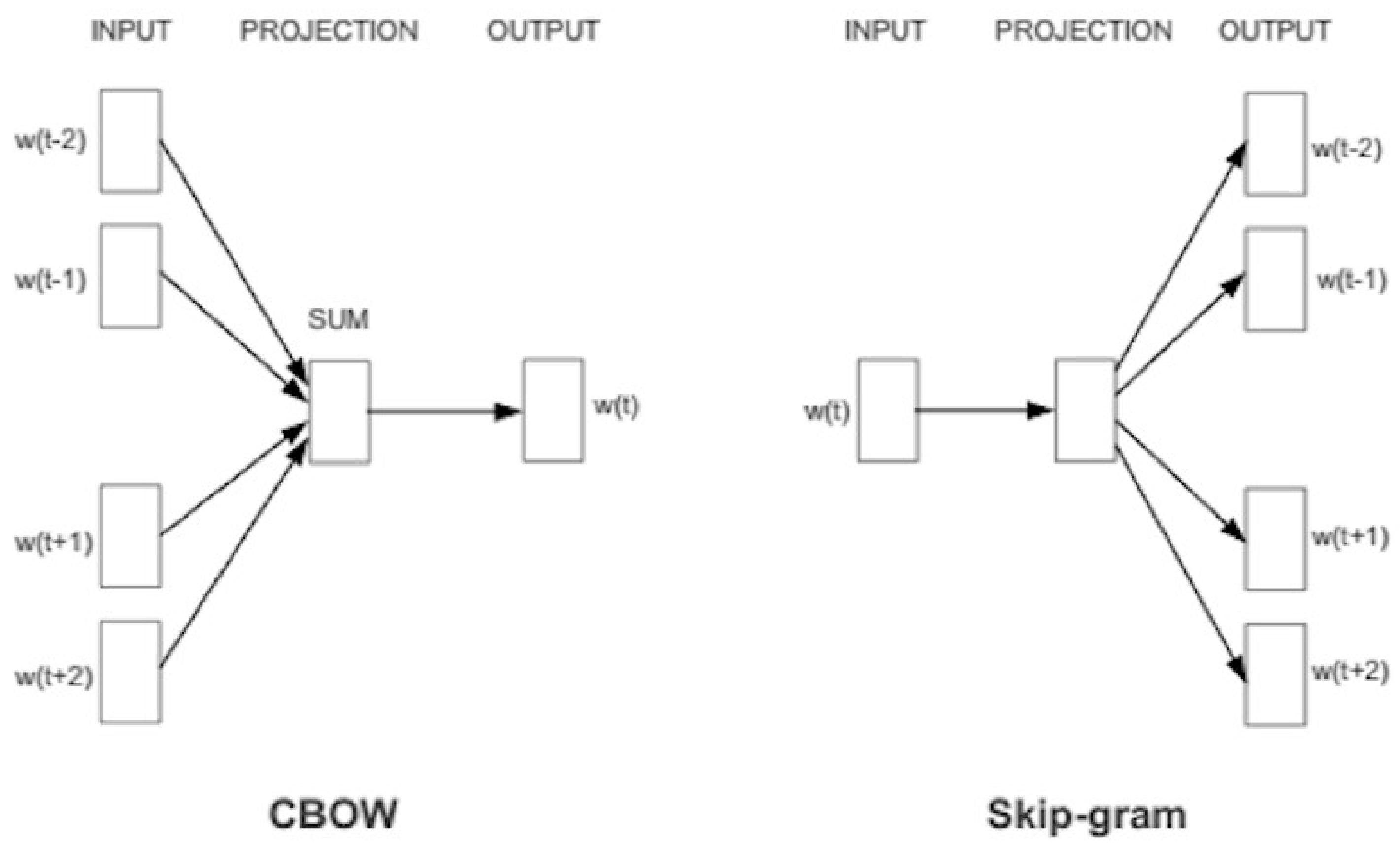
```
some feature names  ['anymore good', 'anymore great', 'anymore guess', 'anymore happy', 'anymore however', 'anymore lea
st', 'anymore like', 'anymore local', 'anymore love', 'anymore make']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text Tfidf vectorizer for Train set  (254919, 141856)
the number of unique words including both unigrams and bigrams in Train set  141856
==================================================
the shape of out text Tfidf vectorizer for Test set  (109252, 141856)
the number of unique words including both unigrams and bigrams Test set  141856
```

[4.4] Word2Vec

In [30]:
```
# 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)](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/)](https://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/)



In [31]:
```python
# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
w2v_model=Word2Vec(sentancesListTrain,min_count=5,size=100, workers=-1)
print(w2v_model.wv.most_similar('tasty'))
print('='*50)
```

```
[('take', 0.4232155978679657), ('cubicle', 0.4066615700721741), ('naples', 0.40344777770328522), ('essentially', 0.38616129755973816), ('flashback', 0.3745872974395752), ('iceland', 0.3685303330421448), ('rather', 0.3595985770225525), ('rascal', 0.35621461272239685), ('ginsing', 0.35514384508132935), ('hip', 0.3500310480594635)]
==================================================
[('yuzu', 0.39672258496284485), ('bentley', 0.38423046469688416), ('recommendations', 0.36636102199554443), ('someplace', 0.36509525775909424), ('poweder', 0.3636173605918884), ('housewarming', 0.35937488079071045), ('japan', 0.34702256321907043), ('web', 0.3456791639328003), ('saucy', 0.34508228302001953), ('andthis', 0.34126847982406616)]
```

In [32]:
```python
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', 'surprising']
```

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

[4.4.1.1] Avg W2v

In [33]:
```python
# 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(100) # 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)
```

```
100%|██████████████████████████████| 254919/254919 [16:32<00:00, 256.92it/s]
```

In [34]:
```python
standardizer = StandardScaler()
trainWord2Vectors = standardizer.fit_transform(trainWord2Vectors)
print(len(trainWord2Vectors))
```

```
254919
100
```

In [35]:
```python
testWord2Vectors = []; # the avg-w2v for each test sentence/review is stored in this list
for eachSentance in tqdm(sentancesListTest):
    sentanceVector = np.zeros(100)
    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)

testWord2Vectors = standardizer.transform(testWord2Vectors)
print(len(testWord2Vectors))
```

```
100%|██████████████████████████████| 109252/109252 [08:11<00:00, 222.23it/s]

109252
100
```

[4.4.1.2] TFIDF weighted W2v

In [36]:
```python
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 [37]:
```
# 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(100) # 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
    trainTfidfWord2Vectors.append(sentanceVector)
standardizer = StandardScaler()
trainTfidfWord2Vectors = standardizer.fit_transform(trainTfidfWord2Vectors)
print(len(trainTfidfWord2Vectors))
```

```
100%|██████████| 254919/254919 [33:56<00:00, 125.17it/s]

254919
100
```

In [38]:
```
testTfidfWord2Vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
for eachSentance in tqdm(sentancesListTest):
    sentanceVector = np.zeros(100) # 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)

testTfidfWord2Vectors = standardizer.transform(testTfidfWord2Vectors)
print(len(testTfidfWord2Vectors))
```

```
100%|██████████| 109252/109252 [15:57<00:00, 114.09it/s]

109252
100
```

Support Vector Machines

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are:

1. Effective in high dimensional spaces.
2. Still effective in cases where number of dimensions is greater than the number of samples.
3. Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
4. Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

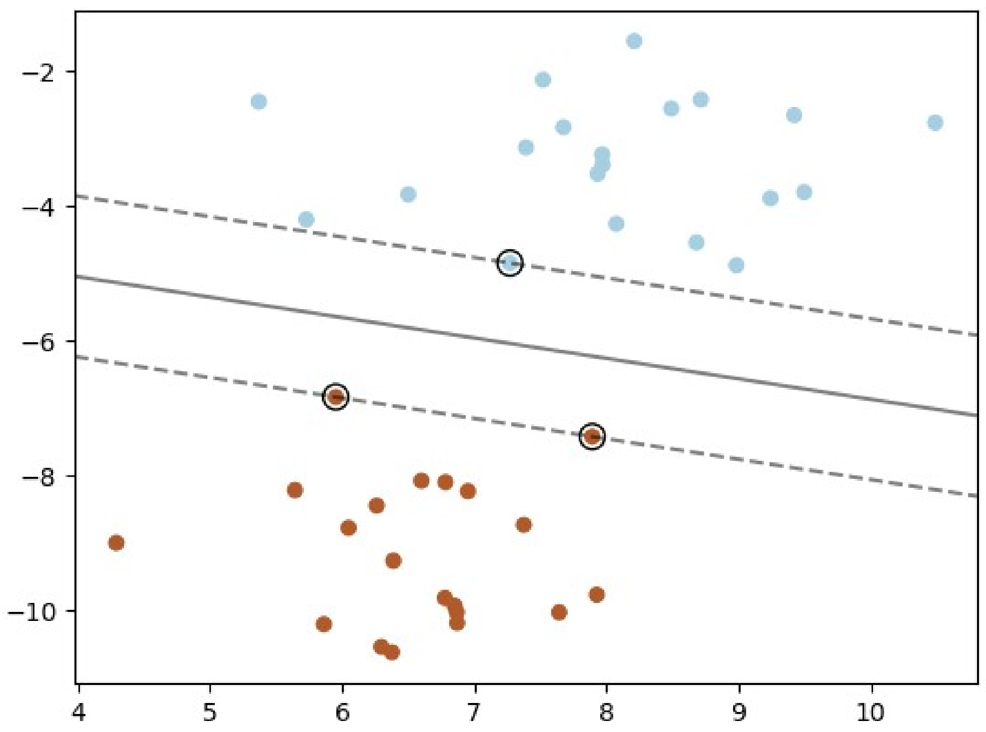
The disadvantages of support vector machines include:

1. If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
2. SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation

The kernel function:

1. linear :**(x,x')**
2. rbf :**exp(**$\gamma||x - x'||^2$**)**

A support vector machine constructs a **hyper-plane or set of hyper-planes in a high or infinite dimensional space, which can be used for classification**. Intuitively, a **good separation** is achieved **by the hyper-plane that has the largest distance to the nearest training data points** of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.



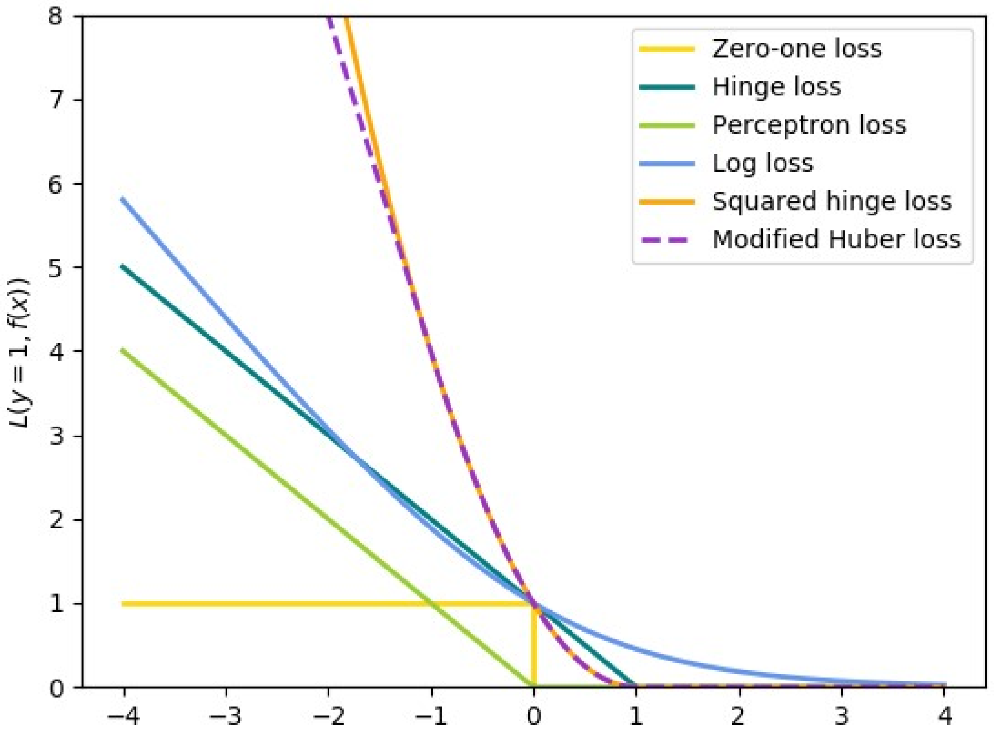
Given a set of training examples $(x_1, y_1), \ldots (x_n, y_n)$ where $x_i \in R^m$ and $y_i \in (-1, 1)$ and, our goal is to learn a linear scoring function $f(x) = W^T x + b$ with model parameters $W \in R^m$ and intercept $b \in R$. In order to make predictions, we simply look at the sign of $f(x)$. A common choice to find the model parameters is by minimizing the regularized training error given by:

$$E(R, b) = \left(\frac{1}{n}\right) \sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(W)$$

where L is a loss function that measures model (mis)fit and R is a regularization term (aka penalty) that penalizes model complexity $\alpha > 0$ is a non-negative hyperparameter.

Different choices for L entail different classifiers such as

1. Hinge: (soft-margin) Support Vector Machines.
2. Log: Logistic Regression.



In [39]:
```
#source -
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
def gridSearcher(model,parameters,inputs,outputs):
    clf = GridSearchCV(model,
                       param_grid = parameters,
                       return_train_score = True,
                       scoring='roc_auc',
                       n_jobs=-1,
                       cv=5)
    clf.fit(inputs,outputs)
```

In [40]:
```
#https://stackoverflow.com/a/42712772/12901493
import seaborn as sns
def plotAUCvsHyperParam(model):
    plt.figure(figsize=(10,10))
    f,(ax1,ax2) = plt.subplots(1,2,figsize=(20,9))

    testScore = model.cv_results_["mean_test_score"]
    testScore = testScore.reshape(len(model.param_grid["alpha"]),len(model.param_grid["penalty"]))
    g1 = sns.heatmap(testScore,
                    annot = True,
                    fmt=".4f",
                    ax = ax1,
                    cmap = sns.color_palette("Paired"),
                    xticklabels=model.param_grid["penalty"],
                    yticklabels=np.round(model.param_grid["alpha"], 3))
    g1.set_xlabel("penalty")
    g1.set_ylabel("alpha")
    title = "Best Cross Validation Score = "+\
            str(model.best_score_)+"\n"\
            " at "+\
            "alpha "+str(model.best_params_["alpha"])+\
            " , "+\
            "penalty "+str(model.best_params_["penalty"])
    ax1.title.set_text(title)
    ax1.title.set_fontsize(15)

    trainScore = model.cv_results_["mean_train_score"]
    trainScore = trainScore.reshape(len(model.param_grid["alpha"]),len(model.param_grid["penalty"]))
    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["penalty"],
                    yticklabels=np.round(model.param_grid["alpha"], 3))
    g2.set_xlabel("penalty")
    g2.set_ylabel("alpha")
    title = "Best Train Score = "+\
            str(trainScore.max())+"\n"\
            " at "+\
            "alpha "+str(model.param_grid["alpha"][indices[0]])+\
            " , "+\
            "penalty "+str(model.param_grid["penalty"][indices[1]])
    ax2.title.set_text(title)
```

In [41]:
```
#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.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
```

In [42]:
```
#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([[tn,fp],[fn,tp]],yticklabels=["Actual 0","Actual 1"],\
```

Applying Linear Support Vector Machines

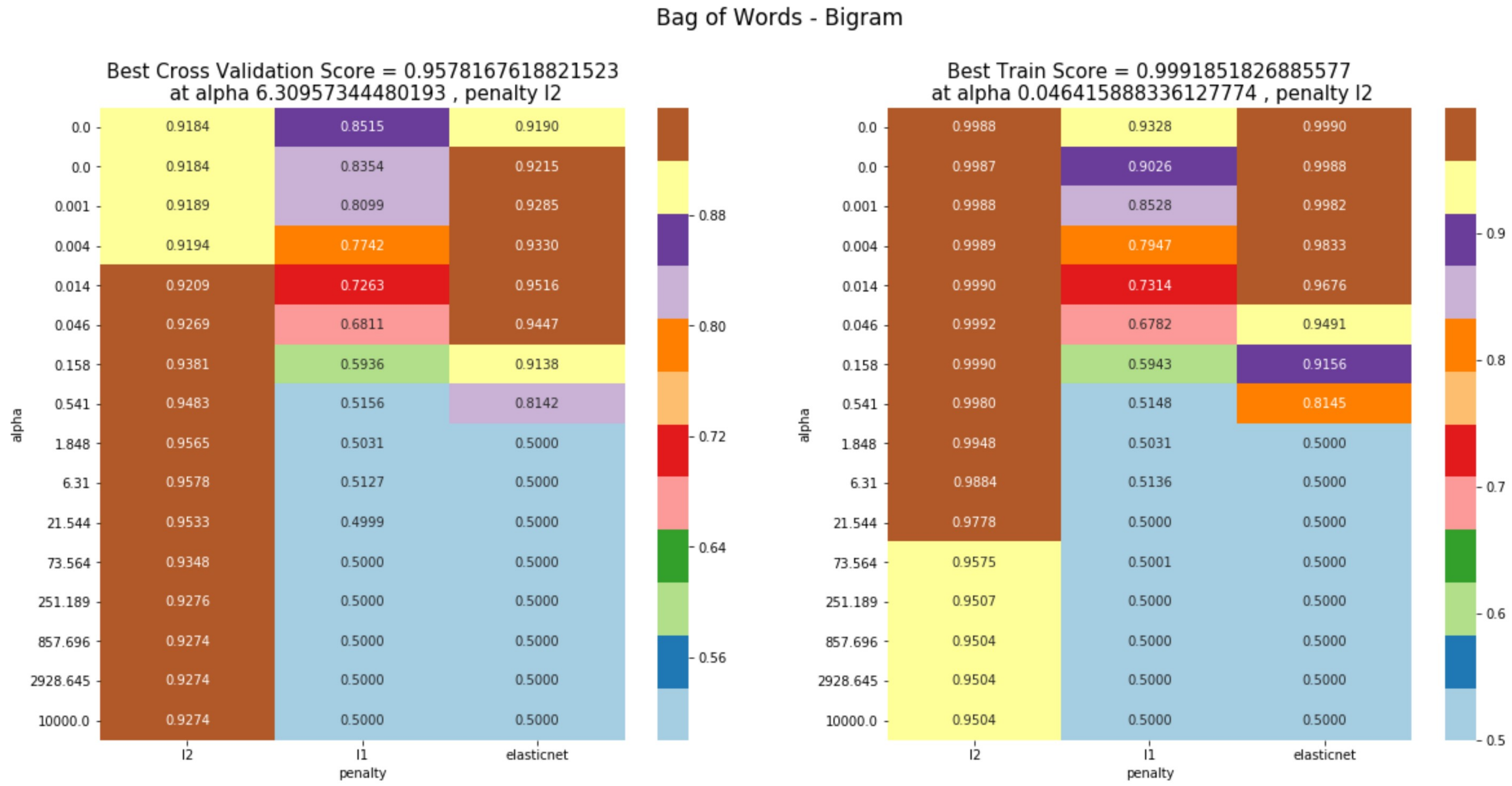
[5.1] Applying Linear SVM on BOW

In [43]:
```
from sklearn.linear_model import SGDClassifier
#https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
linear_bigram_model = gridSearcher(SGDClassifier(class_weight="balanced"),
                                   {"alpha":np.logspace(-4, 4, 16),"penalty":['l2', 'l1', 'elasticnet']},
                                   bigrams_train,
```

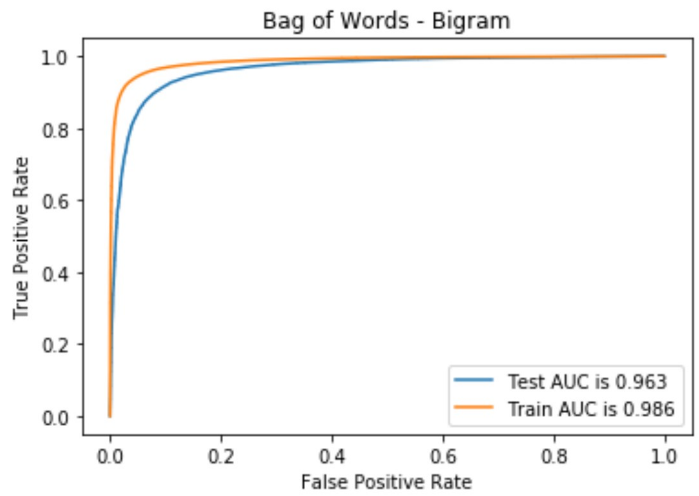
In [44]:
```
plotAUCvsHyperParam(linear_bigram_model)
```

Out[44]:
```
Text(0.49, 1, 'Bag of Words - Bigram')
```

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



In [45]:
```
from sklearn.calibration import CalibratedClassifierCV
#https://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
clf_linear_bigram = CalibratedClassifierCV(base_estimator = linear_bigram_model.best_estimator_, cv="prefit")
clf_linear_bigram.fit(bigrams_train,y_train)
plt.title("Bag of Words - Bigram")
```



In [46]:
```
plt.title("Bag of Words - Bigram")
```

[5.1.1] Top 20 important features from Linear Bigram BOW SVM

In [47]:
```
print("Top 10 Positive Features")
print(pd.DataFrame(data = linear_bigram_model.best_estimator_.coef_[0],index=count_vect.get_feature_names()).sort_values(b
print("\n\nTop 10 Negative Features")
print(pd.DataFrame(data = linear_bigram_model.best_estimator_.coef_[0],index=count_vect.get_feature_names()).sort_values(b
```

```
Top 10 Positive Features
                  0
great      0.025122
love       0.017204
best       0.017076
delicious  0.014575
good       0.012716
loves      0.012574
perfect    0.012344
highly     0.011228
excellent  0.011089
wonderful  0.010888


Top 10 Negative Features
                      0
terrible      -0.011781
waste money   -0.011826
not good      -0.012146
would not     -0.012434
disappointing -0.013386
worst         -0.013623
not           -0.014065
not worth     -0.014503
not recommend -0.014736
disappointed  -0.017543
```

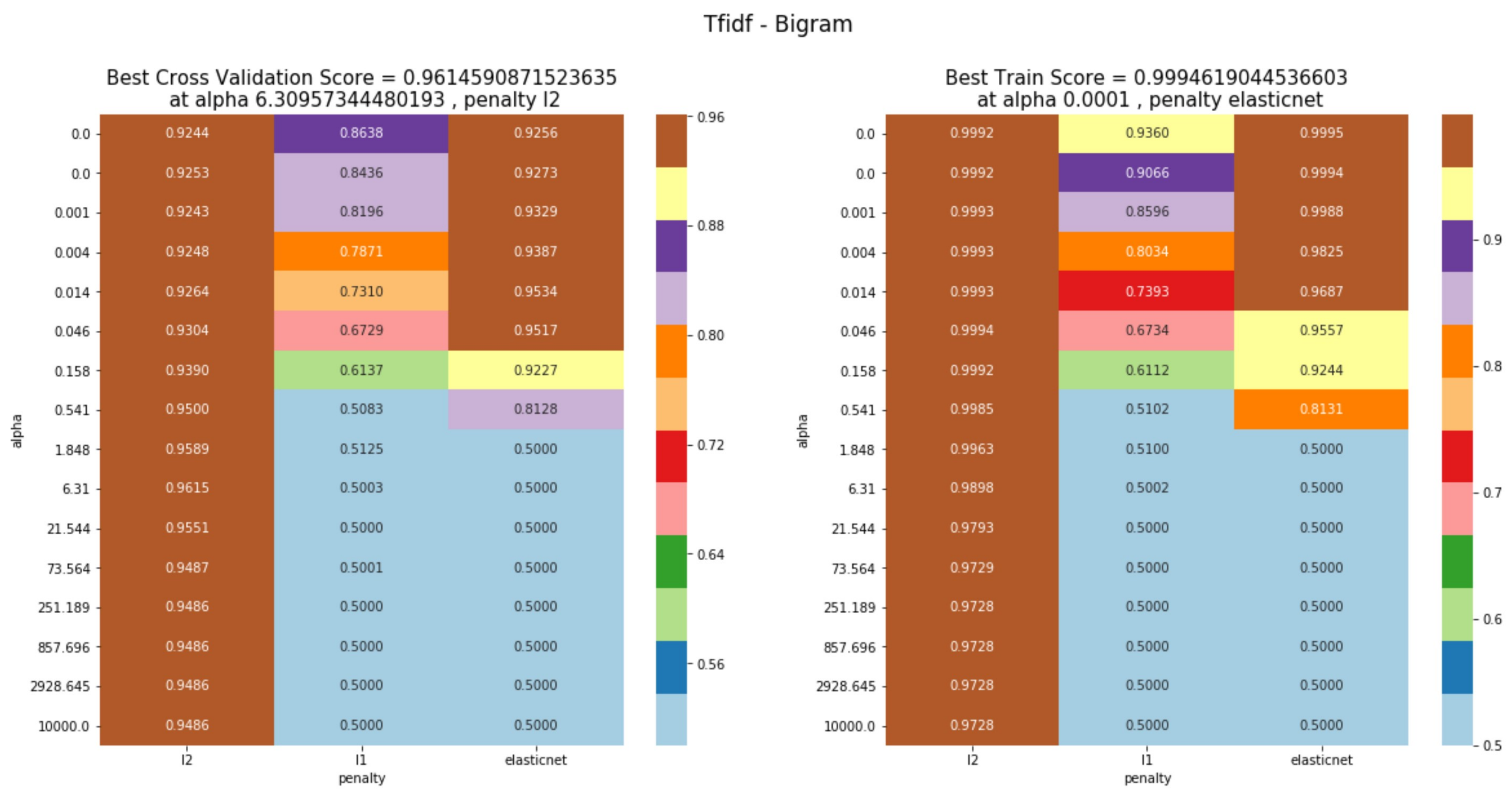
[5.2] Applying Linear SVM on TFIDF

In [48]:
```
tfidf_linear_bigram_model = gridSearcher(SGDClassifier(class_weight="balanced"),
                                         {"alpha":np.logspace(-4, 4, 16),"penalty":['l2', 'l1', 'elasticnet']},
                                         tfidf_bigrams_train,
```

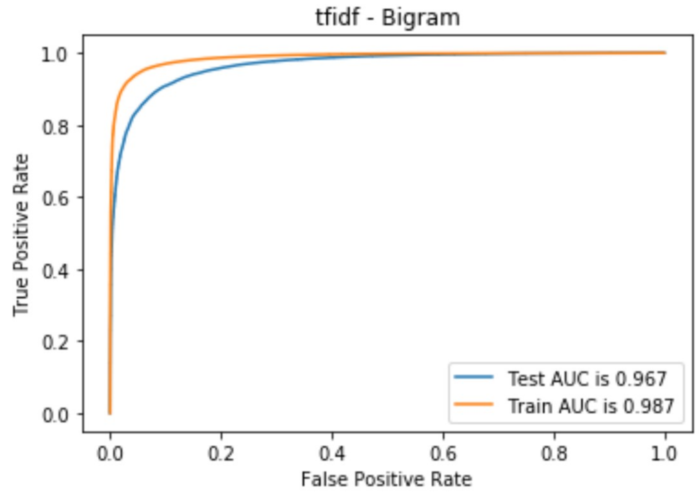
In [49]:
```
plotAUCvsHyperParam(tfidf_linear_bigram_model)
```

Out[49]:
```
Text(0.49, 1, 'Tfidf - Bigram')
```

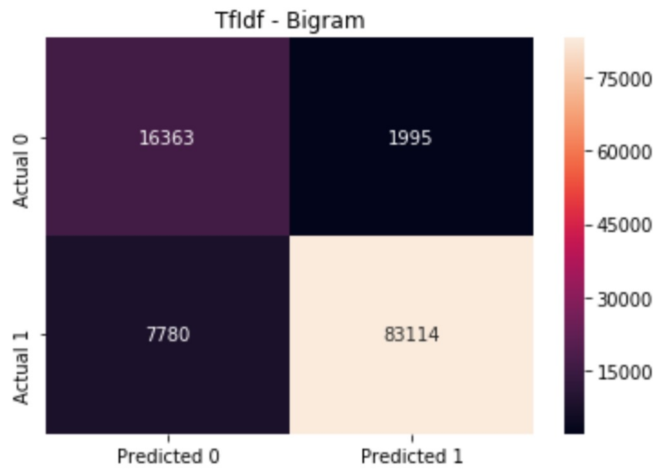
```
<Figure size 720x720 with 0 Axes>
```



In [50]:
```
clf_tfidf_linear_bigram = CalibratedClassifierCV(base_estimator = tfidf_linear_bigram_model.best_estimator_, cv="prefit")
clf_tfidf_linear_bigram.fit(tfidf_bigrams_train,y_train)
plt.title("tfidf - Bigram")
rocCurve(clf_tfidf_linear_bigram,tfidf_bigrams_train,y_train,tfidf_bigrams_test,y_test)
```



In [51]:
```
plt.title("TfIdf - Bigram")
```



[5.2.1] Top 20 important features from linear Tfidf Bigram SVM

In [52]:
```
print("Top 10 Positive Features")
print(pd.DataFrame(data = tfidf_linear_bigram_model.best_estimator_.coef_[0],index=tfidf_vect.get_feature_names()).sort_va
print("\n\nTop 10 Negative Features")
print(pd.DataFrame(data = tfidf_linear_bigram_model.best_estimator_.coef_[0],index=tfidf_vect.get_feature_names()).sort_va
```

```
Top 10 Positive Features
                  0
great      0.026038
love       0.018967
best       0.018875
good       0.015838
delicious  0.015102
perfect    0.012952
loves      0.012836
favorite   0.011554
excellent  0.011425
wonderful  0.011405


Top 10 Negative Features
                      0
awful         -0.011452
terrible      -0.011953
disappointing -0.012530
would not     -0.012649
not good      -0.012996
not worth     -0.013412
not recommend -0.013637
worst         -0.013886
not buy       -0.015546
disappointed  -0.016287
```

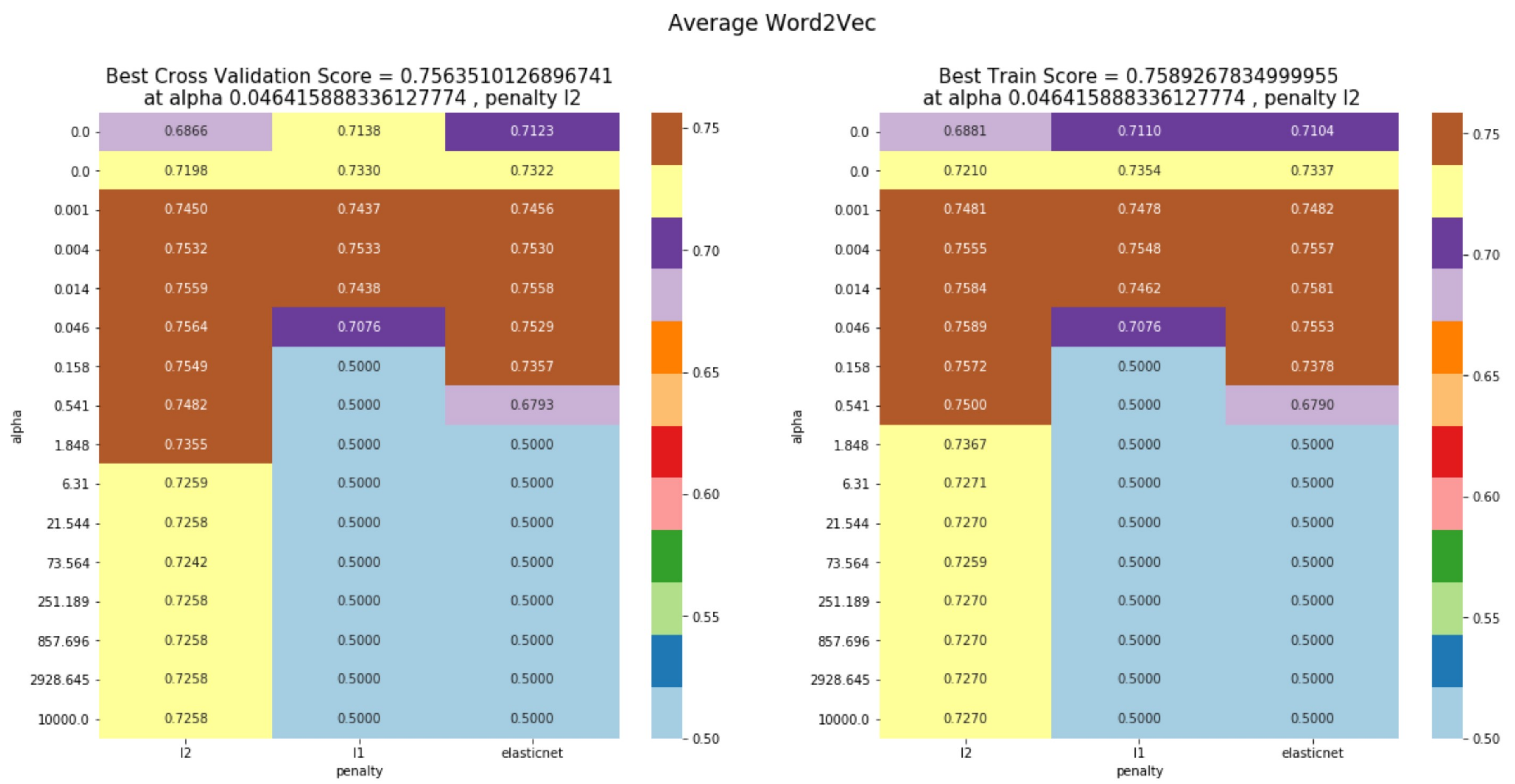
[5.3] Applying Linear SVM on Average Word2VEC

In [53]:
```
linear_avgW2v_model = gridSearcher(SGDClassifier(class_weight="balanced"),
                                   {"alpha":np.logspace(-4, 4, 16),"penalty":['l2', 'l1', 'elasticnet']},
                                   trainWord2Vectors,
```

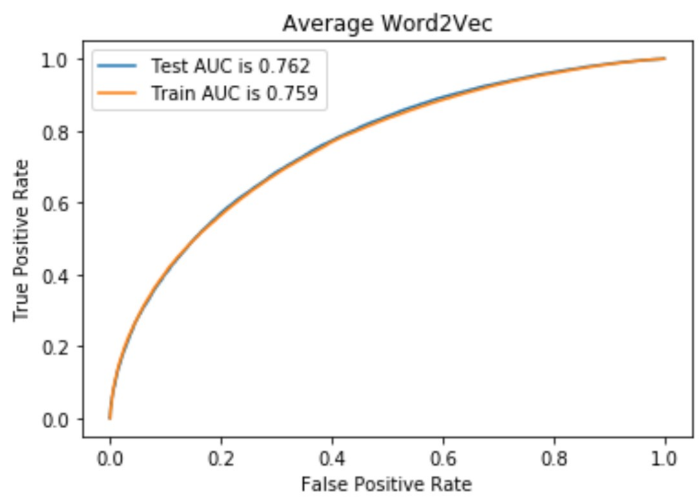
In [54]:
```
plotAUCvsHyperParam(linear_avgW2v_model)
```

Out[54]:
```
Text(0.49, 1, 'Average Word2Vec')
```

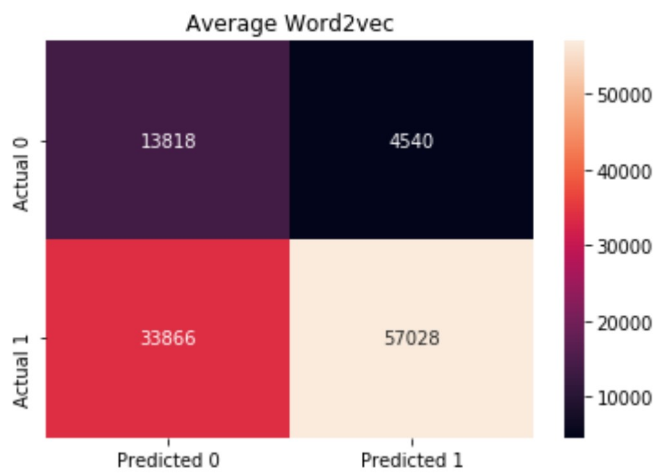
```
<Figure size 720x720 with 0 Axes>
```



In [55]:
```
clf_linear_avgW2v = CalibratedClassifierCV(base_estimator = linear_avgW2v_model.best_estimator_, cv="prefit")
clf_linear_avgW2v.fit(trainWord2Vectors,y_train)
plt.title("Average Word2Vec")
```



In [56]:
```
plt.title("Average Word2vec")
```

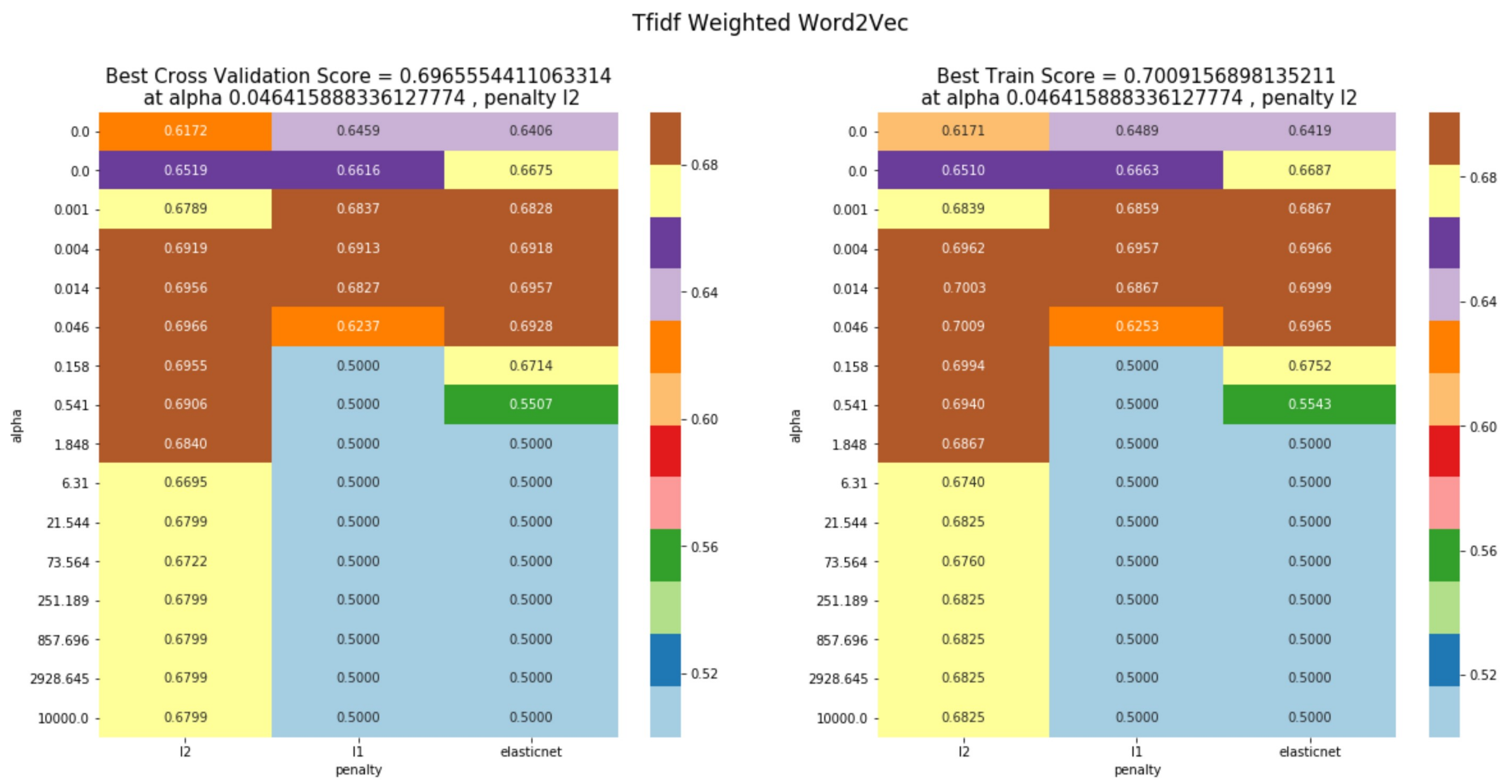


[5.4] Applying Linear SVM on TFIDF Weighted W2V

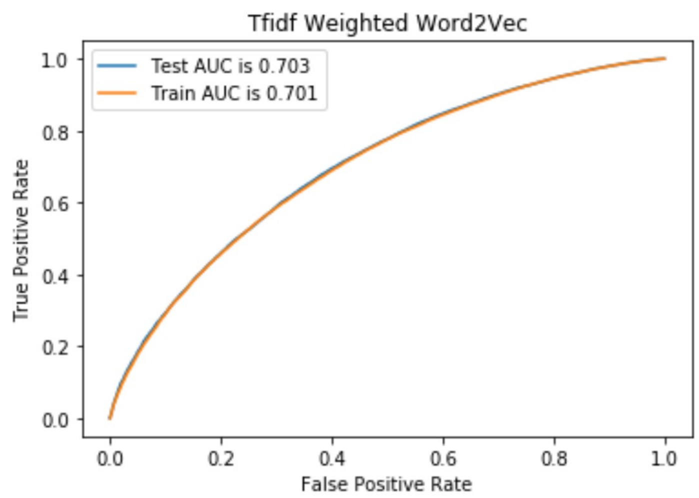
In [57]:
```
linear_tfidfW2v_model = gridSearcher(SGDClassifier(class_weight="balanced"),
                                     {"alpha":np.logspace(-4, 4, 16),"penalty":['l2', 'l1', 'elasticnet']},
                                     trainTfidfWord2Vectors,
```

In [58]:
```
plotAUCvsHyperParam(linear_tfidfW2v_model)
```

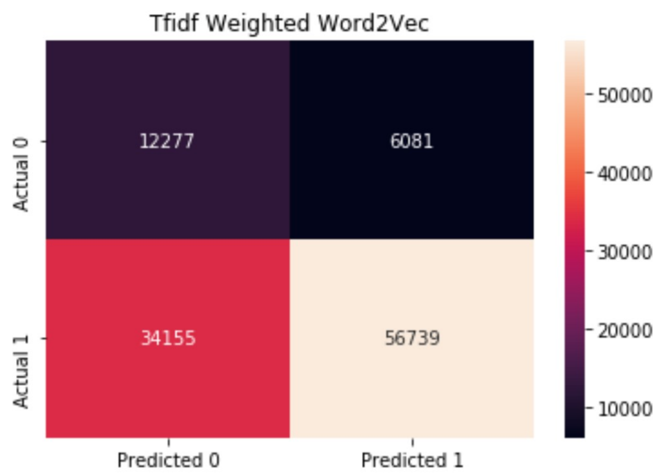
Out[58]:
```
Text(0.49, 1, 'Tfidf Weighted Word2Vec')

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



In [59]:
```
clf_linear_tfidfW2v = CalibratedClassifierCV(base_estimator = linear_tfidfW2v_model.best_estimator_, cv="prefit")
clf_linear_tfidfW2v.fit(trainTfidfWord2Vectors,y_train)
plt.title("Tfidf Weighted Word2Vec")
```



In [60]:
```
plt.title("Tfidf Weighted Word2Vec")
```



In [61]:

```
from prettytable import PrettyTable
table = PrettyTable()
table.field_names = ["Vectoriser","parameters","Train AUC Score","Test AUC score"]
table.add_row(["Linear SVM(BOW)",
               linear_bigram_model.best_params_,
               np.round(linear_bigram_model.score(bigrams_train,y_train),5),
               np.round(linear_bigram_model.score(bigrams_test,y_test),5)])
table.add_row(["Linear SVM(TFIDF)",
               tfidf_linear_bigram_model.best_params_,
               np.round(tfidf_linear_bigram_model.score(tfidf_bigrams_train,y_train),5),
               np.round(tfidf_linear_bigram_model.score(tfidf_bigrams_test,y_test),5)])
table.add_row(["Linear SVM(Avg W2V)",
               linear_avgW2v_model.best_params_,
               np.round(linear_avgW2v_model.score(trainWord2Vectors,y_train),5),
               np.round(linear_avgW2v_model.score(testWord2Vectors,y_test),5)])
table.add_row(["Linear SVM(TFIDF W2V)",
               linear_tfidfW2v_model.best_params_,
               np.round(linear_tfidfW2v_model.score(trainTfidfWord2Vectors,y_train),5),
```

In [62]:

```
from sklearn.preprocessing import StandardScaler
count_vect = CountVectorizer(ngram_range=(1,2), min_df=20, max_features=500)
standardizer = StandardScaler(with_mean=False)
bigrams_train = standardizer.fit_transform(count_vect.fit_transform(x_train))
bigrams_test = standardizer.transform(count_vect.transform(x_test))
print("some feature names ", count_vect.get_feature_names()[489:499])
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])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtyp
e int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtyp
e int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtyp
e int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)

some feature names  ['work', 'works', 'worth', 'would', 'would not', 'wrong', 'year', 'years', 'yes', 'yet']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer for Train set  (254919, 500)
the number of unique words including both unigrams and bigrams in Train set  500
==================================================
the shape of out text BOW vectorizer for Test set  (109252, 500)
the number of unique words including both unigrams and bigrams Test set  500
```

In [63]:

```
tfidf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=20, max_features=500)
standardizer = StandardScaler(with_mean=False)
tfidf_bigrams_train = standardizer.fit_transform(tfidf_vect.fit_transform(x_train))
tfidf_bigrams_test = standardizer.transform(tfidf_vect.transform(x_test))
print("some feature names ", tfidf_vect.get_feature_names()[300:310])
print('='*50)

print("the type of count vectorizer ",type(tfidf_bigrams_train))
print("the shape of out text Tfidf 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 Tfidf 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  ['not sure', 'not taste', 'nothing', 'nuts', 'oatmeal', 'often', 'oil', 'old', 'olive', 'one']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text Tfidf vectorizer for Train set  (254919, 500)
the number of unique words including both unigrams and bigrams in Train set  500
==================================================
the shape of out text Tfidf vectorizer for Test set  (109252, 500)
the number of unique words including both unigrams and bigrams Test set  500
```

In [64]:
```
def plotAUCvsHyperParam(model):
    plt.figure(figsize=(10,10))
    f,(ax1,ax2) = plt.subplots(1,2,figsize=(20,9))

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

    trainScore = model.cv_results_["mean_train_score"]
    trainScore = trainScore.reshape(len(model.param_grid["C"]),len(model.param_grid["gamma"]))
    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["gamma"],
                     yticklabels=np.round(model.param_grid["C"], 4))
    g2.set_xlabel("gamma")
    g2.set_ylabel("C")
    title = "Best Train Score = "+\
            str(trainScore.max())+"\n"\
            " at "+\
            "C "+str(model.param_grid["C"][indices[0]])+\
            " , "+\
            "gamma "+str(model.param_grid["gamma"][indices[1]])
    ax2.title.set_text(title)
```

[6] Applying "RBF" Kernel SVM

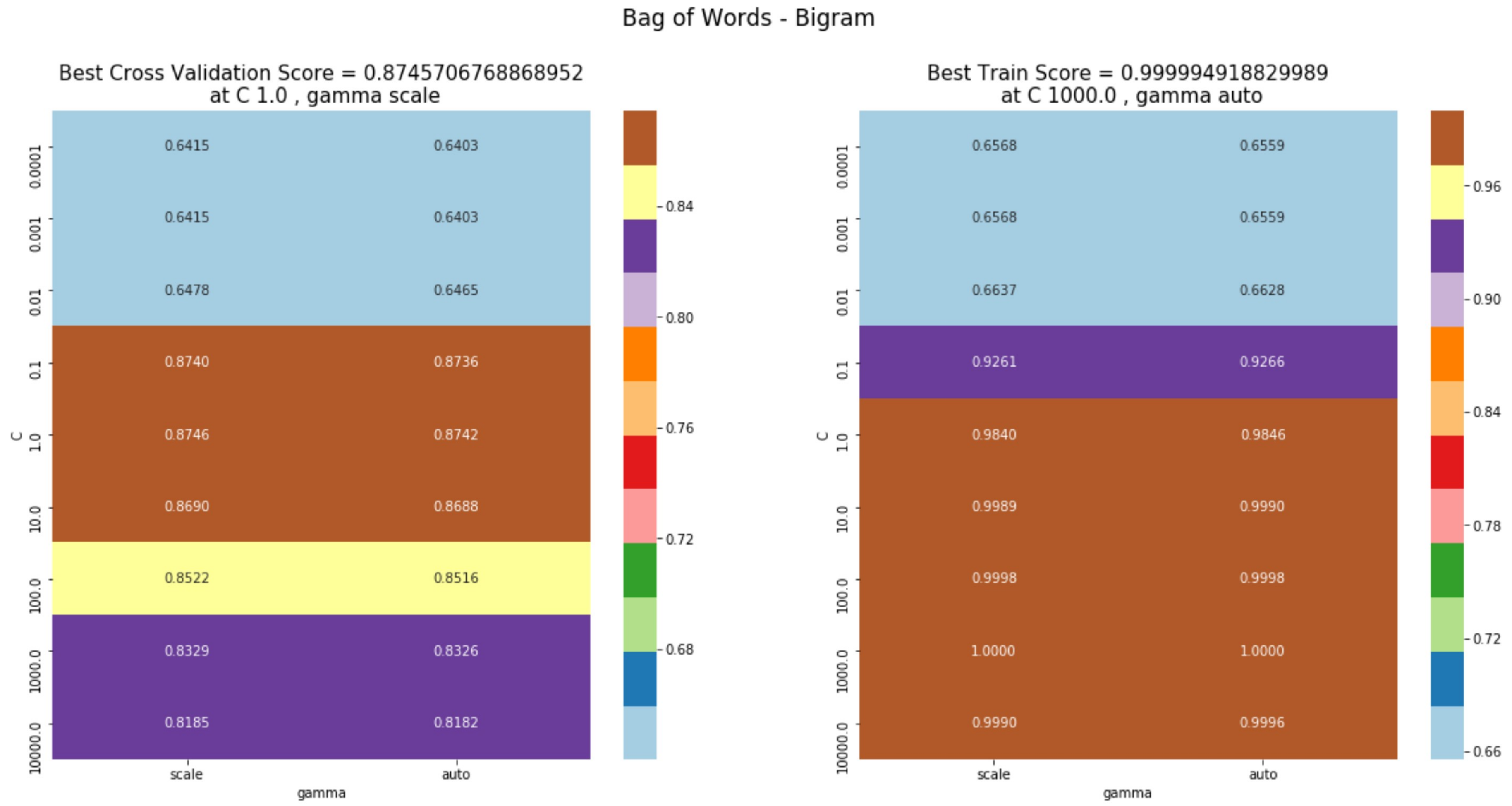
[6.1] Applying RBF SVM on Bigram BOW

In [65]:
```
from sklearn.svm import SVC
#https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC
rbf_bigram_model = gridSearcher(SVC(class_weight="balanced"),
                                {"C":np.logspace(-4, 4, 9),"gamma":['scale','auto']},
                                bigrams_train[:20000],
```

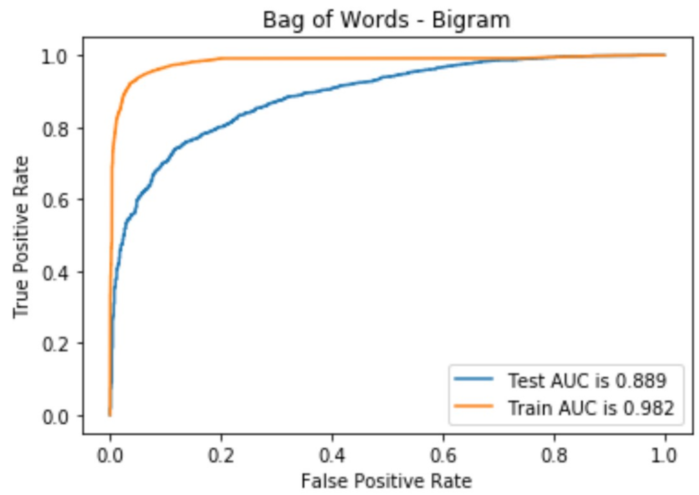
In [66]:
```
plotAUCvsHyperParam(rbf_bigram_model)
```

Out[66]: Text(0.49, 1, 'Bag of Words - Bigram')

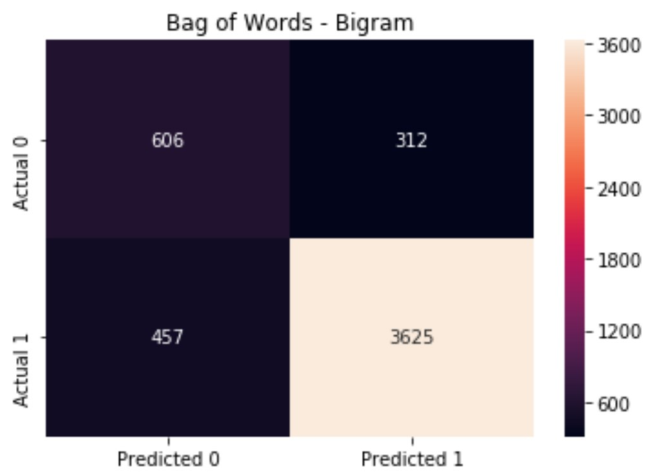
<Figure size 720x720 with 0 Axes>



In [67]:
```
clf_rbf_bigram = CalibratedClassifierCV(base_estimator = rbf_bigram_model.best_estimator_, cv="prefit")
clf_rbf_bigram.fit(bigrams_train[:20000],y_train[:20000])
plt.title("Bag of Words - Bigram")
```



In [68]:
```
plt.title("Bag of Words - Bigram")
```



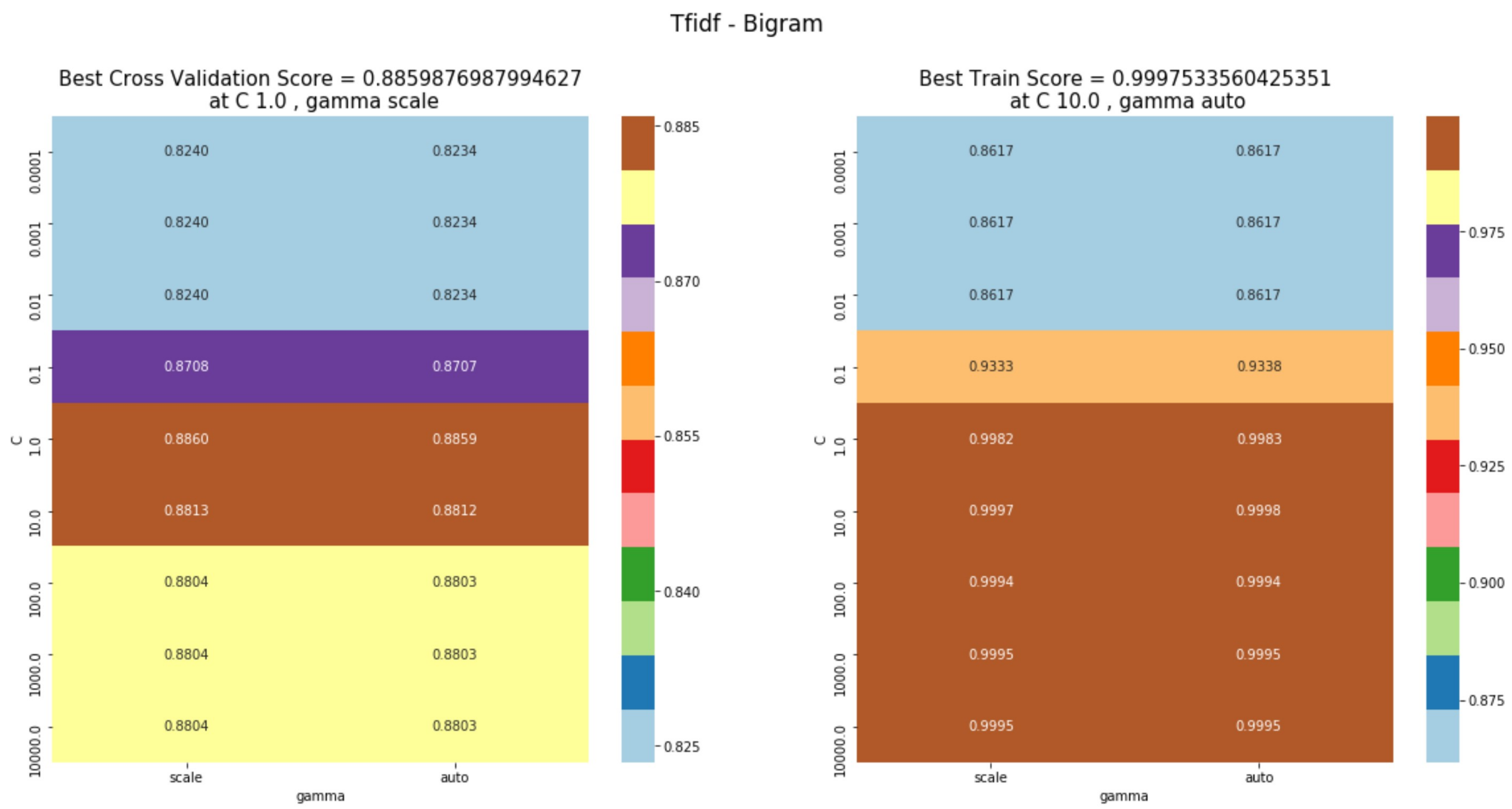
[6.2] Applying RBF SVM on Bigram Tfidf

In [69]:
```
tfidf_rbf_bigram_model = gridSearcher(SVC(class_weight="balanced"),
                                      {"C":np.logspace(-4, 4, 9),"gamma":['scale','auto']},
                                      tfidf_bigrams_train[:20000],
```

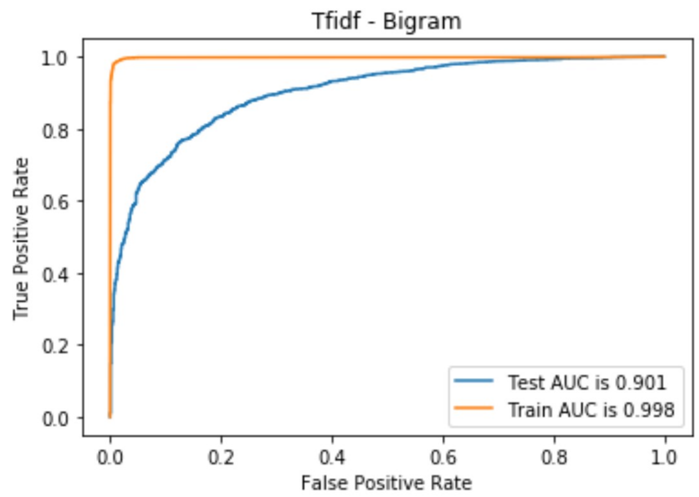
In [70]:
```
plotAUCvsHyperParam(tfidf_rbf_bigram_model)
```

Out[70]:
```
Text(0.49, 1, 'Tfidf - Bigram')
```

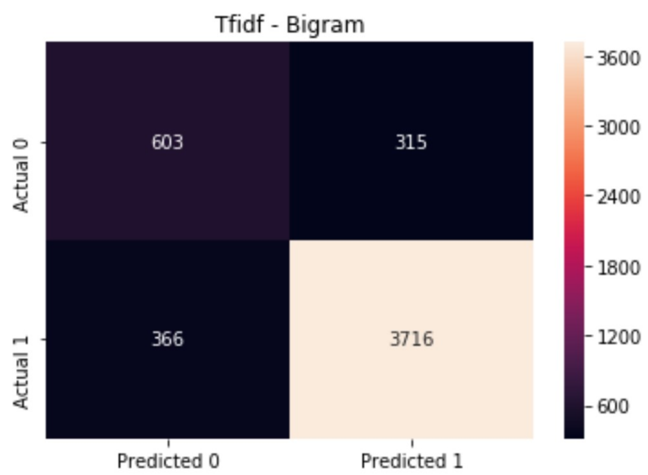
```
<Figure size 720x720 with 0 Axes>
```



In [71]:
```
clf_tfidf_rbf_bigram = CalibratedClassifierCV(base_estimator = tfidf_rbf_bigram_model.best_estimator_, cv="prefit")
clf_tfidf_rbf_bigram.fit(tfidf_bigrams_train[:20000],y_train[:20000])
plt.title("Tfidf - Bigram")
```



In [72]:
```
plt.title("Tfidf - Bigram")
```

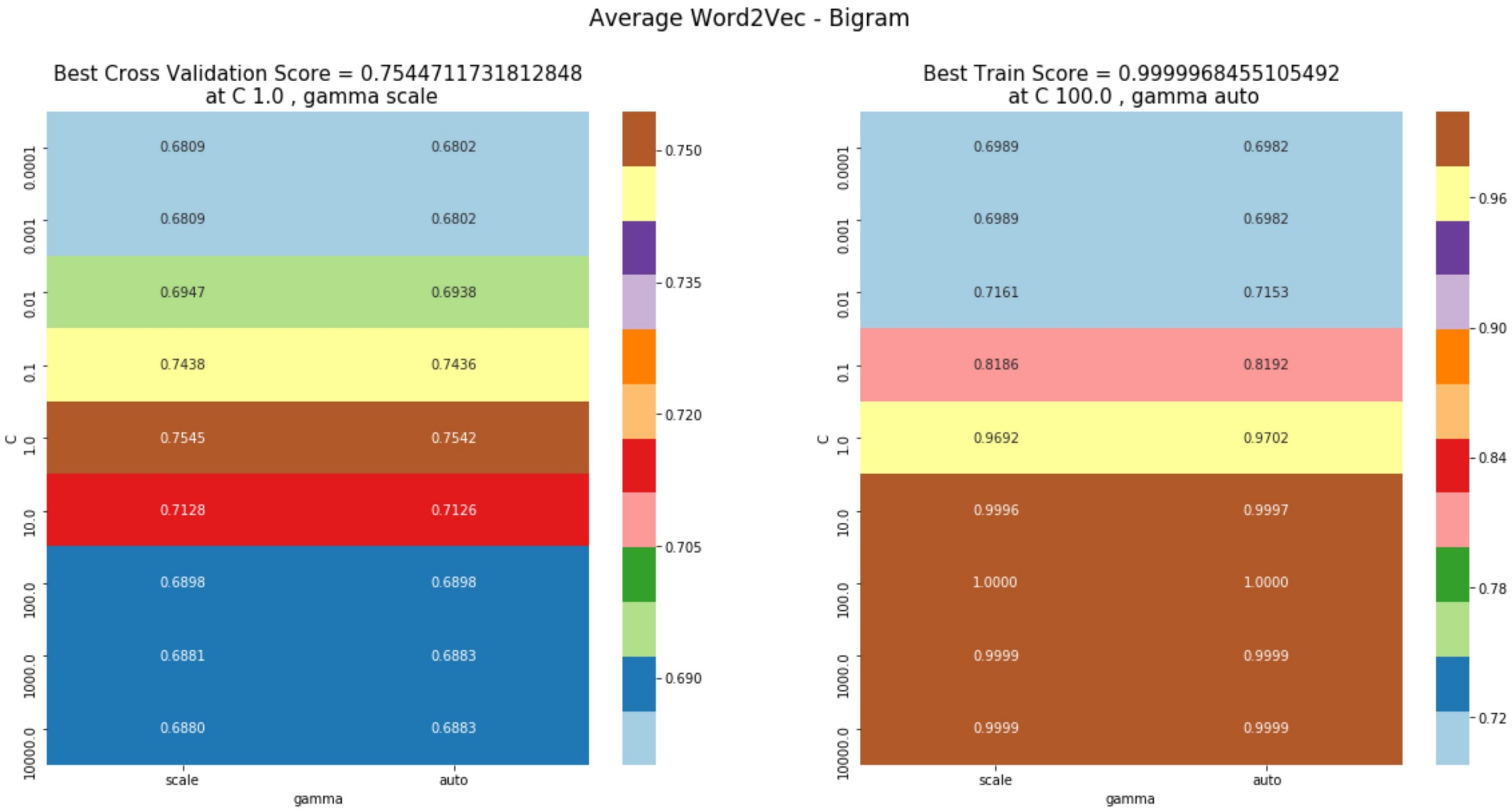


[6.3] Applying RBF SVM on Average Word2VEC

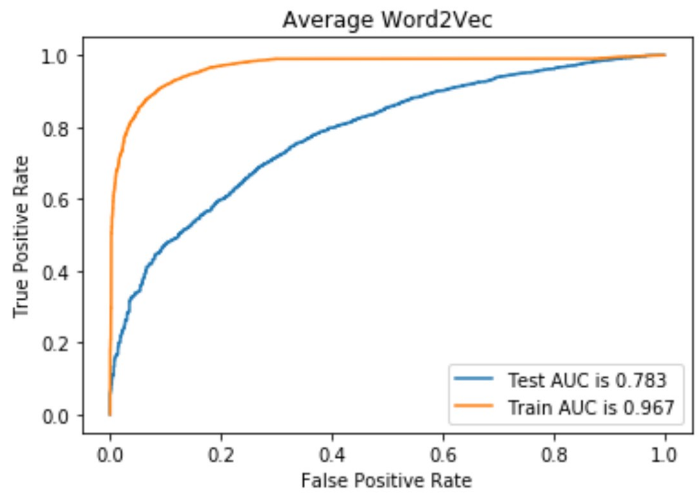
In [73]:
```
rbf_avgW2v_model = gridSearcher(SVC(class_weight="balanced"),
                                 {"C":np.logspace(-4, 4, 9),"gamma":['scale','auto']},
                                 trainWord2Vectors[:20000],
```

In [74]:
```
plotAUCvsHyperParam(rbf_avgW2v_model)
```

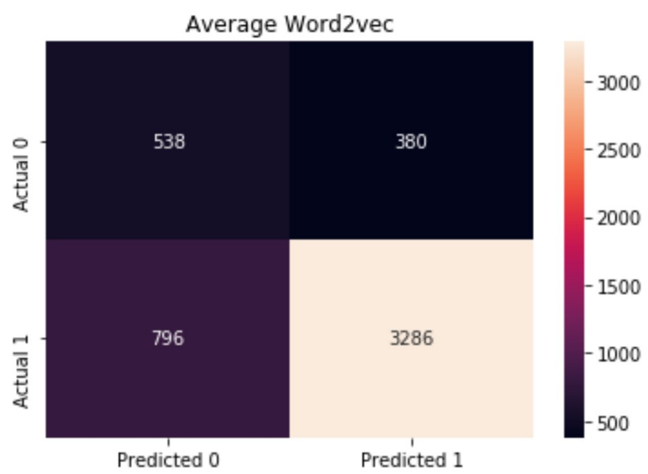
Out[74]:
```
Text(0.49, 1, 'Average Word2Vec - Bigram')

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



In [75]:
```
clf_rbf_avgW2v = CalibratedClassifierCV(base_estimator = rbf_avgW2v_model.best_estimator_, cv="prefit")
clf_rbf_avgW2v.fit(trainWord2Vectors[:20000],y_train[:20000])
plt.title("Average Word2Vec")
```



In [76]:
```
plt.title("Average Word2vec")
```



[6.4] Applying RBF SVM on TFIDF Weighted W2V

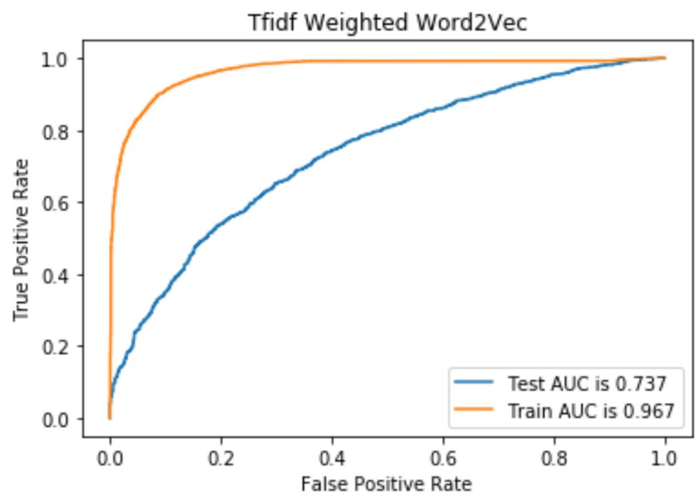
In [77]:
```
rbf_tfidfW2v_model = gridSearcher(SVC(class_weight="balanced"),
                                  {"C":np.logspace(-4, 4, 9),"gamma":['scale','auto']},
                                  trainTfidfWord2Vectors[:20000],
```

In [78]:
```
plotAUCvsHyperParam(rbf_tfidfW2v_model)
```

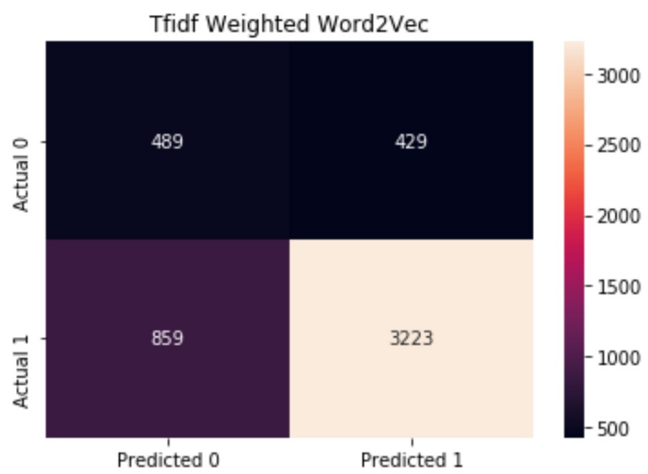
Out[78]:
```
Text(0.49, 1, 'Tfidf Weighted Word2Vec')
```

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

In [79]:
```
clf_rbf_tfidfW2v = CalibratedClassifierCV(base_estimator = rbf_tfidfW2v_model.best_estimator_, cv="prefit")
clf_rbf_tfidfW2v.fit(trainTfidfWord2Vectors[:20000],y_train[:20000])
plt.title("Tfidf Weighted Word2Vec")
```



In [80]:
```
plt.title("Tfidf Weighted Word2Vec")
```



In [81]:
```
table.add_row(["RBF SVM(BOW)",
                   rbf_bigram_model.best_params_,
                   np.round(rbf_bigram_model.score(bigrams_train[:20000],y_train[:20000]),5),
                   np.round(rbf_bigram_model.score(bigrams_test[:5000],y_test[:5000]),5)])
table.add_row(["RBF SVM(TFIDF)",
                   tfidf_rbf_bigram_model.best_params_,
                   np.round(tfidf_rbf_bigram_model.score(tfidf_bigrams_train[:20000],y_train[:20000]),5),
                   np.round(tfidf_rbf_bigram_model.score(tfidf_bigrams_test[:5000],y_test[:5000]),5)])
table.add_row(["RBF SVM(Avg W2V)",
                   rbf_avgW2v_model.best_params_,
                   np.round(rbf_avgW2v_model.score(trainWord2Vectors[:20000],y_train[:20000]),5),
                   np.round(rbf_avgW2v_model.score(testWord2Vectors[:5000],y_test[:5000]),5)])
table.add_row(["RBF SVM(TFIDF W2V)",
                   rbf_tfidfW2v_model.best_params_,
                   np.round(rbf_tfidfW2v_model.score(trainTfidfWord2Vectors[:20000],y_train[:20000]),5),
```

[7] Conclusions

In [82]:

```
print(table)
```

Vectoriser	parameters	Train AUC Score	Test AUC score
Linear SVM(BOW)	{'alpha': 6.30957344480193, 'penalty': 'l2'}	0.98598	0.96321
Linear SVM(TFIDF)	{'alpha': 6.30957344480193, 'penalty': 'l2'}	0.98726	0.9666
Linear SVM(Avg W2V)	{'alpha': 0.046415888336127774, 'penalty': 'l2'}	0.75871	0.76179
Linear SVM(TFIDF W2V)	{'alpha': 0.046415888336127774, 'penalty': 'l2'}	0.7005	0.70305
RBF SVM(BOW)	{'C': 1.0, 'gamma': 'scale'}	0.98233	0.88863
RBF SVM(TFIDF)	{'C': 1.0, 'gamma': 'scale'}	0.99792	0.90103
RBF SVM(Avg W2V)	{'C': 1.0, 'gamma': 'scale'}	0.96689	0.78255
RBF SVM(TFIDF W2V)	{'C': 1.0, 'gamma': 'scale'}	0.96658	0.73667