KNN_AmazonFineFoodReviewsPreProcessing

February 3, 2019

1 Amazon Fine Food Reviews Preprocessing

This IPython notebook consists code for preprocessing of text, conversion of text into vectors and saving that information for further use.

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

1.1 Public Information -

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon. 1. Number of reviews: 568,454 2. Number of users: 256,059 3. Number of products: 74,258 4. Timespan: Oct 1999 - Oct 2012 5. Number of Attributes/Columns in data: 10

1.1.1 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

1.1.2 Current Objective -

Go through the reviews and perform preprocessing, convert them into vectors and save them for future use.

2 [1] Reading Data

2.1 [1.1] Loading data and libraries

```
In [0]: #mounting the dataset from drive
        from google.colab import drive
        drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6

Enter your authorization code:
··········
Mounted at /content/gdrive
```

In [0]:
```
!pip install numba
```

```
Requirement already satisfied: numba in /usr/local/lib/python3.6/dist-packages (0.40.1)
Requirement already satisfied: llvmlite>=0.25.0dev0 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from numba) (1
```

In [0]:
```
#importing necessary libraries
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import missingno as msno

from nltk.stem.wordnet import WordNetLemmatizer
import re
from nltk.corpus import stopwords
from nltk import pos_tag, word_tokenize

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

import nltk
import pickle

from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from gensim.models import Word2Vec
from concurrent.futures import ThreadPoolExecutor, ProcessPoolExecutor
from concurrent import futures

from numba import jit
```

In [0]:
```
!python -m nltk.downloader stopwords
```

```
/usr/lib/python3.6/runpy.py:125: RuntimeWarning: 'nltk.downloader' found in sys.modules after i
  warn(RuntimeWarning(msg))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

In [0]:
```
!python -m nltk.downloader punkt averaged_perceptron_tagger wordnet
```

```
/usr/lib/python3.6/runpy.py:125: RuntimeWarning: 'nltk.downloader' found in sys.modules after i
  warn(RuntimeWarning(msg))
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
```

In [0]:
```
#connecting to sqlite db
con = sqlite3.connect('/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_revie

#filtering only positive and negative reviews
data = pd.read_sql_query("SELECT * FROM Reviews WHERE Score != 3", con)

print("Shape of data:", data.shape)

#scores < 3 are considered to be negative reviews and > 3 are considered to be positiv
data.head()
```

```
Shape of data: (525814, 10)
```

Out[0]:
```
   Id   ProductId          UserId                      ProfileName  \
0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian
1   2  B00813GRG4  A1D87F6ZCVE5NK                           dll pa
2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"
3   4  B000UA0QIQ  A395BORC6FGVXV                             Karl
4   5  B006K2ZZ7K  A1UQRSCLF8GW1T    Michael D. Bigham "M. Wassir"

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     1                       1      5  1303862400
1                     0                       0      1  1346976000
2                     1                       1      4  1219017600
3                     3                       3      2  1307923200
4                     0                       0      5  1350777600

                 Summary                                               Text
0  Good Quality Dog Food  I have bought several of the Vitality canned d...
1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
2  "Delight" says it all  This is a confection that has been around a fe...
```

```
3         Cough Medicine  If you are looking for the secret ingredient i...
4            Great taffy  Great taffy at a great price.  There was a wid...
```

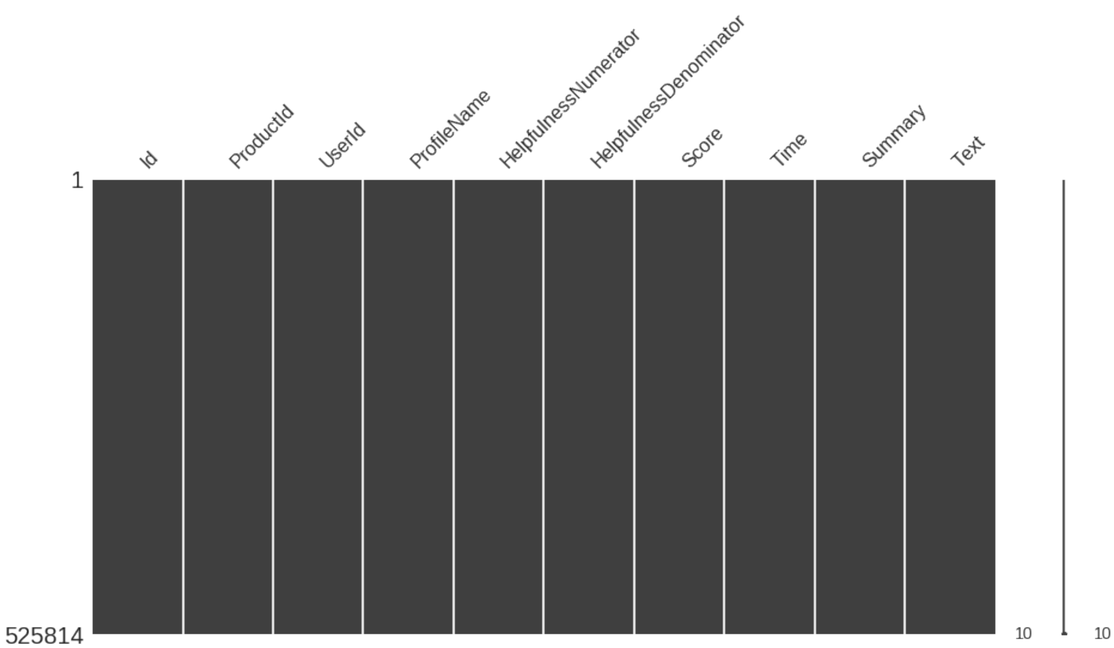
3 [2] Exploratory Data Analysis

3.1 [2.1] Data Cleaning: Missing values

In [0]:
```
#let's just check, just in case if any
print("Missing values? Ans -", data.isnull().values.any())

#visualizing it
msno.matrix(data, figsize=(15,7))
```

```
Missing values? Ans - False
```

Out[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f35bf4505f8>
```



3.2 [2.2] Data cleaning: Multiple reviews for the same product by same person

In [0]:
```
df = data.copy()
df['ProdUser'] = df['ProductId'] + df['UserId']
df[df['ProdUser'].duplicated(keep=False)].sort_values('ProdUser', axis=0, ascending=Tr
```

Out[0]:
```
            Id   ProductId          UserId  \
157863  171174  7310172001   AE9ZBY7WW3LIQ
```

157871	171183	7310172001	AE9ZBY7WW3LIQ
157912	171228	7310172001	AJD41FBJD9010
157841	171152	7310172001	AJD41FBJD9010
157842	171153	7310172001	AJD41FBJD9010
157843	171154	7310172001	AJD41FBJD9010
157876	171189	7310172001	AJD41FBJD9010
157908	171223	7310172001	AJD41FBJD9010
200626	217414	7310172101	AE9ZBY7WW3LIQ
200618	217405	7310172101	AE9ZBY7WW3LIQ
200597	217384	7310172101	AJD41FBJD9010
200631	217420	7310172101	AJD41FBJD9010
200598	217385	7310172101	AJD41FBJD9010
200663	217454	7310172101	AJD41FBJD9010
200667	217459	7310172101	AJD41FBJD9010
200596	217383	7310172101	AJD41FBJD9010
346048	374351	B00004CI84	A1K94LXX833JTT
346106	374412	B00004CI84	A1K94LXX833JTT
346119	374425	B00004CI84	A1K94LXX833JTT
417917	451939	B00004CXX9	A1K94LXX833JTT
417853	451871	B00004CXX9	A1K94LXX833JTT
417930	451952	B00004CXX9	A1K94LXX833JTT
212523	230338	B00004RYGX	A1K94LXX833JTT
212465	230277	B00004RYGX	A1K94LXX833JTT
212536	230351	B00004RYGX	A1K94LXX833JTT
341832	369818	B000084DWM	A25C5MVVCIYT5D
341815	369799	B000084DWM	A25C5MVVCIYT5D
341817	369801	B000084DWM	A36JDIN9RAAIEC
341818	369802	B000084DWM	A36JDIN9RAAIEC
341806	369790	B000084DWM	A36JDIN9RAAIEC
...	...	...	...
411612	445161	B009GHI5Q4	A3TVZM3ZIXG8YW
411671	445223	B009GHI5Q4	A3TVZM3ZIXG8YW
411611	445160	B009GHI5Q4	A3TVZM3ZIXG8YW
411608	445157	B009GHI5Q4	A3TVZM3ZIXG8YW
411603	445152	B009GHI5Q4	A3TVZM3ZIXG8YW
411599	445147	B009GHI5Q4	A3TVZM3ZIXG8YW
411621	445170	B009GHI5Q4	A3TVZM3ZIXG8YW
411659	445211	B009GHI5Q4	A3TVZM3ZIXG8YW
411670	445222	B009GHI5Q4	A3TVZM3ZIXG8YW
411613	445162	B009GHI5Q4	A3TVZM3ZIXG8YW
411631	445181	B009GHI5Q4	A966L65JSN8XN
411651	445203	B009GHI5Q4	A966L65JSN8XN
62140	67512	B009GHI6I6	A2ISKAWUPGGOLZ
62142	67515	B009GHI6I6	A2ISKAWUPGGOLZ
62138	67510	B009GHI6I6	A3TVZM3ZIXG8YW
62143	67516	B009GHI6I6	A3TVZM3ZIXG8YW
463853	501546	B009M2LUEW	A2AY7WOD04JYMY
463852	501545	B009M2LUEW	A2AY7WOD04JYMY

386528	417991	B009RB4G04	A1FQSVI2WVV5W5
386522	417984	B009RB4G04	A1FQSVI2WVV5W5
386525	417988	B009RB4G04	A1N06XIVTDQMP
386455	417911	B009RB4G04	A1N06XIVTDQMP
386483	417942	B009RB4G04	A21GDMT9JN2A5Y
386431	417884	B009RB4G04	A21GDMT9JN2A5Y
386530	417993	B009RB4G04	A353Y7VBQHHW0T
386486	417946	B009RB4G04	A353Y7VBQHHW0T
386492	417952	B009RB4G04	A3QVP3B2VVJ9T0
386356	417800	B009RB4G04	A3QVP3B2VVJ9T0
386460	417917	B009RB4G04	ANMGYT60QP4CM
386458	417914	B009RB4G04	ANMGYT60QP4CM

	ProfileName	HelpfulnessNumerator \
157863	W. K. Ota	0
157871	W. K. Ota	5
157912	N. Ferguson "Two, Daisy, Hannah, and Kitten"	5
157841	N. Ferguson "Two, Daisy, Hannah, and Kitten"	0
157842	N. Ferguson "Two, Daisy, Hannah, and Kitten"	0
157843	N. Ferguson "Two, Daisy, Hannah, and Kitten"	0
157876	N. Ferguson "Two, Daisy, Hannah, and Kitten"	39
157908	N. Ferguson "Two, Daisy, Hannah, and Kitten"	1
200626	W. K. Ota	5
200618	W. K. Ota	0
200597	N. Ferguson "Two, Daisy, Hannah, and Kitten"	0
200631	N. Ferguson "Two, Daisy, Hannah, and Kitten"	39
200598	N. Ferguson "Two, Daisy, Hannah, and Kitten"	0
200663	N. Ferguson "Two, Daisy, Hannah, and Kitten"	1
200667	N. Ferguson "Two, Daisy, Hannah, and Kitten"	5
200596	N. Ferguson "Two, Daisy, Hannah, and Kitten"	0
346048	Sanpete	1
346106	Sanpete	8
346119	Sanpete	10
417917	Sanpete	8
417853	Sanpete	1
417930	Sanpete	10
212523	Sanpete	8
212465	Sanpete	1
212536	Sanpete	10
341832	Natalie Dawn	1
341815	Natalie Dawn	2
341817	Jon	2
341818	Jon	2
341806	Jon	3
...	...	...
411612	christopher hayes	11
411671	christopher hayes	6
411611	christopher hayes	7

411608	christopher hayes	3
411603	christopher hayes	18
411599	christopher hayes	19
411621	christopher hayes	33
411659	christopher hayes	2
411670	christopher hayes	6
411613	christopher hayes	11
411631	N. Schleif "night owl"	1
411651	N. Schleif "night owl"	0
62140	M. S. Handley	2
62142	M. S. Handley	0
62138	christopher hayes	7
62143	christopher hayes	0
463853	J. Norden	1
463852	J. Norden	1
386528	JLF	3
386522	JLF	1
386525	LadyRae13	1
386455	LadyRae13	0
386483	Wayward Traveller "WaywardT"	0
386431	Wayward Traveller "WaywardT"	5
386530	wackygirl "wackygirl"	3
386486	wackygirl "wackygirl"	5
386492	B. Fitzpatrick "BAFXF"	2
386356	B. Fitzpatrick "BAFXF"	0
386460	Patricia Kagie	0
386458	Patricia Kagie	0

	HelpfulnessDenominator	Score	Time	\
157863	0	4	1182902400	
157871	13	1	1219363200	
157912	7	5	1233360000	
157841	0	5	1233360000	
157842	0	5	1233360000	
157843	0	5	1233360000	
157876	51	5	1233360000	
157908	1	5	1233360000	
200626	13	1	1219363200	
200618	0	4	1182902400	
200597	0	5	1233360000	
200631	51	5	1233360000	
200598	0	5	1233360000	
200663	1	5	1233360000	
200667	7	5	1233360000	
200596	0	5	1233360000	
346048	2	5	1211760000	
346106	10	4	1213747200	
346119	14	4	1213747200	

417917	10	4	1213747200
417853	2	5	1211760000
417930	14	4	1213747200
212523	10	4	1213747200
212465	2	5	1211760000
212536	14	4	1213747200
341832	1	5	1304726400
341815	2	5	1304726400
341817	2	5	1292976000
341818	2	5	1292976000
341806	3	5	1292976000
...	...	...	...
411612	15	1	1291420800
411671	15	1	1291420800
411611	9	1	1291420800
411608	3	1	1291420800
411603	24	1	1291420800
411599	21	1	1291420800
411621	48	1	1291420800
411659	4	1	1291420800
411670	14	1	1291420800
411613	15	1	1291420800
411631	1	5	1319241600
411651	0	5	1323820800
62140	4	1	1310774400
62142	1	1	1310774400
62138	11	1	1291420800
62143	2	1	1291420800
463853	1	5	1252195200
463852	1	5	1252713600
386528	4	1	1319760000
386522	1	1	1319760000
386525	1	5	1316563200
386455	0	4	1316563200
386483	1	1	1309910400
386431	5	1	1309910400
386530	4	5	1318896000
386486	10	5	1303776000
386492	2	1	1327017600
386356	0	1	1332633600
386460	0	5	1311120000
386458	0	5	1315785600

	Summary \
157863	Best snack item for my dog.
157871	Why sbould I get crums?
157912	NO waste at all--- all dogs love liver treats-...
157841	dogs LOVE it-- best treat for rewards and tra...

```
157842  best dog treat-- great for training---  all do...
157843  best dog treat-- great for training---  all do...
157876  NO waste at all ----  great for training  ----...
157908  best dog treat-- great for training---  all do...
200626                            Why sbould I get crums?
200618                        Best snack item for my dog.
200597  best dog treat-- great for training---  all do...
200631  NO waste at all ----  great for training  ----...
200598  best dog treat-- great for training---  all do...
200663  best dog treat-- great for training---  all do...
200667  NO waste at all--- all dogs love liver treats-...
200596  dogs LOVE it--  best treat for rewards and tra...
346048  Heads off, I mean up, Beetlejuice fans!  New D...
346106  Some early details about the new 20th Annivers...
346119  Some early details about the new Blu-ray DVD d...
417917  Some early details about the new 20th Annivers...
417853  Heads off, I mean up, Beetlejuice fans!  New D...
417930  Some early details about the new Blu-ray DVD d...
212523  Some early details about the new 20th Annivers...
212465  Heads off, I mean up, Beetlejuice fans!  New D...
212536  Some early details about the new Blu-ray DVD d...
341832                         The only thing that worked
341815                                 Nothing else works
341817     Great product, but trust your vet not the hype
341818      Don't fall prey to fads and anecdotal reviews
341806     Great product, but trust your vet not the hype
...                                                   ...
411612  Filler food is empty, leaves your cat always n...
411671  Filler food is empty, leaves your cat always n...
411611  Filler food is empty, leaves your cat always n...
411608  Filler food is empty, leaves your cat always n...
411603  Filler food is empty, leaves your cat always n...
411599  Filler food is empty, leaves your cat always n...
411621  Filler food is empty, leaves your cat always n...
411659  Filler food is empty, leaves your cat always n...
411670  Filler food is empty, leaves your cat always n...
411613  Filler food is empty, leaves your cat always n...
411631               The only thing all my cats will eat!
411651                     Only food my cats can agree on
62140                                    Kitty Junk Food
62142                                    Kitty Junk Food
62138   Filler food is empty, leaves your cat always n...
62143   Filler food is empty, leaves your cat always n...
463853                 Can't believe it's not REAL sugar!
463852                                      Perfect taste!
386528                                          Too Sweet
386522                                     Way too sweet!
386525                                          Yummmm...
```

```
386455                                          Pretty Good
386483    Does not taste at all like I thought I would :)
386431           The worst tasting thing out of my Keurig
386530             Tried it loved it...subscribing to it.
386486                        This Apple Cider is Awesome
386492                                    Unspeakably bad
386356                              Tastes Like Chemicals
386460                                  Grove Square Cider
386458                                  Grove Square Cider

                                                       Text  \
157863  Otter and I are very happy with this product. ...
157871  I selected this company over the other even th...
157912  Freeze dried liver has a hypnotic effect on do...
157841  Freeze dried liver has a hypnotic effect on do...
157842  Freeze dried liver has a hypnotic effect on do...
157843  Freeze dried liver has a hypnotic effect on do...
157876  Freeze dried liver has a hypnotic effect on do...
157908  Freeze dried liver has a hypnotic effect on do...
200626  I selected this company over the other even th...
200618  Otter and I are very happy with this product. ...
200597  Freeze dried liver has a hypnotic effect on do...
200631  Freeze dried liver has a hypnotic effect on do...
200598  Freeze dried liver has a hypnotic effect on do...
200663  Freeze dried liver has a hypnotic effect on do...
200667  Freeze dried liver has a hypnotic effect on do...
200596  Freeze dried liver has a hypnotic effect on do...
346048  Beetlejuice is coming back to DVD in a newly r...
346106  Beetlejuice is a very Tim Burtonesque Tim Burt...
346119  Beetlejuice is a very Tim Burtonesque Tim Burt...
417917  Beetlejuice is a very Tim Burtonesque Tim Burt...
417853  Beetlejuice is coming back to DVD in a newly r...
417930  Beetlejuice is a very Tim Burtonesque Tim Burt...
212523  Beetlejuice is a very Tim Burtonesque Tim Burt...
212465  Beetlejuice is coming back to DVD in a newly r...
212536  Beetlejuice is a very Tim Burtonesque Tim Burt...
341832  I understand all the complaints about Science ...
341815  I understand all the complaints about Science ...
341817  I have two cats, one 6 and one 2 years old. Bo...
341818  I have two cats, one 6 and one 2 years old. Bo...
341806  I have two cats, one 6 and one 2 years old. Bo...
...                                                   ...
411612  This review will make me sound really stupid, ...
411671  This review will make me sound really stupid, ...
411611  This review will make me sound really stupid, ...
411608  This review will make me sound really stupid, ...
411603  This review will make me sound really stupid, ...
411599  This review will make me sound really stupid, ...
```

```
411621  This review will make me sound really stupid, ...
411659  This review will make me sound really stupid, ...
411670  This review will make me sound really stupid, ...
411613  This review will make me sound really stupid, ...
411631  I have three very different cats, with very di...
411651  This is the only flavor of Science Diet (or an...
62140   We have five cats - one an elderly cat of 15 y...
62142   We have five cats - one an elderly cat of 15 y...
62138   This review will make me sound really stupid, ...
62143   This review will make me sound really stupid, ...
463853  Tastes great!  A super alternative to artifica...
463852  I will buy this product again and again.  Even...
386528  Tasted way too sweet with a terrible after tas...
386522  I posted a review for the Spiced Apple Cider b...
386525  So delicious for K Cup Brewers...could not ge...
386455  Tastes wonderful and is caramel-y, but could b...
386483  I really had high expectations of this product...
386431  I really had high expectations of this product...
386530  This one is good.  I am going to try to Carmel...
386486  I have to say I was afraid to try it because o...
386492  This was unspeakably bad.  Tasked like raw che...
386356  I was very disappointed.  I love good cider, h...
386460  This is a great product and an added bonus is ...
386458  This is a great product for the K-cups. It tas...

                        ProdUser
157863   7310172001AE9ZBY7WW3LIQ
157871   7310172001AE9ZBY7WW3LIQ
157912   7310172001AJD41FBJD9010
157841   7310172001AJD41FBJD9010
157842   7310172001AJD41FBJD9010
157843   7310172001AJD41FBJD9010
157876   7310172001AJD41FBJD9010
157908   7310172001AJD41FBJD9010
200626   7310172101AE9ZBY7WW3LIQ
200618   7310172101AE9ZBY7WW3LIQ
200597   7310172101AJD41FBJD9010
200631   7310172101AJD41FBJD9010
200598   7310172101AJD41FBJD9010
200663   7310172101AJD41FBJD9010
200667   7310172101AJD41FBJD9010
200596   7310172101AJD41FBJD9010
346048  B00004CI84A1K94LXX833JTT
346106  B00004CI84A1K94LXX833JTT
346119  B00004CI84A1K94LXX833JTT
417917  B00004CXX9A1K94LXX833JTT
417853  B00004CXX9A1K94LXX833JTT
417930  B00004CXX9A1K94LXX833JTT
```

```
212523  B00004RYGXA1K94LXX833JTT
212465  B00004RYGXA1K94LXX833JTT
212536  B00004RYGXA1K94LXX833JTT
341832  B000084DWMA25C5MVVCIYT5D
341815  B000084DWMA25C5MVVCIYT5D
341817  B000084DWMA36JDIN9RAAIEC
341818  B000084DWMA36JDIN9RAAIEC
341806  B000084DWMA36JDIN9RAAIEC
...                          ...
411612  B009GHI5Q4A3TVZM3ZIXG8YW
411671  B009GHI5Q4A3TVZM3ZIXG8YW
411611  B009GHI5Q4A3TVZM3ZIXG8YW
411608  B009GHI5Q4A3TVZM3ZIXG8YW
411603  B009GHI5Q4A3TVZM3ZIXG8YW
411599  B009GHI5Q4A3TVZM3ZIXG8YW
411621  B009GHI5Q4A3TVZM3ZIXG8YW
411659  B009GHI5Q4A3TVZM3ZIXG8YW
411670  B009GHI5Q4A3TVZM3ZIXG8YW
411613  B009GHI5Q4A3TVZM3ZIXG8YW
411631   B009GHI5Q4A966L65JSN8XN
411651   B009GHI5Q4A966L65JSN8XN
62140   B009GHI6I6A2ISKAWUPGGOLZ
62142   B009GHI6I6A2ISKAWUPGGOLZ
62138   B009GHI6I6A3TVZM3ZIXG8YW
62143   B009GHI6I6A3TVZM3ZIXG8YW
463853  B009M2LUEWA2AY7WOD04JYMY
463852  B009M2LUEWA2AY7WOD04JYMY
386528  B009RB4GO4A1FQSVI2WVV5W5
386522  B009RB4GO4A1FQSVI2WVV5W5
386525   B009RB4GO4A1N06XIVTDQMP
386455   B009RB4GO4A1N06XIVTDQMP
386483  B009RB4GO4A21GDMT9JN2A5Y
386431  B009RB4GO4A21GDMT9JN2A5Y
386530  B009RB4GO4A353Y7VBQHHW0T
386486  B009RB4GO4A353Y7VBQHHW0T
386492  B009RB4GO4A3QVP3B2VVJ9T0
386356  B009RB4GO4A3QVP3B2VVJ9T0
386460   B009RB4GO4ANMGYT60QP4CM
386458   B009RB4GO4ANMGYT60QP4CM

[11988 rows x 11 columns]
```

3.2.1 Obeservations

1. There are some instances where a user has written more than one review for the same product.
2. We can remove the one which has less Helpfulness but lets keep all and treat it as review from a different user.

3. Will definitely have to remove same reviews because it is just redundant data.

```
In [0]: #Sorting data according to ProductId in ascending order
        data = data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicl
```

3.3 [2.3] Data cleaning: Deduplication - 1

```
In [0]: #Deduplication of entries
        data=data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first',
        data.shape

Out[0]: (364173, 10)

In [0]: data.head(2)

Out[0]:             Id   ProductId          UserId      ProfileName  \
        138706  150524  0006641040   ACITT7DI6IDDL  shari zychinski
        138688  150506  0006641040  A2IW4PEEKO2R0U            Tracy

                HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
        138706                     0                       0      5   939340800
        138688                     1                       1      4  1194739200

                                              Summary  \
        138706                  EVERY book is educational
        138688  Love the book, miss the hard cover version

                                                           Text
        138706  this witty little book makes my son laugh at l...
        138688  I grew up reading these Sendak books, and watc...
```

3.4 [2.4] Data cleaning: Deduplication - 2

Same reviews on multiple products with different timestamps

```
In [0]: data[data['Text'].duplicated(keep=False)].sort_values('Text', axis=0, ascending=True, :

Out[0]:             Id   ProductId          UserId  \
        67574    73444  B0046IISFG  A3OXHLG6DIBRW8
        287090  311004  B001EO6FPU  A3OXHLG6DIBRW8
        302818  327982  B0000CEQ6H  A281NPSIMI1C2R
        494235  534333  B0000CEQ72  A281NPSIMI1C2R
        387315  418839  B000FZYSVC  A1YUL9PCJR3JTY
        164025  177904  B000PSFW9Q  A1YUL9PCJR3JTY
        267899  290387  B000S85AVI  A1YUL9PCJR3JTY
        443822  479891  B000Z91YTC  A1YUL9PCJR3JTY
        442191  478132  B0001GSP9G  A1YUL9PCJR3JTY
        177373  192340  B000M7OWLE  A1YUL9PCJR3JTY
        349975  378572  B0001GSPC8  A1YUL9PCJR3JTY
```

308770	334367	B000M7OWMS	A1YUL9PCJR3JTY
432171	467365	B0002W0RX6	A1YUL9PCJR3JTY
306132	331530	B004JJ6ZN4	A1YUL9PCJR3JTY
68214	74193	B000E4AHAK	A1YUL9PCJR3JTY
36692	39874	B000CMIZ0I	A1YUL9PCJR3JTY
204048	221073	B0001N4890	A1YUL9PCJR3JTY
61803	67142	B0000CGFSC	A1YUL9PCJR3JTY
524984	567556	B003ULE0TS	A1YUL9PCJR3JTY
378979	409774	B000QVDP6Y	A1YUL9PCJR3JTY
438391	474076	B000CQC08C	A1YUL9PCJR3JTY
442055	477988	B000BOZ6K4	A1YUL9PCJR3JTY
410856	444343	B0001M0ZTI	A1YUL9PCJR3JTY
113563	123178	B000CQBZOW	A1YUL9PCJR3JTY
360276	389666	B000Q61HH8	A1YUL9PCJR3JTY
488311	528026	B000EUCKF4	A1YUL9PCJR3JTY
367930	397829	B000PIMWGM	A1YUL9PCJR3JTY
367928	397826	B000PIMWGM	A1YUL9PCJR3JTY
227146	246294	B0009F3SB4	A1YUL9PCJR3JTY
443856	479926	B000VV0512	A1YUL9PCJR3JTY
...	...	...	...
484592	523982	B004JGQ15Y	A1KEK09ZA6J9P8
156517	169743	B004JGQ16I	A1KEK09ZA6J9P8
59565	64702	B0002ERVTM	A281NPSIMI1C2R
401926	434586	B000F9BCLW	A281NPSIMI1C2R
146793	159233	B002IYDXVE	A3R7Q2RWQ8K2S7
357423	386592	B003TN6ZN6	A3R7Q2RWQ8K2S7
503424	544379	B002BCE97K	A3BTL4FV6ODKAT
246458	267240	B000E123IC	A3BTL4FV6ODKAT
505442	546536	B000E148MG	A3BTL4FV6ODKAT
200791	217587	B000N8OLCC	A3BTL4FV6ODKAT
314550	340574	B0018CJYCO	A2AHTUMQC1O3M8
140727	152726	B0018CIPS8	A2AHTUMQC1O3M8
469169	507321	B000EEDJGE	A2OEEWWSFMZ1PN
35905	39033	B002PXEQCS	A2OEEWWSFMZ1PN
479858	518889	B003BXOAKE	A3QZ6JT0R1OWEC
514622	556404	B000IBILV6	A3QZ6JT0R1OWEC
138146	149927	B0028C44IM	AC8C9PT59CDW1
447742	484120	B001IZ9ME6	AC8C9PT59CDW1
485560	525050	B0010B6IFY	A21B8AV7E3MPXE
38078	41352	B0096EZHM2	A21B8AV7E3MPXE
54700	59365	B000FBM3RC	A1CVN6FWUCZOMD
151003	163798	B000FBKFRW	A1CVN6FWUCZOMD
37746	40992	B001T5GHUM	A3PS4V0JQ2003X
118574	128597	B0026A2BS6	A3PS4V0JQ2003X
76868	83624	B005ZBZLT4	A3LL0U6E3QK34A
167105	181178	B007Y59HVM	A3LL0U6E3QK34A
242532	263029	B006N3HYYS	A3RFWQMLYSAKI0
99008	107540	B007TJGY4Q	A3RFWQMLYSAKI0

```
521495  563807  B007JFMH8M  A248UQ9YXAMO9Z
521496  563808  B007JFMH8M  A3IMUU0I31XF33

                                        ProfileName  \
67574                                C. F. Hill "CFH"
287090                               C. F. Hill "CFH"
302818           Rebecca of Amazon "The Rebecca Review"
494235           Rebecca of Amazon "The Rebecca Review"
387315                  O. Brown "Ms. O. Khannah-Brown"
164025                  O. Brown "Ms. O. Khannah-Brown"
267899                  O. Brown "Ms. O. Khannah-Brown"
443822                  O. Brown "Ms. O. Khannah-Brown"
442191                  O. Brown "Ms. O. Khannah-Brown"
177373                  O. Brown "Ms. O. Khannah-Brown"
349975                  O. Brown "Ms. O. Khannah-Brown"
308770                  O. Brown "Ms. O. Khannah-Brown"
432171                  O. Brown "Ms. O. Khannah-Brown"
306132                  O. Brown "Ms. O. Khannah-Brown"
68214                   O. Brown "Ms. O. Khannah-Brown"
36692                   O. Brown "Ms. O. Khannah-Brown"
204048                  O. Brown "Ms. O. Khannah-Brown"
61803                   O. Brown "Ms. O. Khannah-Brown"
524984                  O. Brown "Ms. O. Khannah-Brown"
378979                  O. Brown "Ms. O. Khannah-Brown"
438391                  O. Brown "Ms. O. Khannah-Brown"
442055                  O. Brown "Ms. O. Khannah-Brown"
410856                  O. Brown "Ms. O. Khannah-Brown"
113563                  O. Brown "Ms. O. Khannah-Brown"
360276                  O. Brown "Ms. O. Khannah-Brown"
488311                  O. Brown "Ms. O. Khannah-Brown"
367930                  O. Brown "Ms. O. Khannah-Brown"
367928                  O. Brown "Ms. O. Khannah-Brown"
227146                  O. Brown "Ms. O. Khannah-Brown"
443856                  O. Brown "Ms. O. Khannah-Brown"
...                                                 ...
484592                             Colleen M. Schneider
156517                             Colleen M. Schneider
59565            Rebecca of Amazon "The Rebecca Review"
401926           Rebecca of Amazon "The Rebecca Review"
146793                                         MamaCito
357423                                         MamaCito
503424  fredtownward "The Analytical Mind; Have Brain...
246458  fredtownward "The Analytical Mind; Have Brain...
505442  fredtownward "The Analytical Mind; Have Brain...
200791  fredtownward "The Analytical Mind; Have Brain...
314550                             Glenn Wagstaff "GBW"
140727                             Glenn Wagstaff "GBW"
469169                               bernie "webviator"
```

35905	bernie "webviator"
479858	M. Goldman "M_gold~"
514622	M. Goldman "M_gold~"
138146	M.A.R.
447742	M.A.R.
485560	Natalie V. Galasso
38078	Natalie V. Galasso
54700	A Customer
151003	his_billyness
37746	PookieThePirate
118574	PookieThePirate
76868	A Customer
167105	K. Biddle
242532	Michael Burkett "reader rider"
99008	A Customer
521495	Becky
521496	Becky

	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time \
67574	1	1	5	1342915200
287090	9	9	5	1297036800
302818	3	3	5	1084492800
494235	1	1	5	1093651200
387315	1	1	5	1173052800
164025	1	1	5	1156723200
267899	2	2	5	1173052800
443822	6	6	5	1156723200
442191	1	1	5	1156723200
177373	1	1	5	1173052800
349975	1	1	5	1156896000
308770	3	3	5	1173052800
432171	2	2	5	1157241600
306132	0	0	5	1156723200
68214	4	4	5	1181952000
36692	13	13	5	1181692800
204048	0	0	5	1170460800
61803	1	1	5	1170374400
524984	2	2	5	1296000000
378979	2	2	5	1156723200
438391	2	2	5	1281916800
442055	2	2	5	1191628800
410856	1	1	5	1191628800
113563	2	2	5	1282003200
360276	1	1	5	1189987200
488311	15	15	5	1217030400
367930	1	2	5	1187654400
367928	10	11	5	1187740800
227146	22	24	5	1189468800

```
443856                  0                       0      5  1190764800
...                   ...                     ...    ...         ...
484592                  0                       0      5  1301529600
156517                  0                       0      5  1300752000
59565                   1                       1      5  1230249600
401926                  0                       0      5  1169078400
146793                  0                       0      4  1300406400
357423                  9                       9      4  1300492800
503424                  1                       1      5  1329609600
246458                  0                       0      5  1329350400
505442                  1                       2      5  1329091200
200791                  0                       0      5  1329177600
314550                  1                       1      5  1297382400
140727                  1                       2      5  1297123200
469169                  1                       1      5  1316908800
35905                   1                       1      5  1344643200
479858                  0                       0      1  1349481600
514622                  0                       0      1  1324080000
138146                  0                       0      5  1333756800
447742                  0                       0      5  1330732800
485560                  2                       2      5  1304121600
38078                   3                       3      4  1304467200
54700                   2                       2      5  1169596800
151003                  2                       2      5  1169596800
37746                   8                       9      5  1313452800
118574                  9                      10      5  1314144000
76868                   0                       1      4  1341619200
167105                  0                       1      4  1341619200
242532                  0                       8      4  1298592000
99008                   0                       8      4  1298592000
521495                  0                       0      5  1341792000
521496                  0                       0      5  1341792000

                                                   Summary  \
67574   Great Diabetic Friendly Sweetwner - Highly Rec...
287090  Great Diabetic Friendly Sweetener - Highly Rec...
302818                          Superior for Bread Baking
494235                  Bob's Red Mill Whole Wheat Flour
387315                   Perfect Morning Tea (Caffeinated)
164025                   Perfect Morning Tea (Caffeinated)
267899                  Golden Chai Must Be Experienced!
443822                  Golden Chai Must Be Experienced!
442191               Superb Black Tea Blend (Caffeinated)
177373               Superb Black Tea Blend (Caffeinated)
349975             Wonderful Rooibos---The "Miracle Tea"!
308770             Wonderful Rooibos---The "Miracle Tea"!
432171        A Low-Caffeine, Hand-Rolled Fine Green Tea
306132        A Low-Caffeine, Hand-Rolled Fine Green Tea
```

```
68214   Original Idea---Kombucha from a Tea! And Decaf...
36692   Original Idea---Kombucha from a Tea! And Decaf...
204048           Distinctive, great-flavored Japanese tea
61803            Distinctive, great-flavored Japanese tea
524984               Beautiful Morning Tea, With Caffeine
378979                             Very Nice Morning Tea
438391                                  Soft, Lovely Tea
442055                                Soft, Subtle Blend
410856                               Sweet and Delicious
113563                              Sweet and Lovely Tea
360276            Great Green Tea Experience, Tazo's Best
488311              Fantastic Green Tea, the Best of Tazo
367930                Healthy Tea Supports Weight Control
367928                Healthy Tea Supports Weight Control
227146                Healthy Tea Supports Weight Control
443856           Romantic, Natural, Unique Tea Experience
...                                                   ...
484592  For when the girl scouts (and your stash) are ...
156517  for when the girl scouts and your stash are go...
59565                                    Instant Chai Tea
401926             Chai Tea with Whipped Cream and Nutmeg
146793                                          Lifesaver
357423                                          Lifesaver
503424         The Best Instant Noodle Meals You Can Buy!
246458         The Best Instant Noodle Meals You Can Buy!
505442         The Best Instant Noodle Meals You Can Buy!
200791         The Best Instant Noodle Meals You Can Buy!
314550                          Great food at a good price
140727                          Great food at a good price
469169                       Get them free in your SP Pack
35905                        Get them free in your SP Pack
479858  Made in China treats still KILLING dogs - ABC ...
514622  Made in China treats still KILLING dogs - ABC ...
138146                                    Love these mints
447742                                    Love these mints
485560                                   Cheaper on Amazom
38078                              Better prices on Amazom
54700                              unusual chocolate treat
151003                             unusual chocolate treat
37746   CHECK YOUR LOCAL ASIAN STORE FIRST! CHEAP IN S...
118574         CHECK STORE PRICE FIRST! CHEAPER IN STORE!
76868                                             love it
167105                                            love it
242532  great tasting bold coffee.  make sure you're g...
99008   great tasting bold coffee.  make sure you're g...
521495                                              yummy
521496                                  yummy great cookie
```

```
                                                    Text
67574   "Erythritol" has become one of our favorite su...
287090  "Erythritol" has become one of our favorite su...
302818  "We use and believe in stone milling because n...
494235  "We use and believe in stone milling because n...
387315  *****<br /><br />Numi Tea's Chinese Breakfast ...
164025  *****<br /><br />Numi Tea's Chinese Breakfast ...
267899  *****<br /><br />Numi Tea's Golden Chai Spiced...
443822  *****<br /><br />Numi Tea's Golden Chai Spiced...
442191  *****<br /><br />Numi Tea's Morning Rise Break...
177373  *****<br /><br />Numi Tea's Morning Rise Break...
349975  *****<br /><br />Red Mellow Bush is a premium ...
308770  *****<br /><br />Red Mellow Bush is a premium ...
432171  *****<br /><br />Temple of Heaven Gunpowder Gr...
306132  *****<br /><br />Temple of Heaven Gunpowder Gr...
68214   *****<br /><br />This Organic Green Tea Kombuc...
36692   *****<br /><br />This Organic Green Tea Kombuc...
204048  *****<br />Ashby's Japanese Green Tea is a dis...
61803   *****<br />Ashby's Japanese Green Tea is a dis...
524984  *****<br />Numi Tea's Chinese Breakfast Yunnan...
378979  *****<br />Numi Tea's Chinese Breakfast Yunnan...
438391  *****<br />Stash's Fusion Red & White Tea has ...
442055  *****<br />Stash's Fusion Red & White Tea has ...
410856  *****<br />Stash's Licorice Spice Caffeine Fre...
113563  *****<br />Stash's Licorice Spice Caffeine Fre...
360276  *****<br />Tazo's China Green Tips Green Tea i...
488311  *****<br />Tazo's China Green Tips Green Tea i...
367930  *****<br />This Fasting Tea from Yogi Tea for ...
367928  *****<br />This Fasting Tea from Yogi Tea for ...
227146  *****<br />This Fasting Tea from Yogi Tea for ...
443856  *****<br />White Rose Velvet Garden White Tea ...
...                                                  ...
484592  When the girl scouts are gone and your thin mi...
156517  When the girl scouts are gone and your thin mi...
59565   While you can make your own chai tea, it does ...
401926  While you can make your own chai tea, it does ...
146793  Whoever came up with this is a genius and a li...
357423  Whoever came up with this is a genius and a li...
503424  Why?  Because the noodles come SOFT, sealed in...
246458  Why?  Because the noodles come SOFT, sealed in...
505442  Why?  Because the noodles come SOFT, sealed in...
200791  Why?  Because the noodles come SOFT, sealed in...
314550  With five cats, buying premium cat food can ge...
140727  With five cats, buying premium cat food can ge...
469169  Yep I used to get them for free. All you had t...
35905   Yep I used to get them for free. All you had t...
479858  Yes, dogs love these treats but educate yourse...
514622  Yes, dogs love these treats but educate yourse...
```

```
138146  You can't find these mints in the stores, and ...
447742  You can't find these mints in the stores, and ...
485560  You get a better deal for these if you order a...
38078   You get a better deal for these if you order a...
54700   i tried bahlsen chocolates in europe, but you ...
151003  i tried bahlsen chocolates in europe, but you ...
37746   lol, I read that this item was cheaper in stor...
118574  lol, I read that this item was cheaper in stor...
76868   they are not contained in the keurig plastic c...
167105  they are not contained in the keurig plastic c...
242532  thought i was getting a great deal - wrong.  o...
99008   thought i was getting a great deal - wrong.  o...
521495  yummy great cookie just like my momma makes th...
521496  yummy great cookie just like my momma makes th...

[630 rows x 10 columns]
```

In [0]:
```python
#removing duplicate reviews
data=data.drop_duplicates(subset={"Text"}, keep='first', inplace=False)
data.shape
```

Out[0]: (363836, 10)

3.4.1 Observations

1. There are reviews which are same on similar products (mostly different flavors).
2. These reviews were posted with different timestamps by the same person (weird).
3. Since we are interested in a review being positive or negative, having redundant reviews makes no sense, so removing them.

3.5 [2.5] Data cleaning: Removing practically impossible data

In [0]:
```python
#also removing those reviews where HelpfulnessNumerator is greater than HelpfulnessDen
data=data[data['HelpfulnessNumerator']<=data['HelpfulnessDenominator']]
data.shape
```

Out[0]: (363834, 10)

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

In [0]:
```python
actualScore = data['Score']
positiveNegative = actualScore.map(partition)
data['Score'] = positiveNegative
print("Negatives shape:", data[data['Score']=='negative'].shape)
print("Positives shape:", data[data['Score']=='positive'].shape)
```

```
Negatives shape: (57070, 10)
Positives shape: (306764, 10)
```

4 [3] Text Preprocessing

We will be doing the following in order.

1. Text cleaning - includes removal of special characters which are not required.
2. Check if the word is actually an English word.
3. 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)
4. Convert the word to lower case.
5. Remove stop words but let's keep words like 'not' which makes the sentence negative.
6. POS Tagging and WordNet Lemmatizing the word.

```
In [0]: def cleanhtml(sentence): #function to clean the word of any html-tags
            cleanr = re.compile('<.*?>')
            cleantext = re.sub(cleanr, ' ', sentence)
            return cleantext
        def cleanpunc(sentence): #function to clean the word of any punctuation or special cha
            cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
            cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
            return  cleaned
```

```
In [0]: stop = list(set(stopwords.words('english'))) #set of stopwords
        print(stop)

        #removing words like 'not' that gives negative meaning to a sentence from stopwords
        important_stopwords = ['hadn', 'weren', 'shouldn', "needn't", 'needn', 'doesn', "shan'
                               'wouldn', "weren't", "didn", "mustn't", "wasn't", "didn't", "don
        pre_final_stops = [x for x in stop if x not in important_stopwords]

        #removing punctuation from stop words
        final_stops = list(set([cleanpunc(x) for x in pre_final_stops]))
        print("Final stopwords:", final_stops)
```

```
["it's", 'more', 'just', 'couldn', 'and', 'our', "hasn't", 'this', 've', 'off', 'needn', 'thems
Final stopwords: ['more', 'just', 'couldn', 'and', 'havent', 'our', 've', 'this', 'off', 'thems
```

```
In [0]: wnl = WordNetLemmatizer()
```

```
In [0]: #Code for implementing step-by-step the checks mentioned in the pre-processing phase
        # this code takes a while to run as it needs to run on 500k sentences.
        i=0
        str1=' '
        final_string=[]
```

```
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
scores = data['Score'].values
for sent in data['Text'].values:
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTMl tags
    tokens = pos_tag(word_tokenize(sent))
    for w in tokens:
        for cleaned_words in cleanpunc(w[0]).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    #s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    # lemmatization works better with POS tagging
                    tag = w[1][0].lower()
                    tag = tag if tag in ['a', 'n', 'v'] else None
                    if not tag:
                        s = cleaned_words.lower().encode('utf8')
                    else:
                        s = wnl.lemmatize(cleaned_words.lower(), tag).lower().encode("u
                    filtered_sentence.append(s)
                    if scores[i] == "positive":
                        all_positive_words.append(s) #list of all words used to descri
                    if scores[i] == "negative":
                        all_negative_words.append(s) #list of all words used to descri
                else:
                    continue
            else:
                continue
    #print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("***********************************************************************")

    final_string.append(str1)
    i+=1
print("Done!")
```

```
Done!
```

In [0]:
```
data['CleanedText']=final_string #adding a column of CleanedText which displays the da
data['CleanedText']=data['CleanedText'].str.decode("utf-8")
```

In [0]:
```
# store final table into an SQlLite table for future.
conn = sqlite3.connect('/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_revi
c=conn.cursor()
conn.text_factory = str
data.to_sql('Reviews', conn,  schema=None, if_exists='replace', index=True, index_label
```

In [0]:
```
##################################################################################
##################################################################################
con = sqlite3.connect('/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_revie
data = pd.read_sql_query(""" SELECT * FROM Reviews """, con)
del data['index']
data.shape
```

Out[0]:
```
(363834, 11)
```

In [0]:
```
data.head()
```

Out[0]:
```
       Id   ProductId          UserId                   ProfileName  \
0  150524  0006641040   ACITT7DI6IDDL               shari zychinski
1  150506  0006641040  A2IW4PEEKO2R0U                         Tracy
2  150507  0006641040  A1S4A3IQ2MU7V4         sally sue "sally sue"
3  150508  0006641040     AZGXZ2UUK6X  Catherine Hallberg "(Kate)"
4  150509  0006641040  A3CMRKGEOP909G                        Teresa

   HelpfulnessNumerator  HelpfulnessDenominator     Score        Time  \
0                     0                       0  positive   939340800
1                     1                       1  positive  1194739200
2                     1                       1  positive  1191456000
3                     1                       1  positive  1076025600
4                     3                       4  positive  1018396800

                                       Summary  \
0                    EVERY book is educational
1  Love the book, miss the hard cover version
2                chicken soup with rice months
3      a good swingy rhythm for reading aloud
4             A great way to learn the months

                                                Text  \
0  this witty little book makes my son laugh at l...
1  I grew up reading these Sendak books, and watc...
2  This is a fun way for children to learn their ...
3  This is a great little book to read aloud- it ...
4  This is a book of poetry about the months of t...

                                         CleanedText
0  witty little book make son laugh loud recite c...
1  grow read sendak book watch really rosie movie...
2  fun way child learn month year learn poem thro...
3  great little book read nice rhythm well good r...
4  book poetry month year go month cute little po...
```

In [0]:
```
# positive reviews
pos_df = data[data['Score'] == 'positive'].copy()
pos_df['Time'] = pos_df['Time'].astype('int')
```

```
# sorting it based on time so that we can split based on time
pos_df.sort_values('Time', axis=0, ascending=True, inplace=False, kind='quicksort', na
pos_df.shape
```

Out[0]: (306764, 11)

In [0]:
```
#negative reviews
neg_df = data[data['Score'] == 'negative'].copy()
neg_df['Time'] = neg_df['Time'].astype('int')
neg_df.sort_values('Time', axis=0, ascending=True, inplace=False, kind='quicksort', na
neg_df.shape
```

Out[0]: (57070, 11)

In [0]:
```
pos_50k = pos_df.head(10000).copy()
neg_50k = neg_df.head(10000).copy()
```

In [0]:
```
pos_50k = pos_df.head(50000).copy()
neg_50k = neg_df.head(50000).copy()
```

In [0]:
```
# training data 60%
pos_train = pos_50k.head(6000).copy()
neg_train = neg_50k.head(6000).copy()

# cross validation data 20%
pos_cv = pos_50k[6000:8000].copy()
neg_cv = neg_50k[6000:8000].copy()

# test data 20%
pos_test = pos_50k[8000:].copy()
neg_test = neg_50k[8000:].copy()
```

In [0]:
```
# training data 60%
pos_train = pos_50k.head(30000).copy()
neg_train = neg_50k.head(30000).copy()

# cross validation data 20%
pos_cv = pos_50k[30000:40000].copy()
neg_cv = neg_50k[30000:40000].copy()

# test data 20%
pos_test = pos_50k[40000:].copy()
neg_test = neg_50k[40000:].copy()
```

In [0]:
```
train_df = pos_train.append(neg_train, ignore_index=True).copy()
cv_df = pos_cv.append(neg_cv, ignore_index=True).copy()
test_df = pos_test.append(neg_test, ignore_index=True).copy()
train_df.shape
```

Out[0]: (60000, 11)

5 [4] Featurization

5.1 [4.1] Bag of words - unigrams and bigrams

```
In [0]: #BoW
        count_vect = CountVectorizer() #in scikit-learn
        train_final_counts = count_vect.fit_transform(train_df['CleanedText'].values)
        cv_final_counts = count_vect.transform(cv_df['CleanedText'].values)
        test_final_counts = count_vect.transform(test_df['CleanedText'].values)
        print("the type of count vectorizer ",type(train_final_counts))
        print("the shape of out text BOW vectorizer ",train_final_counts.get_shape())
        print("the number of unique words ", train_final_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (60000, 40286)
the number of unique words  40286
```

```
In [0]: freq_dist_positive=nltk.FreqDist(all_positive_words)
        freq_dist_negative=nltk.FreqDist(all_negative_words)
        print("Most Common Positive Words : ",freq_dist_positive.most_common(20))
        print("Most Common Negative Words : ",freq_dist_negative.most_common(20))
```

```
Most Common Positive Words :  [(b'like', 137224), (b'taste', 122045), (b'good', 111390), (b'lo
Most Common Negative Words :  [(b'taste', 33523), (b'like', 31734), (b'product', 28122), (b'bu
```

```
In [0]: #saving BoW unigrams
        with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_uni_
            pickle.dump(train_final_counts, bow)
        with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/train_la
            pickle.dump(train_df['Score'].values, bow)

        with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_uni_
            pickle.dump(cv_final_counts, bow)
        with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/cv_lab.p
            pickle.dump(cv_df['Score'].values, bow)

        with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_uni_
            pickle.dump(test_final_counts, bow)
        with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/test_lab
            pickle.dump(test_df['Score'].values, bow)
```

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

        #removing stop words like "not" should be avoided before building n-grams
        count_vect = CountVectorizer(ngram_range=(1,2) ) #in scikit-learn
        train_bigram_counts = count_vect.fit_transform(train_df['CleanedText'].values)
        cv_bigram_counts = count_vect.transform(cv_df['CleanedText'].values)
```

```
        test_bigram_counts = count_vect.transform(test_df['CleanedText'].values)
        print("the type of count vectorizer ",type(train_bigram_counts))
        print("the shape of out text BOW vectorizer ",train_bigram_counts.get_shape())
        print("the number of unique words including both unigrams and bigrams ", train_bigram_c
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (60000, 977013)
the number of unique words including both unigrams and bigrams  977013
```

```
In [0]: #saving BoW bigrams
        with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_ve
            pickle.dump(train_bigram_counts, bow)
        # with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_
        #     pickle.dump(train_df['Score'].values, bow)

        with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_ve
            pickle.dump(cv_bigram_counts, bow)
        # with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_
        #     pickle.dump(cv_df['Score'].values, bow)

        with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_ve
            pickle.dump(test_bigram_counts, bow)
        # with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_
        #     pickle.dump(test_df['Score'].values, bow)
```

5.2 [4.2] TF-IDF

```
In [0]: #tf-idf
        tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
        train_tf_idf = tf_idf_vect.fit_transform(train_df['CleanedText'].values)
        cv_tfidf = tf_idf_vect.transform(cv_df['CleanedText'].values)
        test_tfidf = tf_idf_vect.transform(test_df['CleanedText'].values)
        print("the type of count vectorizer ",type(train_tf_idf))
        print("the shape of out text TFIDF vectorizer ",train_tf_idf.get_shape())
        print("the number of unique words including both unigrams and bigrams ", train_tf_idf.g
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (60000, 977013)
the number of unique words including both unigrams and bigrams  977013
```

```
In [0]: features = tf_idf_vect.get_feature_names()
        print("some sample features(unique words in the corpus)",features[100000:100010])
```

```
some sample features(unique words in the corpus) ['bpa originally', 'bpa packaging', 'bpa perso
```

```
In [0]: def top_tfidf_feats(row, features, top_n=25):
            ''' Get top n tfidf values in row and return them with their corresponding feature
```

```
    topn_ids = np.argsort(row)[::-1][:top_n]
    top_feats = [(features[i], row[i]) for i in topn_ids]
    df = pd.DataFrame(top_feats)
    df.columns = ['feature', 'tfidf']
    return df

top_tfidf = top_tfidf_feats(train_tf_idf[1,:].toarray()[0],features,25)
```

In [0]:
```
top_tfidf
```

Out[0]:

	feature	tfidf
0	paperback seem	0.182072
1	rosie movie	0.182072
2	incorporate love	0.182072
3	version paperback	0.182072
4	cover version	0.182072
5	page open	0.182072
6	keep page	0.182072
7	read sendak	0.182072
8	movie incorporate	0.182072
9	hard cover	0.175544
10	miss hard	0.175544
11	sendak book	0.175544
12	grow read	0.175544
13	kind flimsy	0.175544
14	really rosie	0.175544
15	watch really	0.175544
16	flimsy take	0.175544
17	however miss	0.175544
18	book watch	0.175544
19	two hand	0.175544
20	love son	0.167320
21	rosie	0.164385
22	paperback	0.164385
23	seem kind	0.161903
24	hand keep	0.157857

In [0]:
```
#saving tfidf
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_ve
    pickle.dump(train_tf_idf, bow)
# with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_
#     pickle.dump(train_df['Score'].values, bow)

with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_ve
    pickle.dump(cv_tfidf, bow)
# with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_
#     pickle.dump(cv_df['Score'].values, bow)
```

```
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_vec
    pickle.dump(test_tfidf, bow)
# with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_t
#     pickle.dump(test_df['Score'].values, bow)
```

```
In [0]: list_of_sent=[]
        for sent in train_df['CleanedText'].values:
            list_of_sent.append(sent.split())
```

```
In [0]: list_of_sent=[]
        for sent in cv_df['CleanedText'].values:
            list_of_sent.append(sent.split())
```

```
In [0]: list_of_sent=[]
        for sent in test_df['CleanedText'].values:
            list_of_sent.append(sent.split())
```

5.3 [4.3] Word2Vec

```
In [0]: w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=7)
```

```
In [0]: #saving w2v model
        w2v_model.save("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/amzn
```

```
In [0]: #loading model
        w2v_model = Word2Vec.load("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_r
```

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

```
number of words that occured minimum 5 times  12979
sample words  ['witty', 'little', 'book', 'make', 'son', 'laugh', 'loud', 'recite', 'car', 'dri
```

5.4 [4.3.1] Average Word2Vec

```
In [0]: # average Word2Vec
        # compute average word2vec for each review.
        def avg_w2vec(list_of_sent):
            sent_vectors = [] # the avg-w2v for each sentence/review is stored in this list
            for sent in list_of_sent: # for each review/sentence
                sent_vec = np.zeros(50) # as word vectors are of zero length
                cnt_words =0 # num of words with a valid vector in the sentence/review
                for word in sent: # for each word in a review/sentence
                    if word in w2v_words:
                        vec = w2v_model.wv[word]
                        sent_vec += vec
                        cnt_words += 1
```

```
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors.append(sent_vec)
    print(len(sent_vectors))
    print(len(sent_vectors[0]))
    return sent_vectors
```

In [0]:
```
avg_w2v_train = avg_w2vec([sent.split() for sent in train_df['CleanedText'].values])
avg_w2v_cv = avg_w2vec([sent.split() for sent in cv_df['CleanedText'].values])
avg_w2v_test = avg_w2vec([sent.split() for sent in test_df['CleanedText'].values])
```

```
60000
50
20000
50
20000
50
```

In [0]:
```
#saving word2vec
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/avg_w2v_
    pickle.dump(avg_w2v_train, w2v_pickle)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/avg_w2v_
    pickle.dump(avg_w2v_cv, w2v_pickle)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/avg_w2v_
    pickle.dump(avg_w2v_test, w2v_pickle)
```

5.5 [4.3.2] TFIDF-Word2Vec

In [0]:
```
def helper(list_of_sent, final_tf_idf):
    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in thi
    row=0;
    for sent in tqdm(list_of_sent): # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length
        weight_sum =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_words:
                vec = w2v_model.wv[word]
                # obtain the tf_idfidf of a word in a sentence/review
                tf_idf = final_tf_idf[row, tfidf_feat.index(word)]
                sent_vec += (vec * tf_idf)
                weight_sum += tf_idf
        if weight_sum != 0:
            sent_vec /= weight_sum
        tfidf_sent_vectors.append(sent_vec)
        row += 1
        #print(row, end=" ")
    return tfidf_sent_vectors
```

In [0]:
```
from tqdm import tqdm
```

In [0]:
```
helper_numba = jit()(helper)
```

In [0]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(train_df['CleanedText'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [0]:
```
# TF-IDF weighted Word2Vec
def tfidf_w2vec(list_of_sent):
    tfidf_feat = model.get_feature_names() # tfidf words/col-names
    # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tf

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

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

#tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this l
#row=0;
#for sent in list_of_sent: # for each review/sentence
#this was taking a lot of time

# with ThreadPoolExecutor(max_workers=10000) as executor:
#     result_futures = [executor.submit(helper_numba, sent=x, row=y) for y, x in enume
#     for f in futures.as_completed(result_futures):
#         i = f.result()
```

```
#         print(i)
# print("Threading done!")

# for y, x in enumerate(list_of_sent):
#     i = helper_numba(sent=x, row=y)
#     print(i)
list_of_sent = [sent.split() for sent in train_df['CleanedText'].values]
# train_tfidf_w2v = helper(list_of_sent, train_tf_idf)
train_tfidf_w2v = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sent): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = train_tf_idf[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    train_tfidf_w2v.append(sent_vec)
    row += 1
    #print(row, end=" ")

print("Done!")
```

```
  0%|          | 121/60000 [03:36<33:49:49,  2.03s/it]


    ---------------------------------------------------------------------------

    KeyboardInterrupt                         Traceback (most recent call last)

    <ipython-input-20-7d49c173ee40> in <module>()
     28             vec = w2v_model.wv[word]
     29             # obtain the tf_idfidf of a word in a sentence/review
---> 30             tf_idf = train_tf_idf[row, tfidf_feat.index(word)]
     31             sent_vec += (vec * tf_idf)
     32             weight_sum += tf_idf


    KeyboardInterrupt:
```

In [0]:
```
#saving tfidf weighted w2v
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_we
```

```
            pickle.dump(tfidf_sent_vectors, tfidf_w2v_pickle)

        print("Done!")
```

Done!

```
In [0]: with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_we
            pickle.dump(tfidf_sent_vectors, tfidf_w2v_pickle)
```

```
In [0]: with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_we
            pickle.dump(tfidf_sent_vectors, tfidf_w2v_pickle)
```

6 [5] KNN Assignment

6.1 [5.1] KNN Brute Force

6.1.1 [5.1.1] Bag of Words

```
In [0]: # loading the libraries
        from sklearn.neighbors import KNeighborsClassifier
        from tqdm import tqdm
        import matplotlib.pyplot as plt
```

```
In [0]: from sklearn.metrics import classification_report
```

```
In [0]: from sklearn.model_selection import GridSearchCV
```

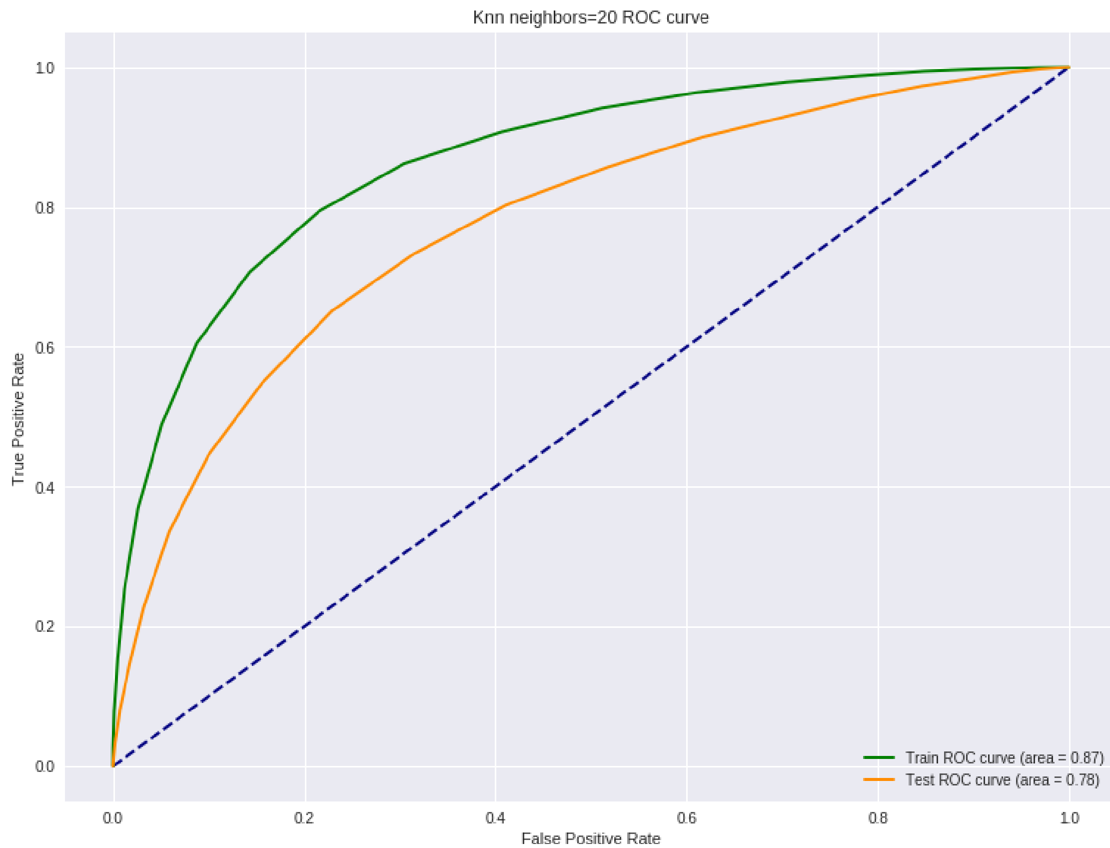
```
In [0]: from sklearn.metrics import roc_curve, auc
```

```
In [0]: # loading the pickle file

        with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_uni_
            bow_train = pickle.load(bow)
        with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/train_lal
            bow_train_lab = pickle.load(bow)

        with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_uni_
            bow_cv = pickle.load(bow)
        with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/cv_lab.pl
            bow_cv_lab = pickle.load(bow)

        with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_uni_
            bow_test = pickle.load(bow)
        with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/test_lab
            bow_test_lab = pickle.load(bow)
```

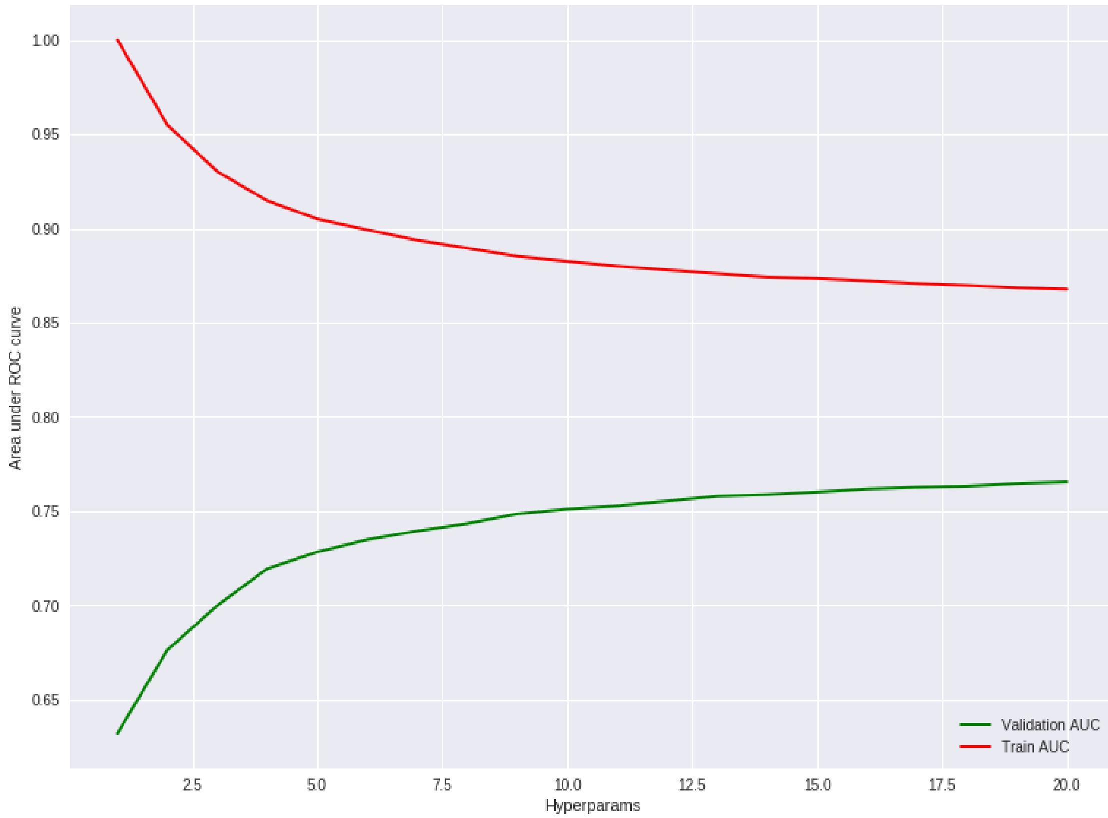
```
In [0]: train_lab_bin = [1 if x=='positive' else 0 for x in bow_train_lab]
        test_lab_bin = [1 if x=='positive' else 0 for x in bow_test_lab]
        cv_lab_bin = [1 if x=='positive' else 0 for x in bow_cv_lab]
```

In [0]:
```
# finding best k using AUC
lw = 2
auc_train = []
auc_cv = []
auc_test = []
fpr_train = dict()
tpr_train = dict()
fpr_test = dict()
tpr_test = dict()
fpr_cv = dict()
tpr_cv = dict()

bow_train_lab_bin = [1 if x=='positive' else 0 for x in bow_train_lab]
bow_test_lab_bin = [1 if x=='positive' else 0 for x in bow_test_lab]
bow_cv_lab_bin = [1 if x=='positive' else 0 for x in bow_cv_lab]

for idx, k in enumerate(range(1, 21)):
    knn_classifier = KNeighborsClassifier(n_neighbors=k, algorithm='brute')
    knn_classifier.fit(bow_train, bow_train_lab_bin)
    bow_train_proba = knn_classifier.predict_proba(bow_train)
    fpr_train[idx], tpr_train[idx], _ = roc_curve(bow_train_lab_bin, bow_train_proba[:
    auc_train.append(auc(fpr_train[idx], tpr_train[idx]))

    bow_test_proba = knn_classifier.predict_proba(bow_test)
    fpr_test[idx], tpr_test[idx], _ = roc_curve(bow_test_lab_bin, bow_test_proba[:,1])
    auc_test.append(auc(fpr_test[idx], tpr_test[idx]))

    bow_cv_proba = knn_classifier.predict_proba(bow_cv)
    fpr_cv[idx], tpr_cv[idx], _ = roc_curve(bow_cv_lab_bin, bow_cv_proba[:,1])
    auc_cv.append(auc(fpr_cv[idx], tpr_cv[idx]))
```

In [0]:
```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_cv.index(max(auc_cv))
plt.plot(fpr_train[max_idx], tpr_train[max_idx], color='green', lw=lw, label='Train RO
plt.plot(fpr_test[max_idx], tpr_test[max_idx], color='darkorange', lw=lw, label='Test R
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



In [0]:
```
# graph train auc, cv auc and hyper params
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p

plt.figure(figsize=(12.8, 9.6))
#plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_train.index(max(auc_train))
plt.plot(range(1, 21), auc_cv, color='green', lw=lw, label='Validation AUC')
plt.plot(range(1, 21), auc_train, color='red', lw=lw, label='Train AUC')
plt.xlabel('Hyperparams')
plt.ylabel('Area under ROC curve')
# plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



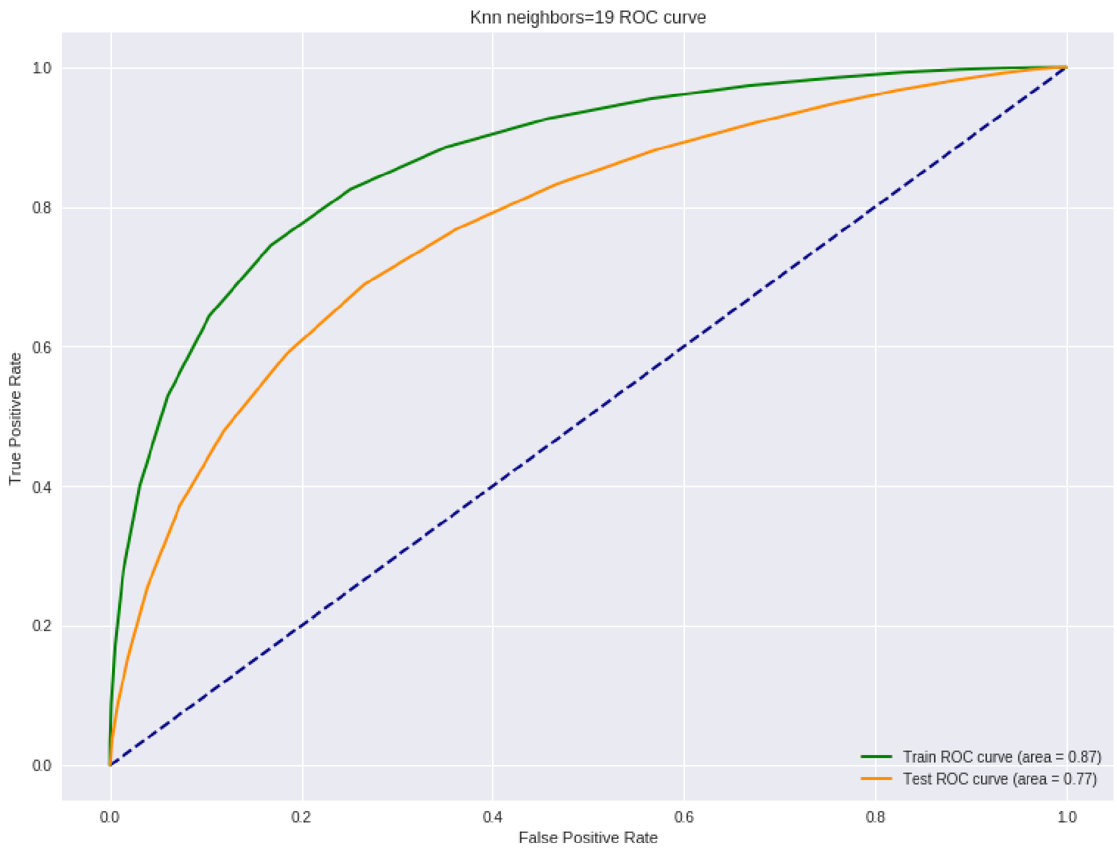

```
In [0]: params = {"n_neighbors": np.arange(1, 31, 2)}

        classifier = KNeighborsClassifier()
        grid = GridSearchCV(classifier, params, n_jobs=-1, verbose=2)
        grid.fit(bow_train, bow_train_lab_bin)
        acc = grid.score(bow_cv, bow_cv_lab_bin)

        print("CV Accuracy:", acc)
        print("Best Params", grid.best_params_)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 40.0min
[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed: 48.5min finished


CV Accuracy: 0.6905
Best Params {'n_neighbors': 19}
```

```
In [0]: # 19-NN
        knn_classifier = KNeighborsClassifier(n_neighbors=19, algorithm='brute')
        knn_classifier.fit(bow_train, bow_train_lab)
        bow_cv_predict = knn_classifier.predict(bow_cv)
        print(classification_report(bow_cv_lab, bow_cv_predict))
        train_proba = knn_classifier.predict_proba(bow_train)
        fpr_train, tpr_train, _ = roc_curve(train_lab_bin, train_proba[:,1])
        auc_train = auc(fpr_train, tpr_train)

        test_proba = knn_classifier.predict_proba(bow_test)
        fpr_test, tpr_test, _ = roc_curve(test_lab_bin, test_proba[:,1])
        auc_test = auc(fpr_test, tpr_test)
```

```
              precision    recall  f1-score   support

    negative       0.73      0.60      0.66     10000
    positive       0.66      0.78      0.72     10000

   micro avg       0.69      0.69      0.69     20000
   macro avg       0.70      0.69      0.69     20000
weighted avg       0.70      0.69      0.69     20000
```

```
In [0]: lw=2
        # plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p
        plt.figure(figsize=(12.8, 9.6))
        plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
        # max_idx = auc_cv.index(max(auc_cv))
        plt.plot(fpr_train, tpr_train, color='green', lw=lw, label='Train ROC curve (area = %0
        plt.plot(fpr_test, tpr_test, color='darkorange', lw=lw, label='Test ROC curve (area = %
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Knn neighbors=' + str(19) + ' ROC curve')
        plt.legend(loc="lower right")
        plt.show()
```



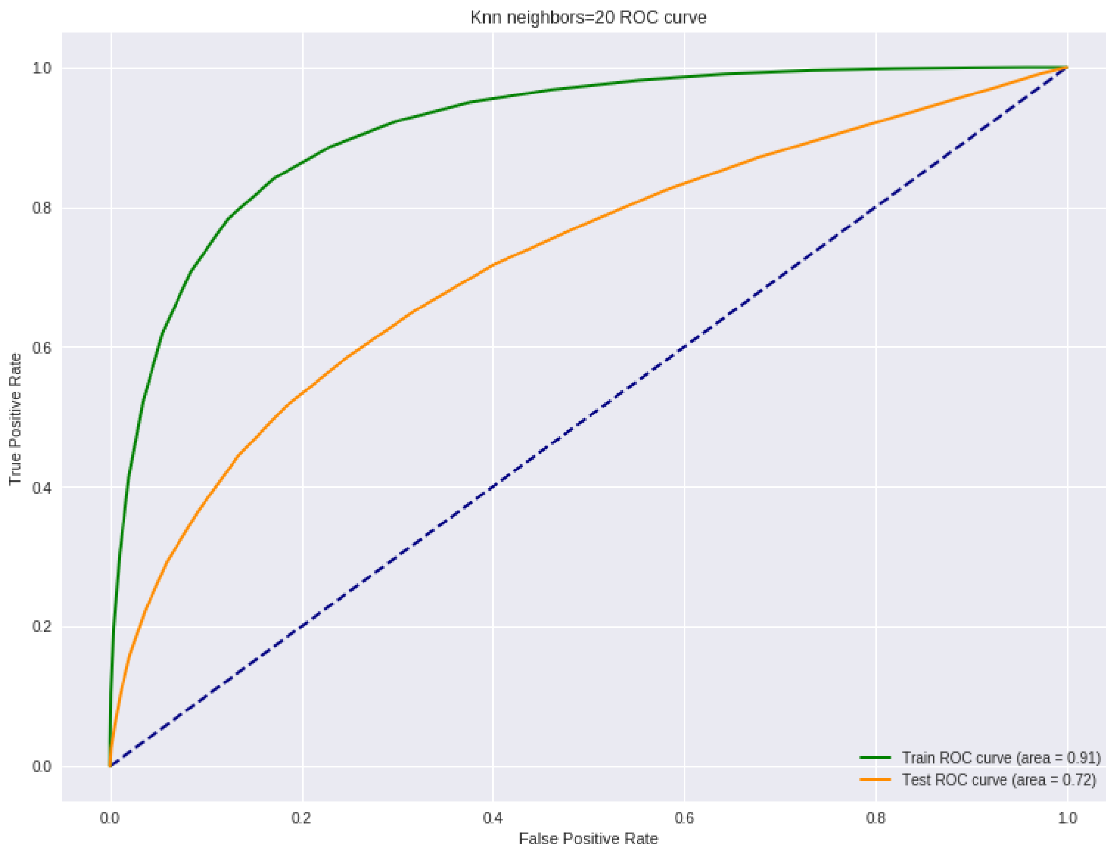
6.1.2 [5.1.2] TFIDF

In [0]:
```
# loading tfidf vectors
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_ve
    train_data = pickle.load(bow)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_ve
    cv_data = pickle.load(bow)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_ve
    test_data = pickle.load(bow)
```

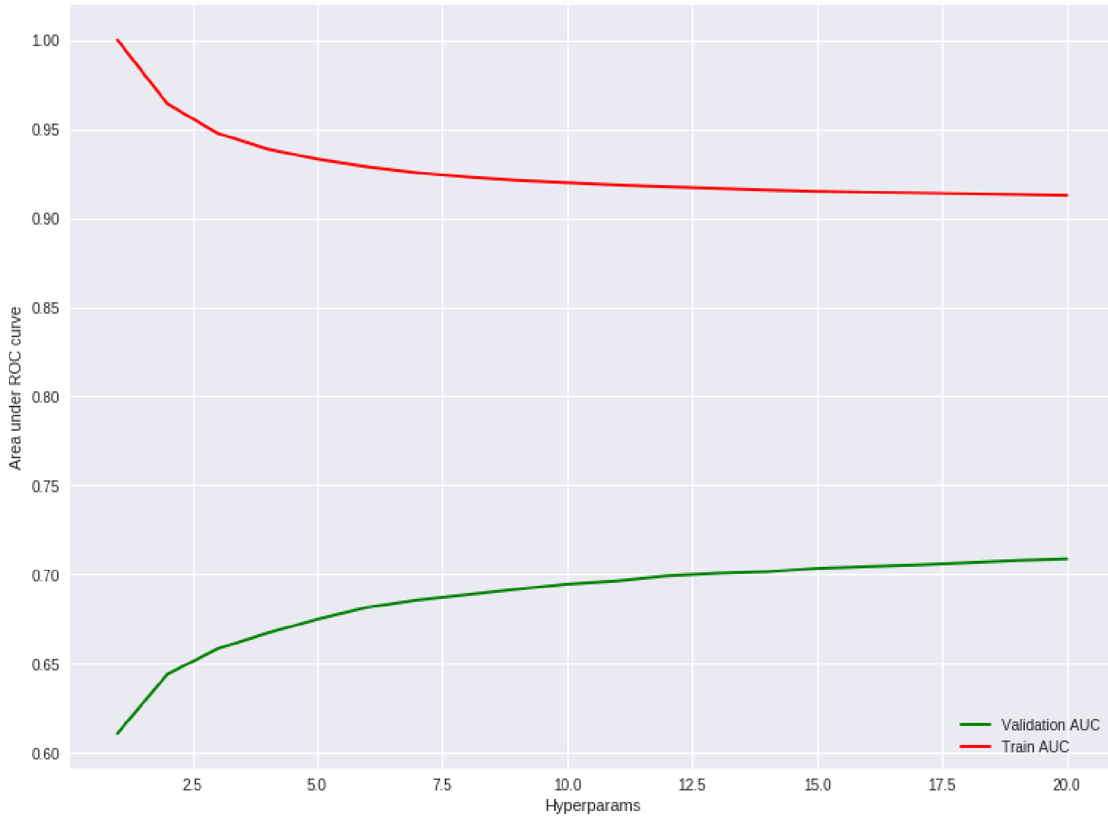
In [0]:
```
# finding best k using AUC
lw = 2
auc_train = []
auc_cv = []
auc_test = []
fpr_train = dict()
tpr_train = dict()
fpr_test = dict()
tpr_test = dict()
fpr_cv = dict()
tpr_cv = dict()
```

```
train_lab_bin = [1 if x=='positive' else 0 for x in bow_train_lab]
test_lab_bin = [1 if x=='positive' else 0 for x in bow_test_lab]
cv_lab_bin = [1 if x=='positive' else 0 for x in bow_cv_lab]

for idx, k in enumerate(range(1, 21)):
    knn_classifier = KNeighborsClassifier(n_neighbors=k, algorithm='brute')
    knn_classifier.fit(train_data, train_lab_bin)
    train_proba = knn_classifier.predict_proba(train_data)
    fpr_train[idx], tpr_train[idx], _ = roc_curve(train_lab_bin, train_proba[:,1])
    auc_train.append(auc(fpr_train[idx], tpr_train[idx]))

    test_proba = knn_classifier.predict_proba(test_data)
    fpr_test[idx], tpr_test[idx], _ = roc_curve(test_lab_bin, test_proba[:,1])
    auc_test.append(auc(fpr_test[idx], tpr_test[idx]))

    cv_proba = knn_classifier.predict_proba(cv_data)
    fpr_cv[idx], tpr_cv[idx], _ = roc_curve(cv_lab_bin, cv_proba[:,1])
    auc_cv.append(auc(fpr_cv[idx], tpr_cv[idx]))
```

In [0]:
```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_cv.index(max(auc_cv))
plt.plot(fpr_train[max_idx], tpr_train[max_idx], color='green', lw=lw, label='Train ROC
plt.plot(fpr_test[max_idx], tpr_test[max_idx], color='darkorange', lw=lw, label='Test R
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



In [0]:
```
# graph train auc, cv auc and hyper params
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p

plt.figure(figsize=(12.8, 9.6))
#plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_train.index(max(auc_train))
plt.plot(range(1, 21), auc_cv, color='green', lw=lw, label='Validation AUC')
plt.plot(range(1, 21), auc_train, color='red', lw=lw, label='Train AUC')
plt.xlabel('Hyperparams')
plt.ylabel('Area under ROC curve')
# plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



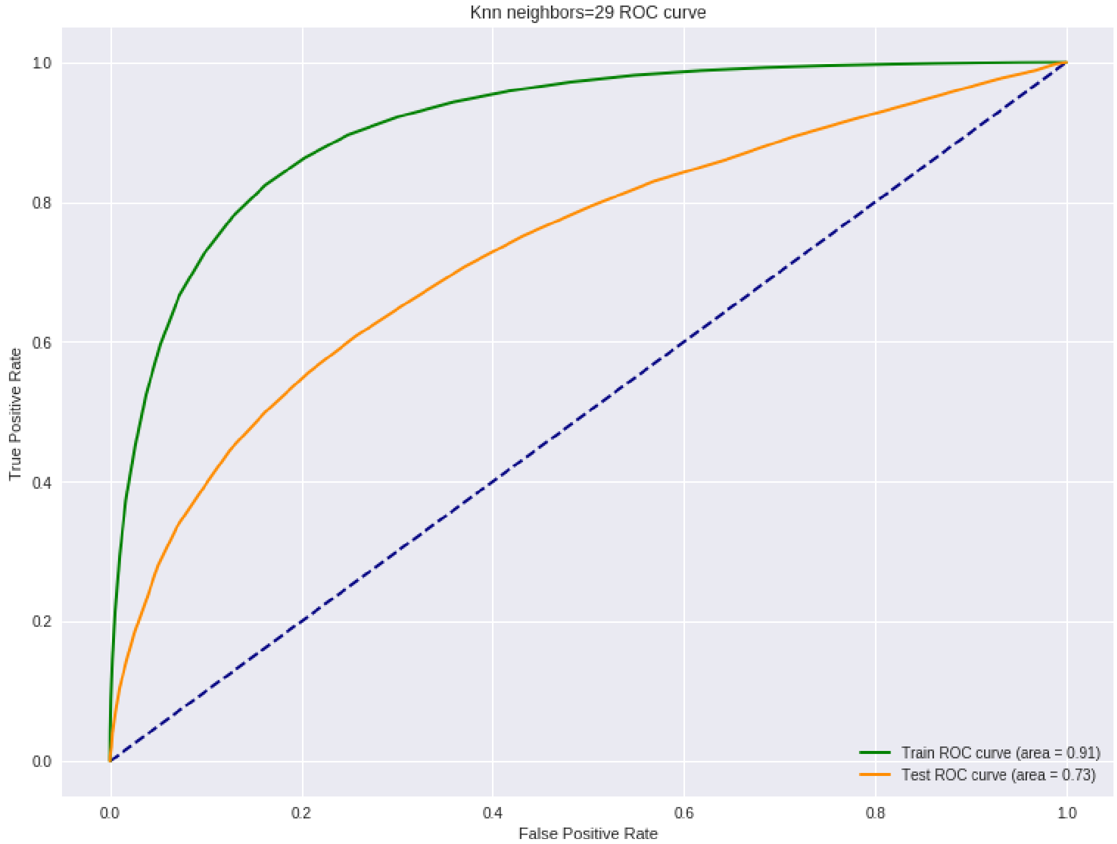

```
In [0]: params = {"n_neighbors": np.arange(1, 31, 2)}

        classifier = KNeighborsClassifier()
        grid = GridSearchCV(classifier, params, n_jobs=-1, verbose=2)
        grid.fit(train_data, train_lab_bin)
        acc = grid.score(cv_data, cv_lab_bin)

        print("CV Accuracy:", acc)
        print("Best Params", grid.best_params_)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 46.4min
[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed: 56.0min finished


CV Accuracy: 0.6638
Best Params {'n_neighbors': 29}
```

```
In [0]: # 29-NN
        knn_classifier = KNeighborsClassifier(n_neighbors=29, algorithm='brute')
        knn_classifier.fit(train_data, bow_train_lab)
        cv_predict = knn_classifier.predict(cv_data)
        print(classification_report(bow_cv_lab, cv_predict))
        train_proba = knn_classifier.predict_proba(train_data)
        fpr_train, tpr_train, _ = roc_curve(train_lab_bin, train_proba[:,1])
        auc_train = auc(fpr_train, tpr_train)

        test_proba = knn_classifier.predict_proba(test_data)
        fpr_test, tpr_test, _ = roc_curve(test_lab_bin, test_proba[:,1])
        auc_test = auc(fpr_test, tpr_test)
```

```
              precision    recall  f1-score   support

    negative       0.63      0.77      0.70     10000
    positive       0.71      0.56      0.62     10000

   micro avg       0.66      0.66      0.66     20000
   macro avg       0.67      0.66      0.66     20000
weighted avg       0.67      0.66      0.66     20000
```

```
In [0]: lw=2
        # plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p
        plt.figure(figsize=(12.8, 9.6))
        plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
        #max_idx = auc_cv.index(max(auc_cv))
        plt.plot(fpr_train, tpr_train, color='green', lw=lw, label='Train ROC curve (area = %0
        plt.plot(fpr_test, tpr_test, color='darkorange', lw=lw, label='Test ROC curve (area = %
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Knn neighbors=' + str(29) + ' ROC curve')
        plt.legend(loc="lower right")
        plt.show()
```



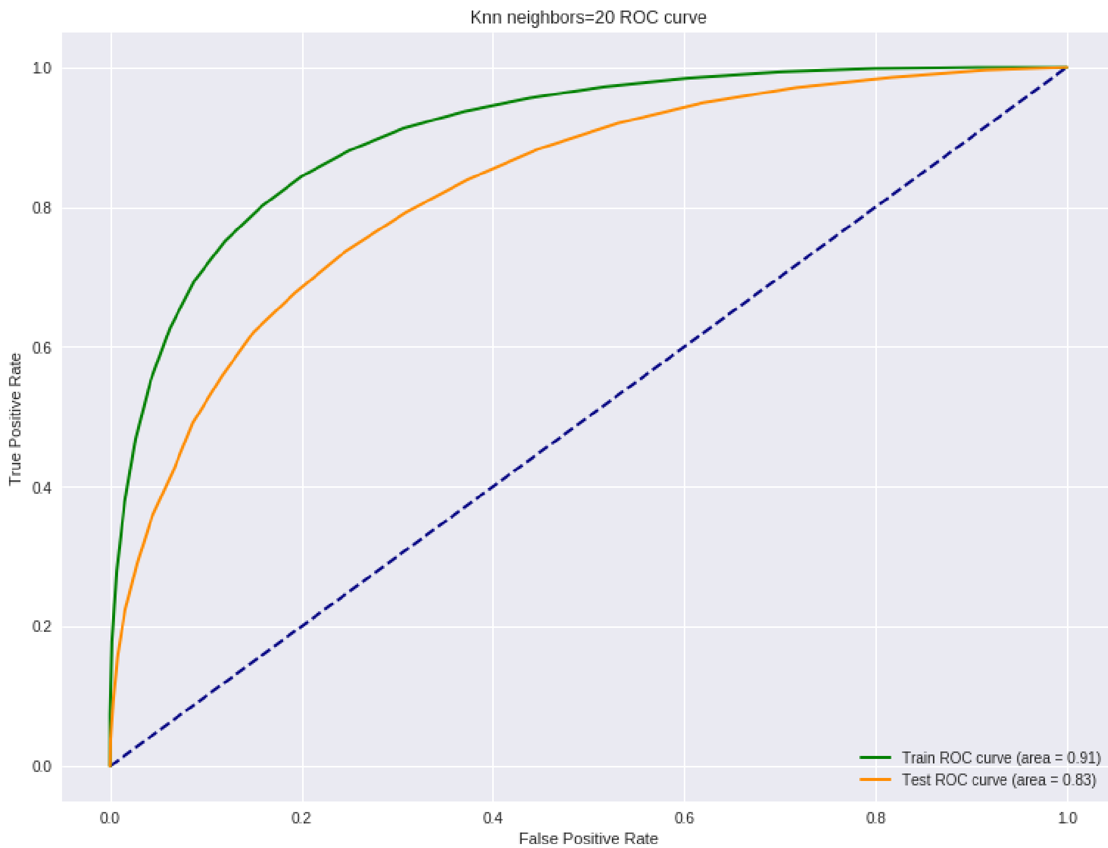
6.1.3 [5.1.3] Word2Vec

In [0]:
```
#loading word2vec vectors
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/avg_w2v_t
    train_data = pickle.load(w2v_pickle)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/avg_w2v_c
    cv_data = pickle.load(w2v_pickle)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/avg_w2v_t
    test_data = pickle.load(w2v_pickle)
```

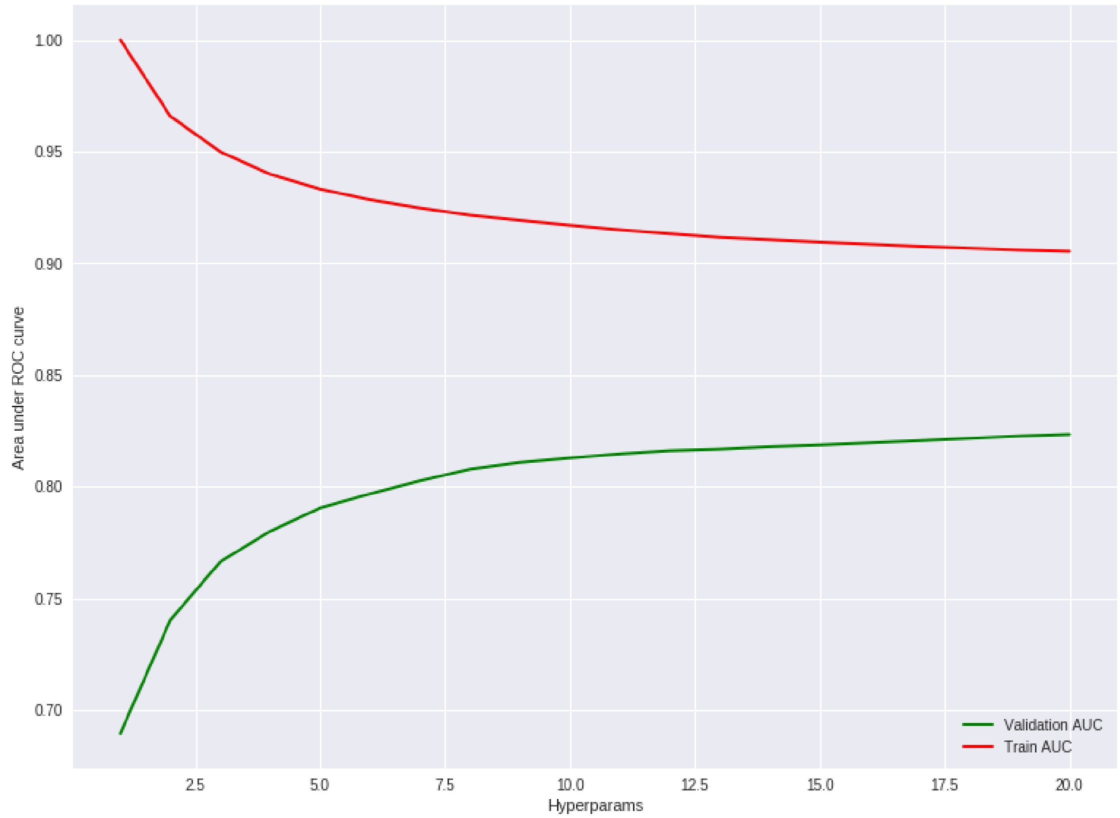
In [0]:
```
# finding best k using AUC
lw = 2
auc_train = []
auc_cv = []
auc_test = []
fpr_train = dict()
tpr_train = dict()
fpr_test = dict()
tpr_test = dict()
fpr_cv = dict()
tpr_cv = dict()
```

```
train_lab_bin = [1 if x=='positive' else 0 for x in bow_train_lab]
test_lab_bin = [1 if x=='positive' else 0 for x in bow_test_lab]
cv_lab_bin = [1 if x=='positive' else 0 for x in bow_cv_lab]

for idx, k in enumerate(range(1, 21)):
    knn_classifier = KNeighborsClassifier(n_neighbors=k, algorithm='brute')
    knn_classifier.fit(train_data, train_lab_bin)
    train_proba = knn_classifier.predict_proba(train_data)
    fpr_train[idx], tpr_train[idx], _ = roc_curve(train_lab_bin, train_proba[:,1])
    auc_train.append(auc(fpr_train[idx], tpr_train[idx]))

    test_proba = knn_classifier.predict_proba(test_data)
    fpr_test[idx], tpr_test[idx], _ = roc_curve(test_lab_bin, test_proba[:,1])
    auc_test.append(auc(fpr_test[idx], tpr_test[idx]))

    cv_proba = knn_classifier.predict_proba(cv_data)
    fpr_cv[idx], tpr_cv[idx], _ = roc_curve(cv_lab_bin, cv_proba[:,1])
    auc_cv.append(auc(fpr_cv[idx], tpr_cv[idx]))
```

In [0]:
```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_cv.index(max(auc_cv))
plt.plot(fpr_train[max_idx], tpr_train[max_idx], color='green', lw=lw, label='Train ROC
plt.plot(fpr_test[max_idx], tpr_test[max_idx], color='darkorange', lw=lw, label='Test R
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



In [0]:
```
# graph train auc, cv auc and hyper params
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p

plt.figure(figsize=(12.8, 9.6))
#plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_train.index(max(auc_train))
plt.plot(range(1, 21), auc_cv, color='green', lw=lw, label='Validation AUC')
plt.plot(range(1, 21), auc_train, color='red', lw=lw, label='Train AUC')
plt.xlabel('Hyperparams')
plt.ylabel('Area under ROC curve')
# plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



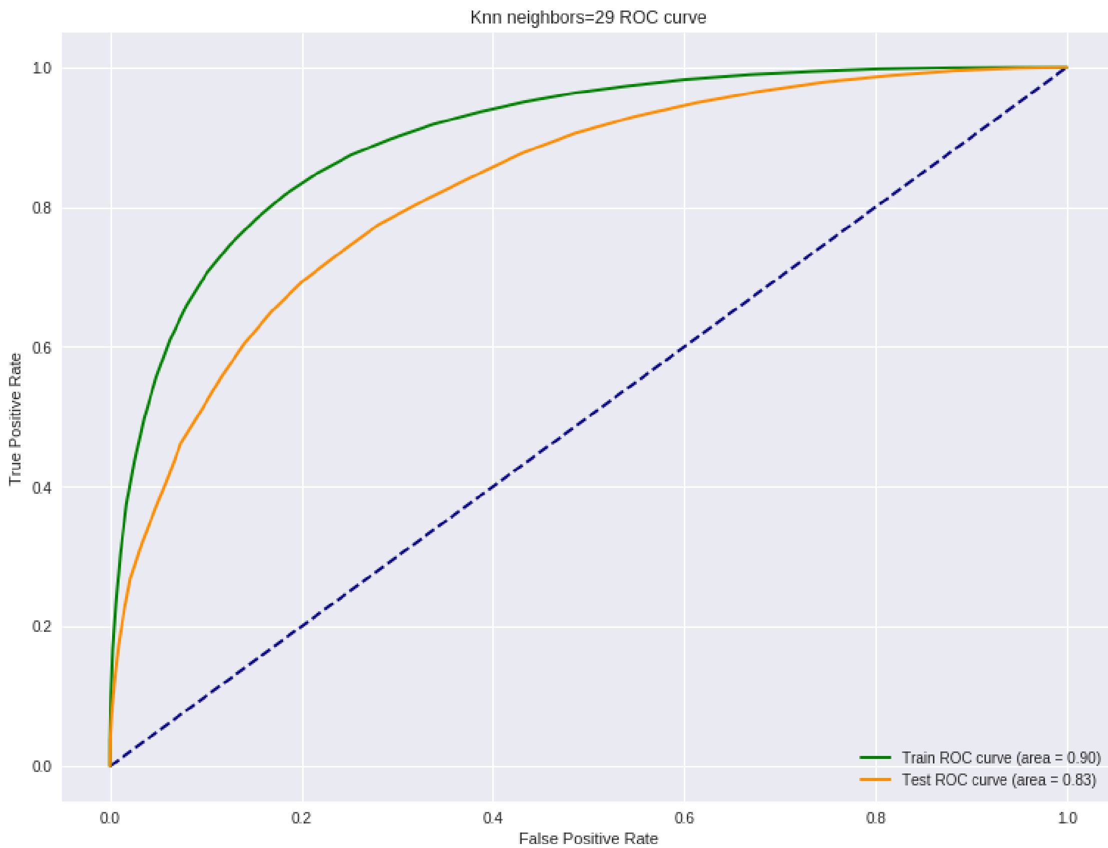

```
In [0]: params = {"n_neighbors": np.arange(1, 31, 2)}

        classifier = KNeighborsClassifier()
        grid = GridSearchCV(classifier, params, n_jobs=-1, verbose=2)
        grid.fit(train_data, train_lab_bin)
        acc = grid.score(cv_data, cv_lab_bin)

        print("CV Accuracy:", acc)
        print("Best Params", grid.best_params_)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 213.4min
[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed: 262.4min finished


CV Accuracy: 0.74375
Best Params {'n_neighbors': 29}
```

In [0]:
```python
# 29-NN
knn_classifier = KNeighborsClassifier(n_neighbors=29, algorithm='brute')
knn_classifier.fit(train_data, bow_train_lab)
cv_predict = knn_classifier.predict(cv_data)
print(classification_report(bow_cv_lab, cv_predict))
train_proba = knn_classifier.predict_proba(train_data)
fpr_train, tpr_train, _ = roc_curve(train_lab_bin, train_proba[:,1])
auc_train = auc(fpr_train, tpr_train)

test_proba = knn_classifier.predict_proba(test_data)
fpr_test, tpr_test, _ = roc_curve(test_lab_bin, test_proba[:,1])
auc_test = auc(fpr_test, tpr_test)
```

```
              precision    recall  f1-score   support

    negative       0.71      0.82      0.76     10000
    positive       0.78      0.67      0.72     10000

   micro avg       0.74      0.74      0.74     20000
   macro avg       0.75      0.74      0.74     20000
weighted avg       0.75      0.74      0.74     20000
```

In [0]:
```python
lw=2
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
#max_idx = auc_cv.index(max(auc_cv))
plt.plot(fpr_train, tpr_train, color='green', lw=lw, label='Train ROC curve (area = %0
plt.plot(fpr_test, tpr_test, color='darkorange', lw=lw, label='Test ROC curve (area = %
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(29) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



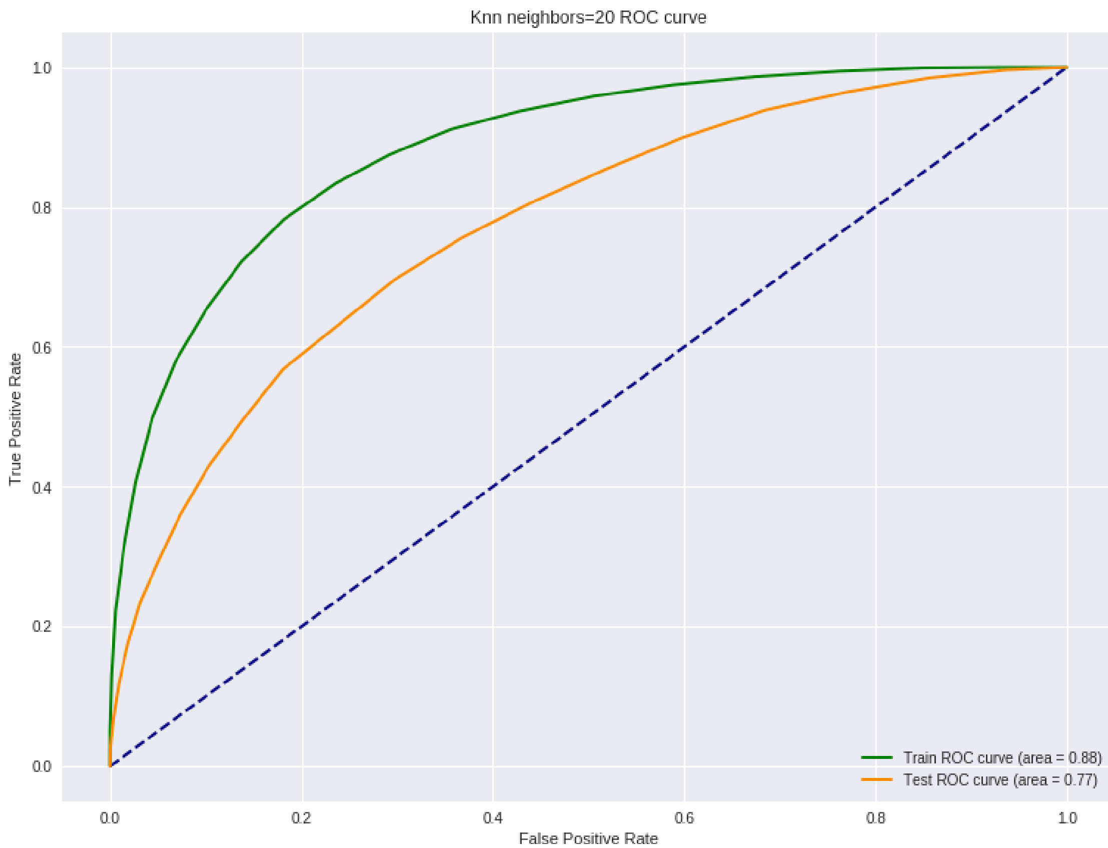
6.1.4 [5.1.4] TFIDF Word2Vec

In [0]:
```
#loading tfidf word2vec

with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_wei
    train_data = pickle.load(tfidf_w2v_pickle)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_wei
    cv_data = pickle.load(tfidf_w2v_pickle)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_wei
    test_data = pickle.load(tfidf_w2v_pickle)
```

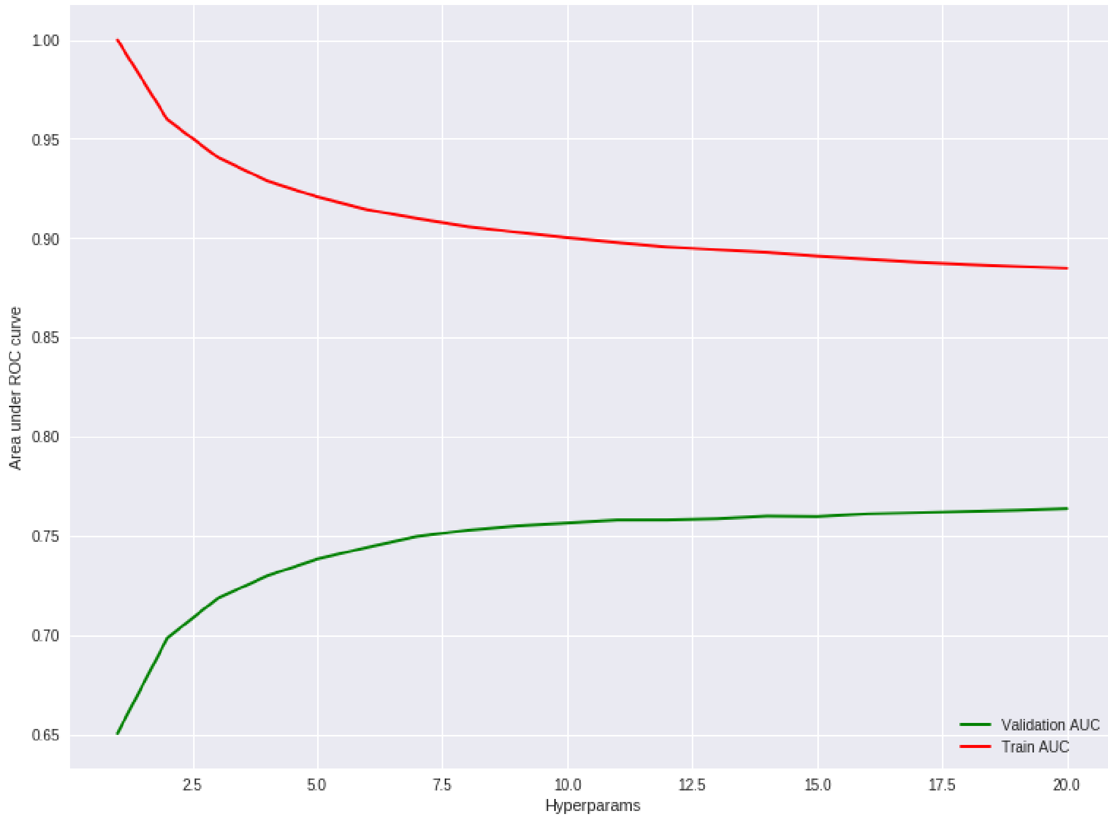
In [0]:
```
# finding best k using AUC
lw = 2
auc_train = []
auc_cv = []
auc_test = []
fpr_train = dict()
tpr_train = dict()
fpr_test = dict()
tpr_test = dict()
fpr_cv = dict()
```

```
tpr_cv = dict()

train_lab_bin = [1 if x=='positive' else 0 for x in bow_train_lab]
test_lab_bin = [1 if x=='positive' else 0 for x in bow_test_lab]
cv_lab_bin = [1 if x=='positive' else 0 for x in bow_cv_lab]

for idx, k in enumerate(range(1, 21)):
    knn_classifier = KNeighborsClassifier(n_neighbors=k, algorithm='brute')
    knn_classifier.fit(train_data, train_lab_bin)
    train_proba = knn_classifier.predict_proba(train_data)
    fpr_train[idx], tpr_train[idx], _ = roc_curve(train_lab_bin, train_proba[:,1])
    auc_train.append(auc(fpr_train[idx], tpr_train[idx]))

    test_proba = knn_classifier.predict_proba(test_data)
    fpr_test[idx], tpr_test[idx], _ = roc_curve(test_lab_bin, test_proba[:,1])
    auc_test.append(auc(fpr_test[idx], tpr_test[idx]))

    cv_proba = knn_classifier.predict_proba(cv_data)
    fpr_cv[idx], tpr_cv[idx], _ = roc_curve(cv_lab_bin, cv_proba[:,1])
    auc_cv.append(auc(fpr_cv[idx], tpr_cv[idx]))
```

In [0]:
```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_cv.index(max(auc_cv))
plt.plot(fpr_train[max_idx], tpr_train[max_idx], color='green', lw=lw, label='Train ROC
plt.plot(fpr_test[max_idx], tpr_test[max_idx], color='darkorange', lw=lw, label='Test R
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



In [0]:
```
# graph train auc, cv auc and hyper params
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p

plt.figure(figsize=(12.8, 9.6))
#plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_train.index(max(auc_train))
plt.plot(range(1, 21), auc_cv, color='green', lw=lw, label='Validation AUC')
plt.plot(range(1, 21), auc_train, color='red', lw=lw, label='Train AUC')
plt.xlabel('Hyperparams')
plt.ylabel('Area under ROC curve')
# plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



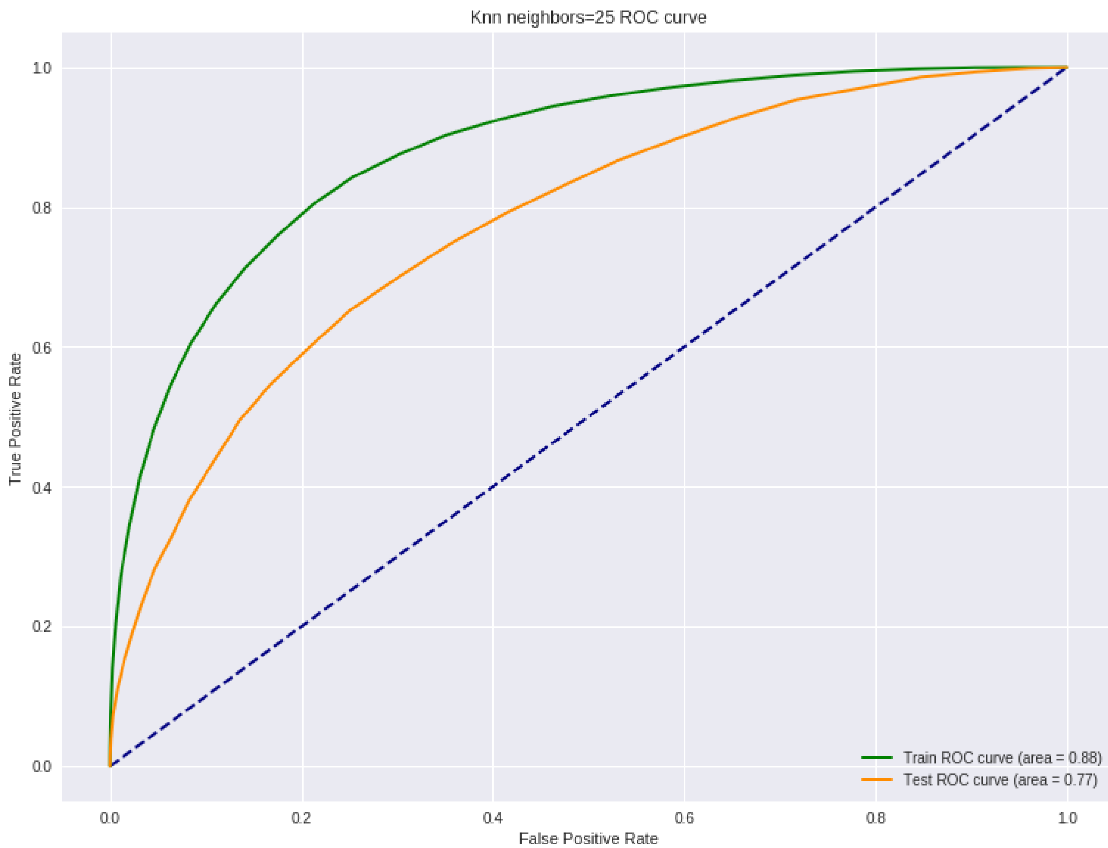

```
In [0]: params = {"n_neighbors": np.arange(1, 31, 2)}

        classifier = KNeighborsClassifier()
        grid = GridSearchCV(classifier, params, n_jobs=-1, verbose=2)
        grid.fit(train_data, train_lab_bin)
        acc = grid.score(cv_data, cv_lab_bin)

        print("CV Accuracy:", acc)
        print("Best Params", grid.best_params_)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 166.7min
[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed: 205.3min finished


CV Accuracy: 0.69445
Best Params {'n_neighbors': 27}
```

```
In [0]: # 27-NN
        knn_classifier = KNeighborsClassifier(n_neighbors=27, algorithm='brute')
        knn_classifier.fit(train_data, train_lab_bin)
        cv_predict = knn_classifier.predict(cv_data)
        print(classification_report(cv_lab_bin, cv_predict))
        train_proba = knn_classifier.predict_proba(train_data)
        fpr_train, tpr_train, _ = roc_curve(train_lab_bin, train_proba[:,1])
        auc_train = auc(fpr_train, tpr_train)

        test_proba = knn_classifier.predict_proba(test_data)
        fpr_test, tpr_test, _ = roc_curve(test_lab_bin, test_proba[:,1])
        auc_test = auc(fpr_test, tpr_test)
```

```
              precision    recall  f1-score   support

           0       0.67      0.77      0.72     10000
           1       0.73      0.62      0.67     10000

   micro avg       0.69      0.69      0.69     20000
   macro avg       0.70      0.69      0.69     20000
weighted avg       0.70      0.69      0.69     20000
```

```
In [0]: lw=2
        # plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p
        plt.figure(figsize=(12.8, 9.6))
        plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
        #max_idx = auc_cv.index(max(auc_cv))
        plt.plot(fpr_train, tpr_train, color='green', lw=lw, label='Train ROC curve (area = %0
        plt.plot(fpr_test, tpr_test, color='darkorange', lw=lw, label='Test ROC curve (area = %
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Knn neighbors=' + str(25) + ' ROC curve')
        plt.legend(loc="lower right")
        plt.show()
```



6.2 [5.2] KNN kd-tree

In [0]:
```
train_lab_bin = [1 if x=='positive' else 0 for x in train_df['Score'].values]
test_lab_bin = [1 if x=='positive' else 0 for x in test_df['Score'].values]
cv_lab_bin = [1 if x=='positive' else 0 for x in cv_df['Score'].values]
```

6.2.1 [5.2.1] Bag of words

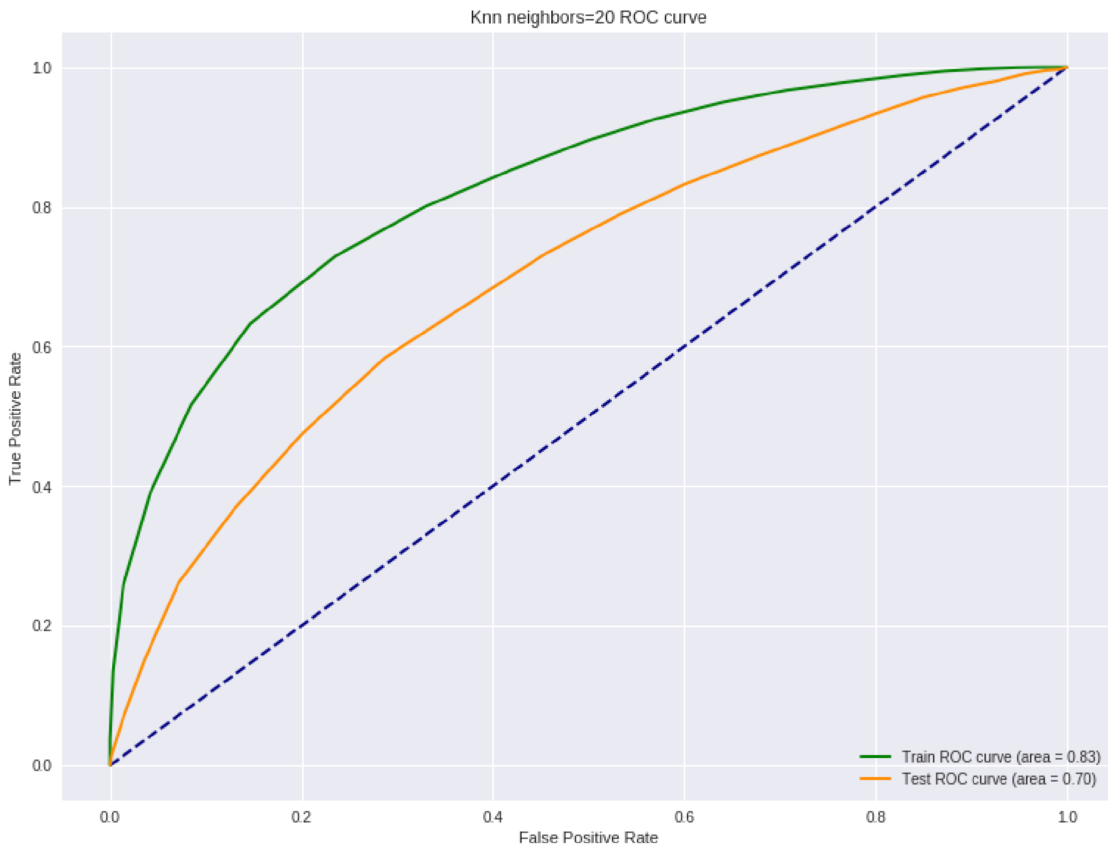
In [0]:
```
#BoW
count_vect = CountVectorizer(min_df=10, max_features=500) #in scikit-learn
train_data = count_vect.fit_transform(train_df['CleanedText'].values).toarray()
cv_data = count_vect.transform(cv_df['CleanedText'].values).toarray()
test_data = count_vect.transform(test_df['CleanedText'].values).toarray()
print("the type of count vectorizer ",type(train_data))
#print("the shape of out text BOW vectorizer ",train_data.get_shape())
#print("the number of unique words ", train_data.get_shape()[1])
```

```
the type of count vectorizer  <class 'numpy.ndarray'>
```

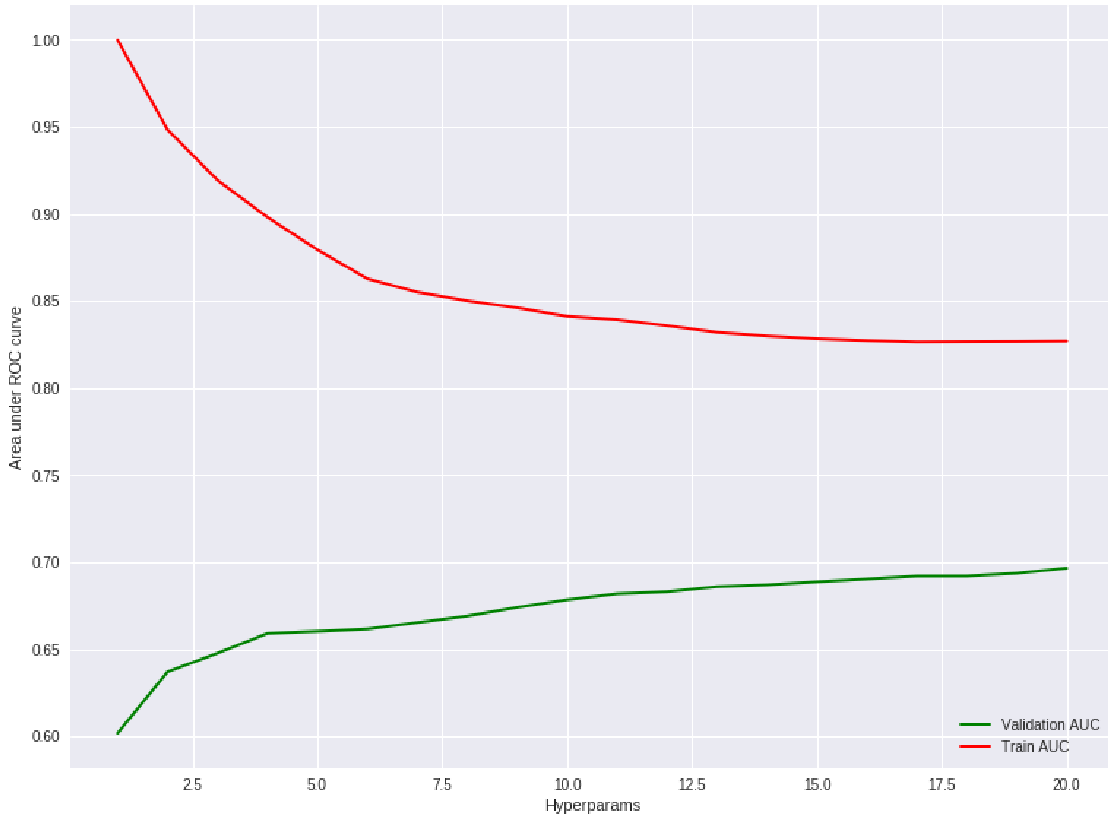
In [0]:
```
# finding best k using AUC
lw = 2
auc_train = []
auc_cv = []
auc_test = []
fpr_train = dict()
tpr_train = dict()
fpr_test = dict()
tpr_test = dict()
fpr_cv = dict()
tpr_cv = dict()

for idx, k in enumerate(range(1, 21)):
    print(k, end=" ")
    knn_classifier = KNeighborsClassifier(n_neighbors=k, algorithm='kd_tree')
    knn_classifier.fit(train_data, train_lab_bin)
    train_proba = knn_classifier.predict_proba(train_data)
    fpr_train[idx], tpr_train[idx], _ = roc_curve(train_lab_bin, train_proba[:,1])
    auc_train.append(auc(fpr_train[idx], tpr_train[idx]))

    test_proba = knn_classifier.predict_proba(test_data)
    fpr_test[idx], tpr_test[idx], _ = roc_curve(test_lab_bin, test_proba[:,1])
    auc_test.append(auc(fpr_test[idx], tpr_test[idx]))

    cv_proba = knn_classifier.predict_proba(cv_data)
    fpr_cv[idx], tpr_cv[idx], _ = roc_curve(cv_lab_bin, cv_proba[:,1])
    auc_cv.append(auc(fpr_cv[idx], tpr_cv[idx]))
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

In [0]:
```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_cv.index(max(auc_cv))
plt.plot(fpr_train[max_idx], tpr_train[max_idx], color='green', lw=lw, label='Train RO
plt.plot(fpr_test[max_idx], tpr_test[max_idx], color='darkorange', lw=lw, label='Test 
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



In [0]:
```
# graph train auc, cv auc and hyper params
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p

plt.figure(figsize=(12.8, 9.6))
#plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_train.index(max(auc_train))
plt.plot(range(1, 21), auc_cv, color='green', lw=lw, label='Validation AUC')
plt.plot(range(1, 21), auc_train, color='red', lw=lw, label='Train AUC')
plt.xlabel('Hyperparams')
plt.ylabel('Area under ROC curve')
# plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



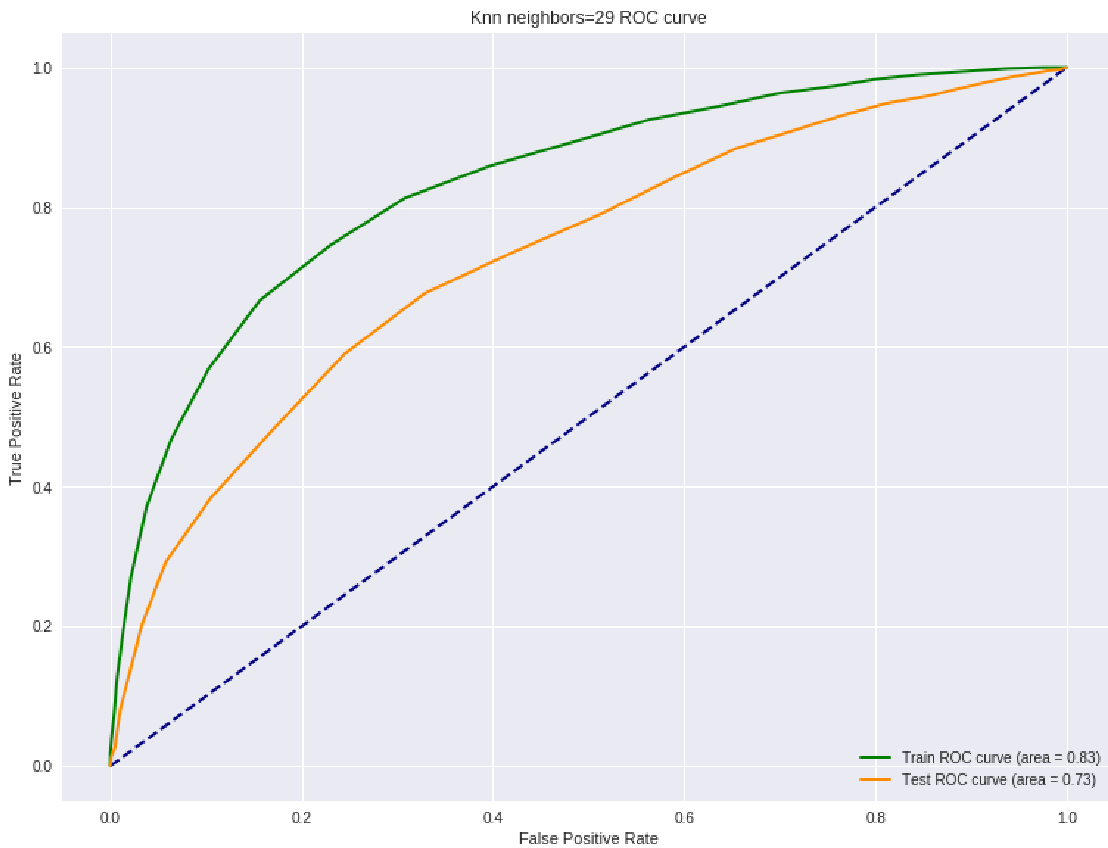

```
In [0]: params = {"n_neighbors": np.arange(1, 31, 3)}

        classifier = KNeighborsClassifier(algorithm='kd_tree')
        grid = GridSearchCV(classifier, params, n_jobs=-1, verbose=5)
        grid.fit(train_data, train_lab_bin)
        acc = grid.score(cv_data, cv_lab_bin)

        print("CV Accuracy:", acc)
        print("Best Params", grid.best_params_)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 18.3min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 40.0min finished


CV Accuracy: 0.63975
Best Params {'n_neighbors': 28}
```

In [0]:
```
# 28-NN
knn_classifier = KNeighborsClassifier(n_neighbors=28, algorithm='brute')
knn_classifier.fit(train_data, train_lab_bin)
cv_predict = knn_classifier.predict(cv_data)
print(classification_report(cv_lab_bin, cv_predict))
train_proba = knn_classifier.predict_proba(train_data)
fpr_train, tpr_train, _ = roc_curve(train_lab_bin, train_proba[:,1])
auc_train = auc(fpr_train, tpr_train)

test_proba = knn_classifier.predict_proba(test_data)
fpr_test, tpr_test, _ = roc_curve(test_lab_bin, test_proba[:,1])
auc_test = auc(fpr_test, tpr_test)
```

```
              precision    recall  f1-score   support

           0       0.68      0.60      0.64      2000
           1       0.64      0.72      0.68      2000

   micro avg       0.66      0.66      0.66      4000
   macro avg       0.66      0.66      0.66      4000
weighted avg       0.66      0.66      0.66      4000
```

In [0]:
```
lw=2
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
#max_idx = auc_cv.index(max(auc_cv))
plt.plot(fpr_train, tpr_train, color='green', lw=lw, label='Train ROC curve (area = %0
plt.plot(fpr_test, tpr_test, color='darkorange', lw=lw, label='Test ROC curve (area = %
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(29) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



6.2.2 [5.2.2] TFIDF

```
In [0]: #tf-idf
        tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features=500)
        train_data = tf_idf_vect.fit_transform(train_df['CleanedText'].values).toarray()
        cv_data = tf_idf_vect.transform(cv_df['CleanedText'].values).toarray()
        test_data = tf_idf_vect.transform(test_df['CleanedText'].values).toarray()
        print("the type of count vectorizer ",type(train_data))
        #print("the shape of out text TFIDF vectorizer ",train_data.get_shape())
        #print("the number of unique words including both unigrams and bigrams ", train_data.g
```

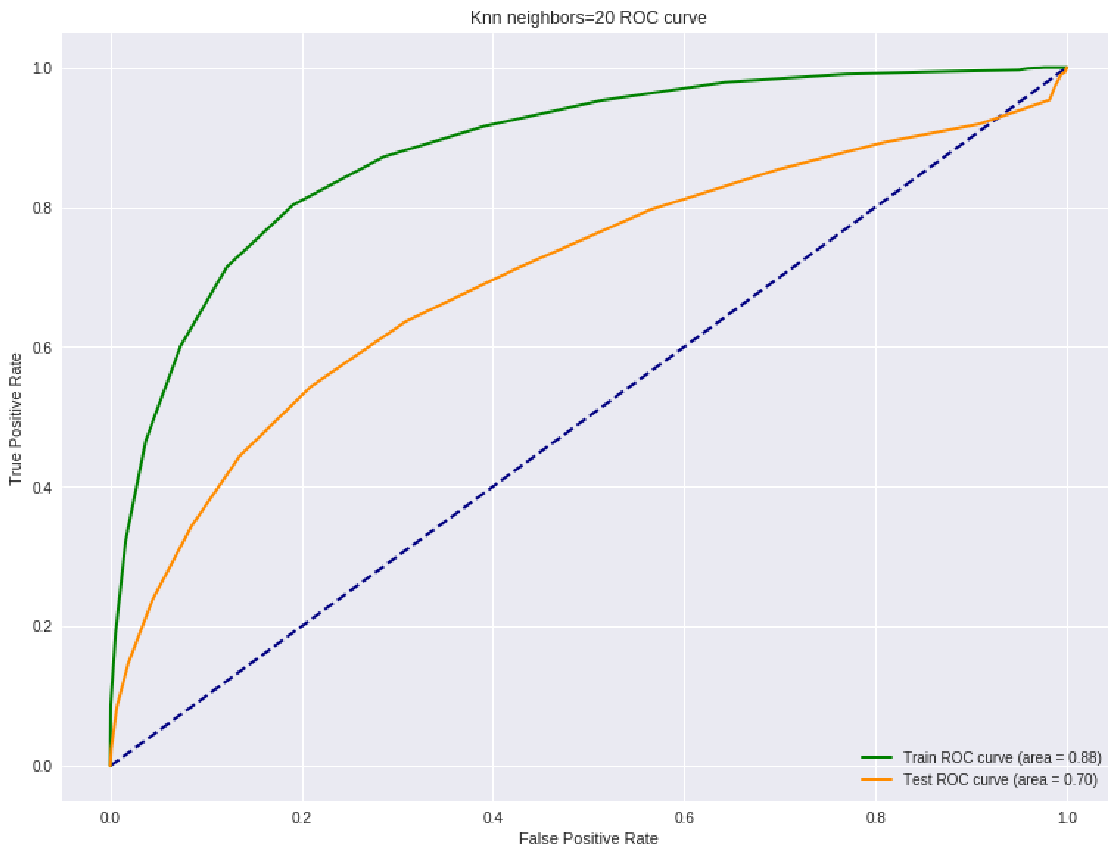
```
the type of count vectorizer  <class 'numpy.ndarray'>
```

```
In [0]: # finding best k using AUC
        lw = 2
        auc_train = []
        auc_cv = []
        auc_test = []
        fpr_train = dict()
        tpr_train = dict()
```

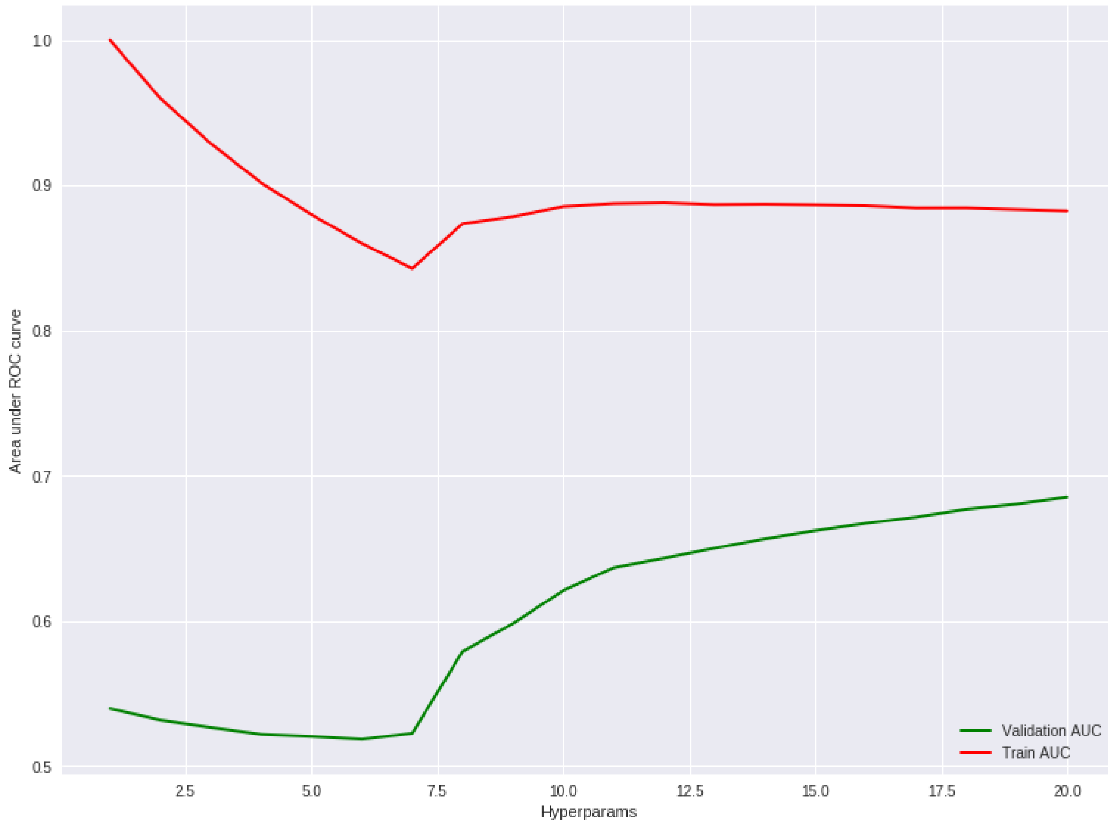
```
fpr_test = dict()
tpr_test = dict()
fpr_cv = dict()
tpr_cv = dict()

for idx, k in enumerate(range(1, 21)):
    print(k, end=" ")
    knn_classifier = KNeighborsClassifier(n_neighbors=k, algorithm='kd_tree')
    knn_classifier.fit(train_data, train_lab_bin)
    train_proba = knn_classifier.predict_proba(train_data)
    fpr_train[idx], tpr_train[idx], _ = roc_curve(train_lab_bin, train_proba[:,1])
    auc_train.append(auc(fpr_train[idx], tpr_train[idx]))

    test_proba = knn_classifier.predict_proba(test_data)
    fpr_test[idx], tpr_test[idx], _ = roc_curve(test_lab_bin, test_proba[:,1])
    auc_test.append(auc(fpr_test[idx], tpr_test[idx]))

    cv_proba = knn_classifier.predict_proba(cv_data)
    fpr_cv[idx], tpr_cv[idx], _ = roc_curve(cv_lab_bin, cv_proba[:,1])
    auc_cv.append(auc(fpr_cv[idx], tpr_cv[idx]))
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

In [0]:
```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_cv.index(max(auc_cv))
plt.plot(fpr_train[max_idx], tpr_train[max_idx], color='green', lw=lw, label='Train ROC
plt.plot(fpr_test[max_idx], tpr_test[max_idx], color='darkorange', lw=lw, label='Test R
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



In [0]:
```
# graph train auc, cv auc and hyper params
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p

plt.figure(figsize=(12.8, 9.6))
#plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_train.index(max(auc_train))
plt.plot(range(1, 21), auc_cv, color='green', lw=lw, label='Validation AUC')
plt.plot(range(1, 21), auc_train, color='red', lw=lw, label='Train AUC')
plt.xlabel('Hyperparams')
plt.ylabel('Area under ROC curve')
# plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



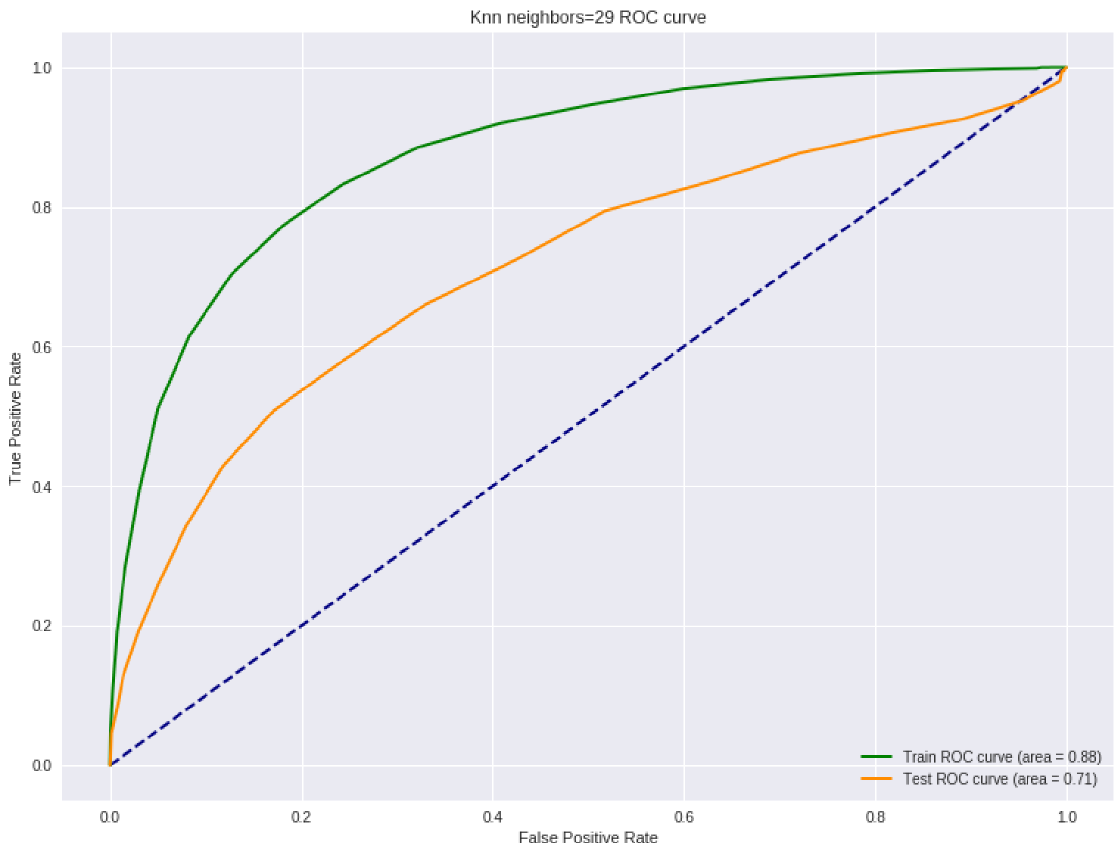

```
In [0]: params = {"n_neighbors": np.arange(1, 31, 3)}

        classifier = KNeighborsClassifier(algorithm='kd_tree')
        grid = GridSearchCV(classifier, params, n_jobs=-1, verbose=2)
        grid.fit(train_data, train_lab_bin)
        acc = grid.score(cv_data, cv_lab_bin)

        print("CV Accuracy:", acc)
        print("Best Params", grid.best_params_)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 40.1min finished


CV Accuracy: 0.65025
Best Params {'n_neighbors': 25}
```

In [0]:
```
# 25-NN
knn_classifier = KNeighborsClassifier(n_neighbors=25, algorithm='kd_tree')
knn_classifier.fit(train_data, train_lab_bin)
cv_predict = knn_classifier.predict(cv_data)
print(classification_report(cv_lab_bin, cv_predict))
train_proba = knn_classifier.predict_proba(train_data)
fpr_train, tpr_train, _ = roc_curve(train_lab_bin, train_proba[:,1])
auc_train = auc(fpr_train, tpr_train)

test_proba = knn_classifier.predict_proba(test_data)
fpr_test, tpr_test, _ = roc_curve(test_lab_bin, test_proba[:,1])
auc_test = auc(fpr_test, tpr_test)
```

```
              precision    recall  f1-score   support

           0       0.66      0.63      0.64      2000
           1       0.64      0.67      0.66      2000

   micro avg       0.65      0.65      0.65      4000
   macro avg       0.65      0.65      0.65      4000
weighted avg       0.65      0.65      0.65      4000
```

In [0]:
```
lw=2
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
#max_idx = auc_cv.index(max(auc_cv))
plt.plot(fpr_train, tpr_train, color='green', lw=lw, label='Train ROC curve (area = %0
plt.plot(fpr_test, tpr_test, color='darkorange', lw=lw, label='Test ROC curve (area = %
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(29) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



6.2.3 [5.2.3] Word2Vec

```
In [0]: train_data = avg_w2vec([sent.split() for sent in train_df['CleanedText'].values])
        cv_data = avg_w2vec([sent.split() for sent in cv_df['CleanedText'].values])
        test_data = avg_w2vec([sent.split() for sent in test_df['CleanedText'].values])
```

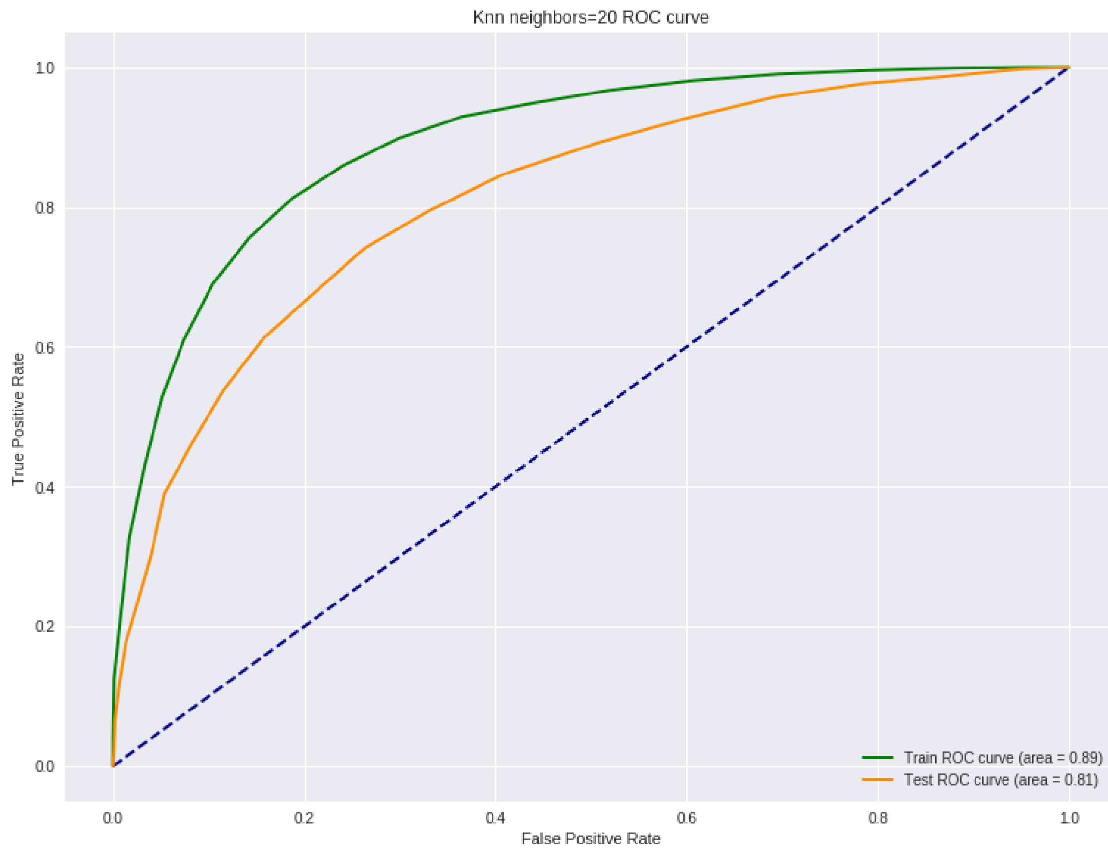
```
12000
50
4000
50
4000
50
```

```
In [0]: # finding best k using AUC
        lw = 2
        auc_train = []
        auc_cv = []
        auc_test = []
        fpr_train = dict()
        tpr_train = dict()
```

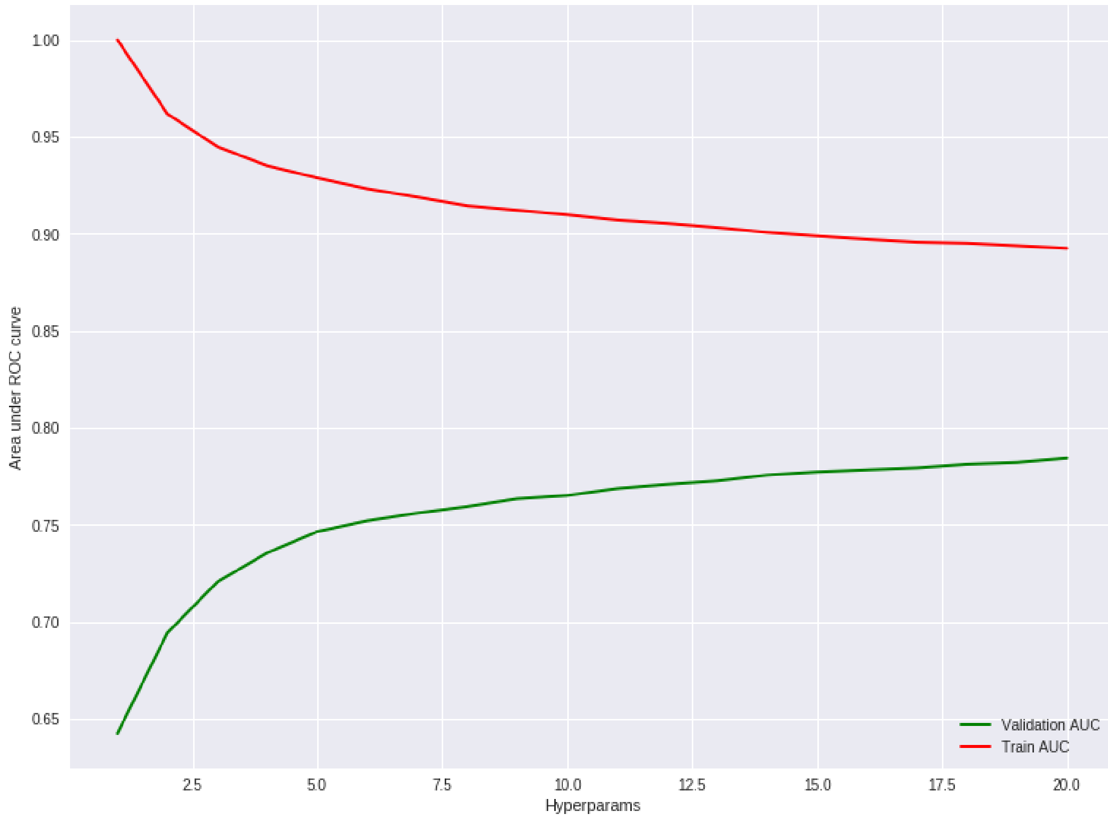
```
fpr_test = dict()
tpr_test = dict()
fpr_cv = dict()
tpr_cv = dict()

for idx, k in enumerate(range(1, 21)):
    print(k, end=" ")
    knn_classifier = KNeighborsClassifier(n_neighbors=k, algorithm='kd_tree')
    knn_classifier.fit(train_data, train_lab_bin)
    train_proba = knn_classifier.predict_proba(train_data)
    fpr_train[idx], tpr_train[idx], _ = roc_curve(train_lab_bin, train_proba[:,1])
    auc_train.append(auc(fpr_train[idx], tpr_train[idx]))

    test_proba = knn_classifier.predict_proba(test_data)
    fpr_test[idx], tpr_test[idx], _ = roc_curve(test_lab_bin, test_proba[:,1])
    auc_test.append(auc(fpr_test[idx], tpr_test[idx]))

    cv_proba = knn_classifier.predict_proba(cv_data)
    fpr_cv[idx], tpr_cv[idx], _ = roc_curve(cv_lab_bin, cv_proba[:,1])
    auc_cv.append(auc(fpr_cv[idx], tpr_cv[idx]))
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

In [0]:
```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_cv.index(max(auc_cv))
plt.plot(fpr_train[max_idx], tpr_train[max_idx], color='green', lw=lw, label='Train RO
plt.plot(fpr_test[max_idx], tpr_test[max_idx], color='darkorange', lw=lw, label='Test 
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



In [0]:
```
# graph train auc, cv auc and hyper params
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p

plt.figure(figsize=(12.8, 9.6))
#plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_train.index(max(auc_train))
plt.plot(range(1, 21), auc_cv, color='green', lw=lw, label='Validation AUC')
plt.plot(range(1, 21), auc_train, color='red', lw=lw, label='Train AUC')
plt.xlabel('Hyperparams')
plt.ylabel('Area under ROC curve')
# plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```




```
In [0]: params = {"n_neighbors": np.arange(1, 31, 3)}

        classifier = KNeighborsClassifier(algorithm='kd_tree')
        grid = GridSearchCV(classifier, params, n_jobs=-1, verbose=2)
        grid.fit(train_data, train_lab_bin)
        acc = grid.score(cv_data, cv_lab_bin)

        print("CV Accuracy:", acc)
        print("Best Params", grid.best_params_)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  5.7min finished


CV Accuracy: 0.691
Best Params {'n_neighbors': 7}
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

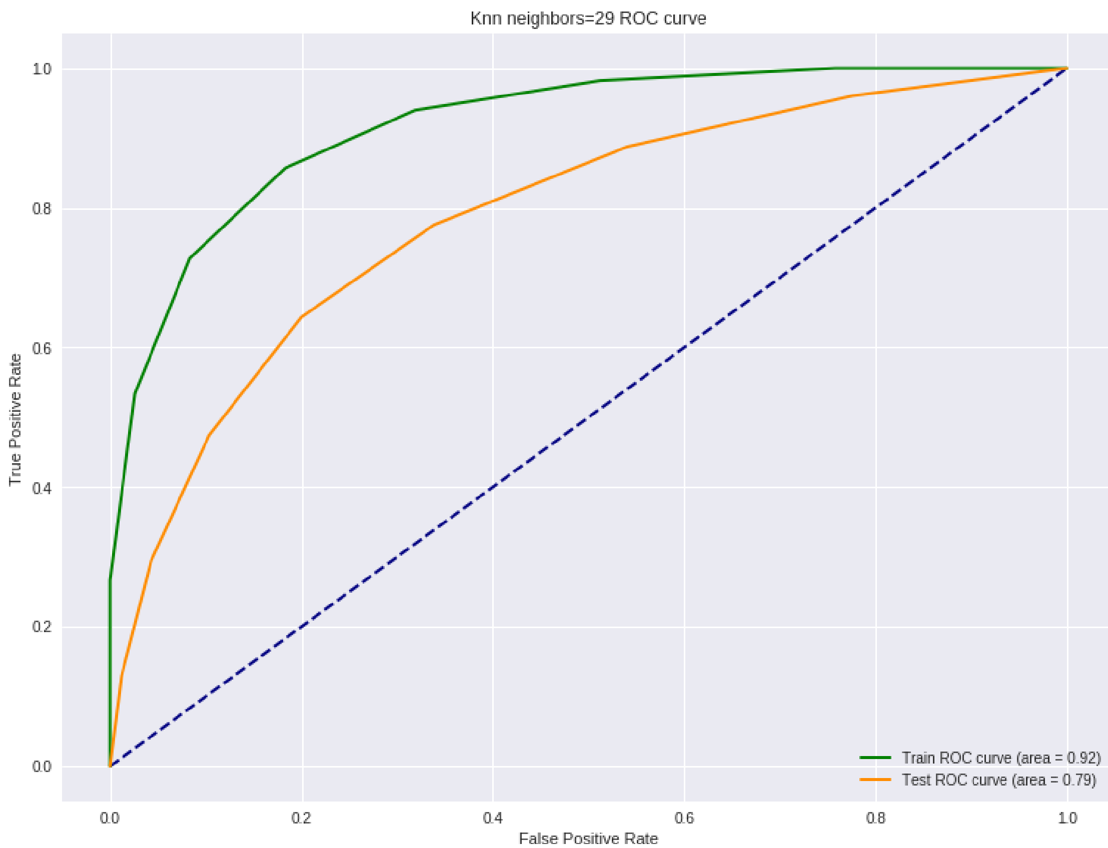
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  5.7min finished


CV Accuracy: 0.691
Best Params {'n_neighbors': 7}
Fitting 3 folds for each of 10 candidates, totalling 30 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  5.7min finished


CV Accuracy: 0.691
Best Params {'n_neighbors': 7}
```

In [0]:
```
# 7-NN
knn_classifier = KNeighborsClassifier(n_neighbors=7, algorithm='kd_tree')
knn_classifier.fit(train_data, train_lab_bin)
cv_predict = knn_classifier.predict(cv_data)
print(classification_report(cv_lab_bin, cv_predict))
train_proba = knn_classifier.predict_proba(train_data)
fpr_train, tpr_train, _ = roc_curve(train_lab_bin, train_proba[:,1])
auc_train = auc(fpr_train, tpr_train)

test_proba = knn_classifier.predict_proba(test_data)
fpr_test, tpr_test, _ = roc_curve(test_lab_bin, test_proba[:,1])
auc_test = auc(fpr_test, tpr_test)
```

```
              precision    recall  f1-score   support

           0       0.65      0.81      0.72      2000
           1       0.75      0.57      0.65      2000

   micro avg       0.69      0.69      0.69      4000
   macro avg       0.70      0.69      0.69      4000
weighted avg       0.70      0.69      0.69      4000
```

In [0]:
```
lw=2
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
#max_idx = auc_cv.index(max(auc_cv))
plt.plot(fpr_train, tpr_train, color='green', lw=lw, label='Train ROC curve (area = %0
plt.plot(fpr_test, tpr_test, color='darkorange', lw=lw, label='Test ROC curve (area = %
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(29) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



6.2.4 [5.2.4] TFIDF Word2Vec

```
In [0]: train_data = tfidf_w2vec([sent.split() for sent in train_df['CleanedText'].values])
        cv_data = tfidf_w2vec([sent.split() for sent in cv_df['CleanedText'].values])
        test_data = tfidf_w2vec([sent.split() for sent in test_df['CleanedText'].values])
```

```
100%|| 60000/60000 [18:48<00:00, 53.15it/s]
100%|| 20000/20000 [06:12<00:00, 53.74it/s]
100%|| 20000/20000 [06:01<00:00, 55.34it/s]
```

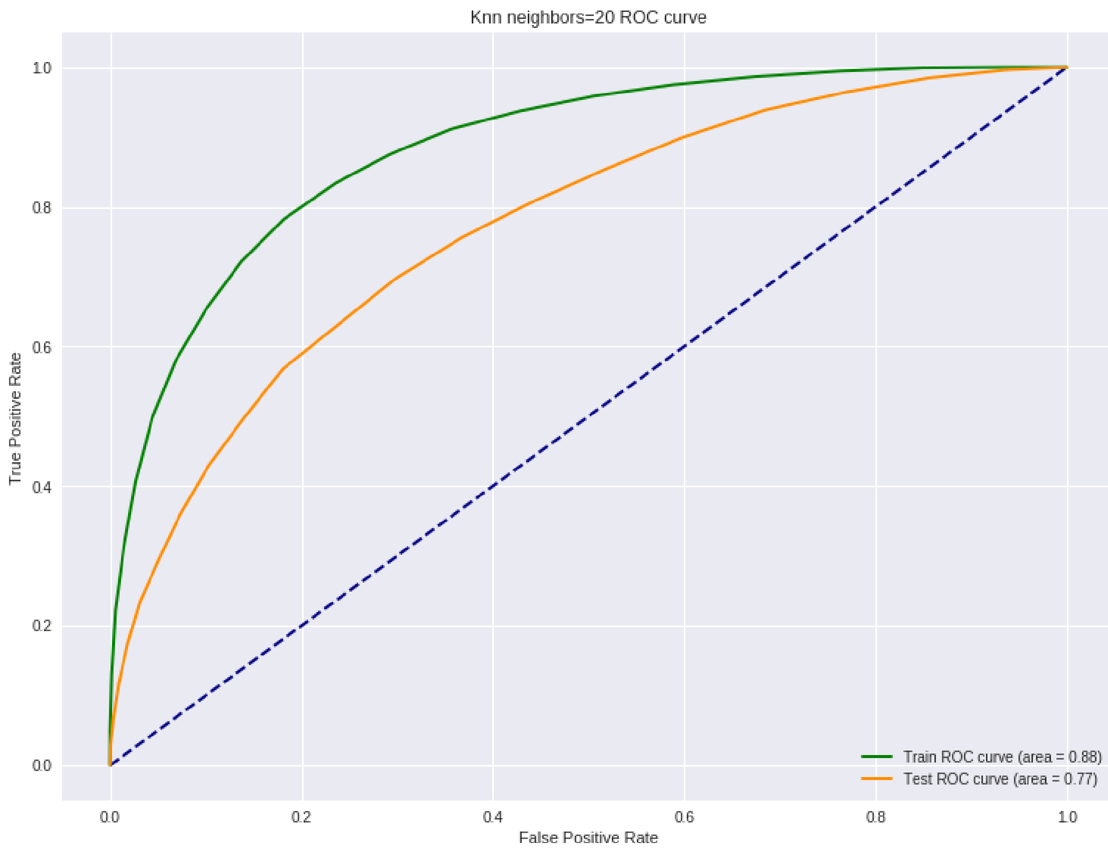
```
In [0]: print(len(train_data), len(train_lab_bin))
```

```
60000 60000
```

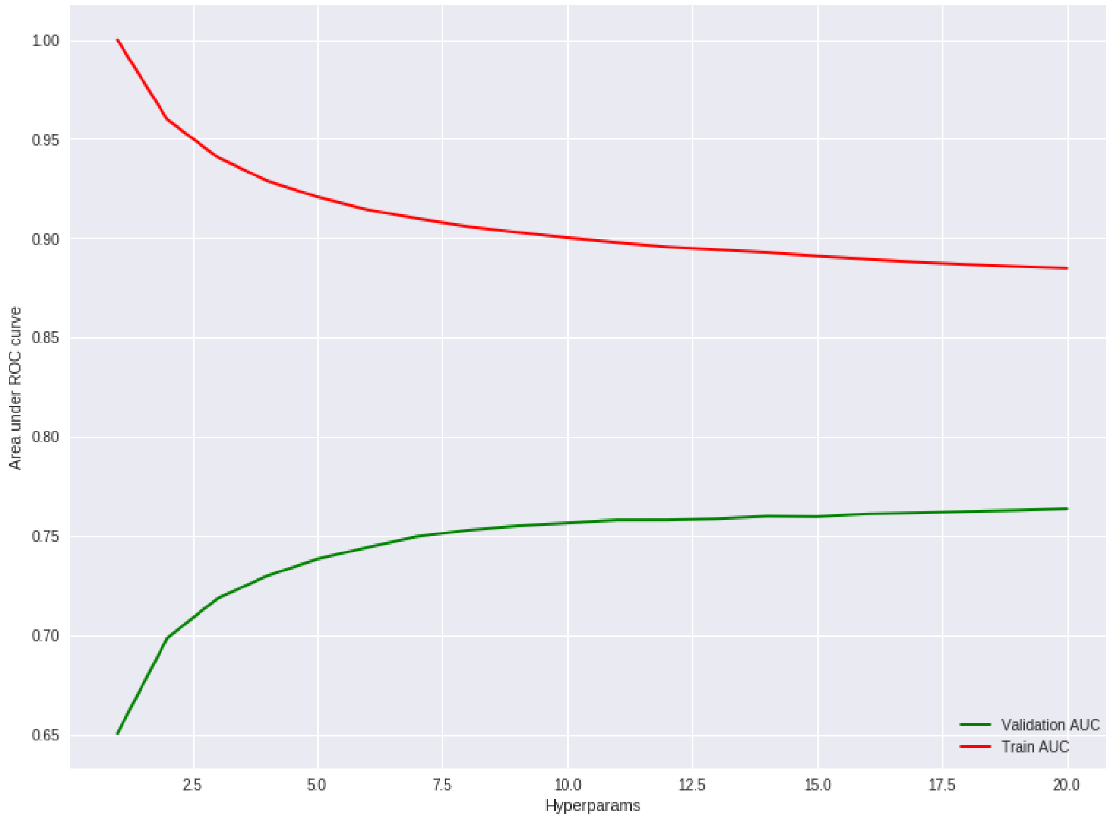
```
In [0]: # finding best k using AUC
        lw = 2
        auc_train = []
        auc_cv = []
        auc_test = []
        fpr_train = dict()
        tpr_train = dict()
        fpr_test = dict()
        tpr_test = dict()
        fpr_cv = dict()
        tpr_cv = dict()

        for idx, k in enumerate(range(1, 21)):
            print(k, end=" ")
            knn_classifier = KNeighborsClassifier(n_neighbors=k, algorithm='kd_tree')
            knn_classifier.fit(train_data, train_lab_bin)
            train_proba = knn_classifier.predict_proba(train_data)
            fpr_train[idx], tpr_train[idx], _ = roc_curve(train_lab_bin, train_proba[:,1])
            auc_train.append(auc(fpr_train[idx], tpr_train[idx]))

            test_proba = knn_classifier.predict_proba(test_data)
            fpr_test[idx], tpr_test[idx], _ = roc_curve(test_lab_bin, test_proba[:,1])
            auc_test.append(auc(fpr_test[idx], tpr_test[idx]))

            cv_proba = knn_classifier.predict_proba(cv_data)
            fpr_cv[idx], tpr_cv[idx], _ = roc_curve(cv_lab_bin, cv_proba[:,1])
            auc_cv.append(auc(fpr_cv[idx], tpr_cv[idx]))
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

```
In [0]: # plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p
        plt.figure(figsize=(12.8, 9.6))
        plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
        max_idx = auc_cv.index(max(auc_cv))
        plt.plot(fpr_train[max_idx], tpr_train[max_idx], color='green', lw=lw, label='Train ROC
        plt.plot(fpr_test[max_idx], tpr_test[max_idx], color='darkorange', lw=lw, label='Test R
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
        plt.legend(loc="lower right")
        plt.show()
```



In [0]:
```
# graph train auc, cv auc and hyper params
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p

plt.figure(figsize=(12.8, 9.6))
#plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_train.index(max(auc_train))
plt.plot(range(1, 21), auc_cv, color='green', lw=lw, label='Validation AUC')
plt.plot(range(1, 21), auc_train, color='red', lw=lw, label='Train AUC')
plt.xlabel('Hyperparams')
plt.ylabel('Area under ROC curve')
# plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```




```
In [0]: params = {"n_neighbors": np.arange(1, 31, 3)}

        classifier = KNeighborsClassifier(algorithm='kd_tree')
        grid = GridSearchCV(classifier, params, n_jobs=-1, verbose=2)
        grid.fit(train_data, train_lab_bin)
        acc = grid.score(cv_data, cv_lab_bin)

        print("CV Accuracy:", acc)
        print("Best Params", grid.best_params_)
```

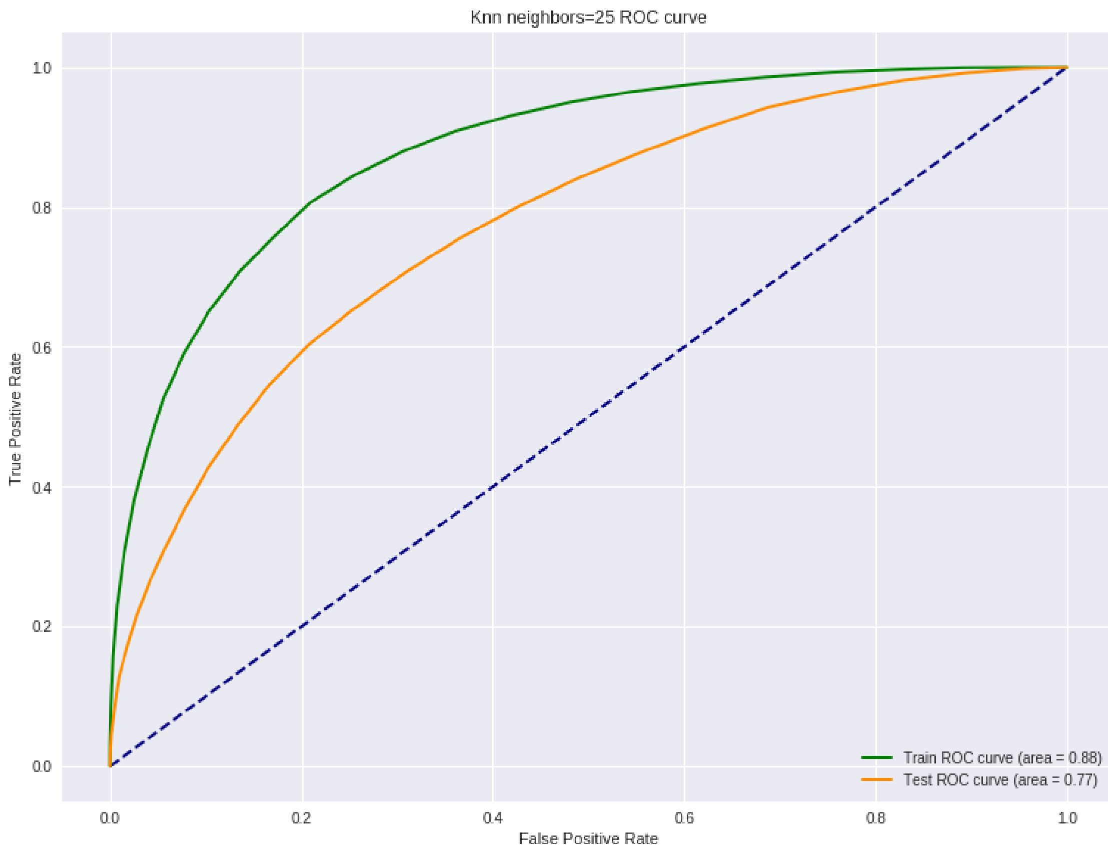
```
Fitting 3 folds for each of 10 candidates, totalling 30 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 144.9min finished


CV Accuracy: 0.6928
Best Params {'n_neighbors': 25}
```

```
In [0]: # 25-NN
        knn_classifier = KNeighborsClassifier(n_neighbors=25, algorithm='kd_tree')
```

```
knn_classifier.fit(train_data, train_lab_bin)
cv_predict = knn_classifier.predict(cv_data)
print(classification_report(cv_lab_bin, cv_predict))
train_proba = knn_classifier.predict_proba(train_data)
fpr_train, tpr_train, _ = roc_curve(train_lab_bin, train_proba[:,1])
auc_train = auc(fpr_train, tpr_train)

test_proba = knn_classifier.predict_proba(test_data)
fpr_test, tpr_test, _ = roc_curve(test_lab_bin, test_proba[:,1])
auc_test = auc(fpr_test, tpr_test)
```

```
              precision    recall  f1-score   support

           0       0.67      0.77      0.71     10000
           1       0.73      0.62      0.67     10000

   micro avg       0.69      0.69      0.69     20000
   macro avg       0.70      0.69      0.69     20000
weighted avg       0.70      0.69      0.69     20000
```

```
In [0]: lw=2
        # plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/p
        plt.figure(figsize=(12.8, 9.6))
        plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
        #max_idx = auc_cv.index(max(auc_cv))
        plt.plot(fpr_train, tpr_train, color='green', lw=lw, label='Train ROC curve (area = %0
        plt.plot(fpr_test, tpr_test, color='darkorange', lw=lw, label='Test ROC curve (area = %
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Knn neighbors=' + str(25) + ' ROC curve')
        plt.legend(loc="lower right")
        plt.show()
```



7 [6] Conclusion

```
In [0]: from prettytable import PrettyTable

In [0]: x = PrettyTable()

In [0]: x.field_names = ["Vectorizer", "Model", "Hyper parameter", "Test AUC"]

In [0]: x.add_row(["BoW", "Brute", 19, 0.77])
        x.add_row(["TFIDF", "Brute", 29, 0.73])
        x.add_row(["Word2Vec", "Brute", 29, 0.83])
        x.add_row(["TFIDF Word2Vec", "Brute", 27, 0.77])
        x.add_row(["BoW", "kd-tree", 28, 0.73])
        x.add_row(["TFIDF", "kd-tree", 25, 0.71])
        x.add_row(["Word2Vec", "kd-tree", 7, 0.79])
        x.add_row(["TFIDF Word2Vec", "kd-tree", 27, 0.77])
        print(x)
```

```
+----------------+---------+-----------------+----------+
|   Vectorizer   |  Model  | Hyper parameter | Test AUC |
```

BoW	Brute	19	0.77
TFIDF	Brute	29	0.73
Word2Vec	Brute	29	0.83
TFIDF Word2Vec	Brute	27	0.77
BoW	kd-tree	28	0.73
TFIDF	kd-tree	25	0.71
Word2Vec	kd-tree	7	0.79
TFIDF Word2Vec	kd-tree	27	0.77