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

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

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

Current Objective -

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

[1] Reading Data

[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=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%
b&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fww
ogleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

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
(from numba) (0.27.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from numba)
```

```
(1.14.6)
```

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
import of package 'nltk', but prior to execution of 'nltk.downloader'; this may result in
unpredictable behaviour
  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
import of package 'nltk', but prior to execution of 'nltk.downloader'; this may result in
unpredictable behaviour
  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_reviews/database.sqlite')

#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 positive reviews
data.head()
```

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

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Missing values

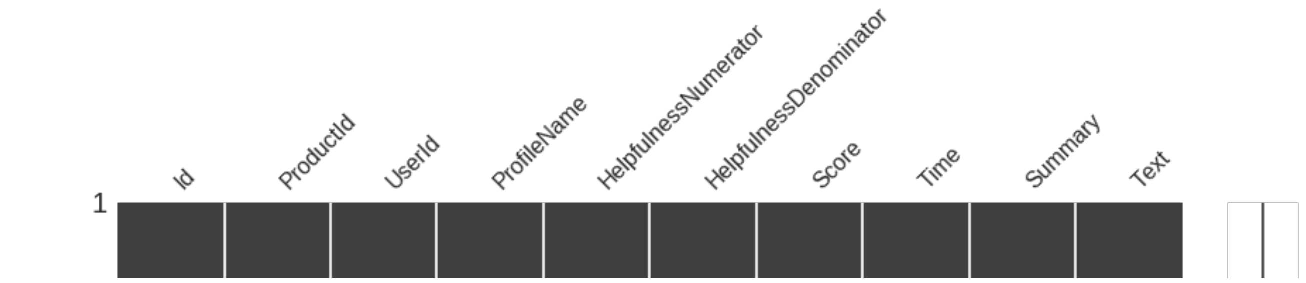
In [0]:

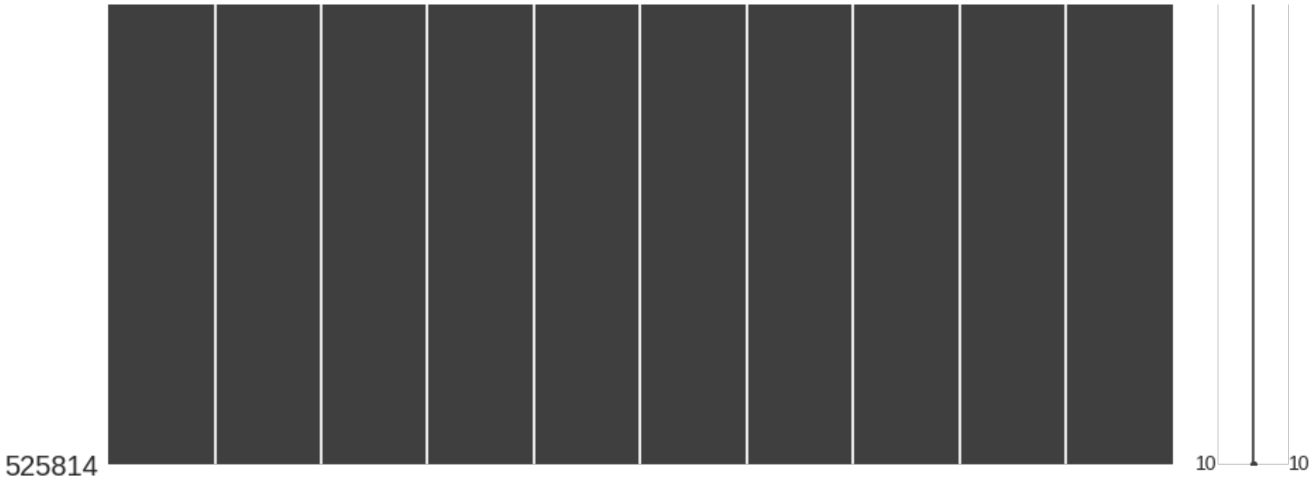
```
#let's just check, just in case if any
print("Missing values? Ans -", data.isnull().values.any())

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

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

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f35bf4505f8>
```





[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=True, inplace=False, kind='quicksort', na_position='last')
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
157863	171174	7310172001	AE9ZBY7WW3LIQ	W. K. Ota	0	0	4	1182902400
157871	171183	7310172001	AE9ZBY7WW3LIQ	W. K. Ota	5	13	1	1219363200
157912	171228	7310172001	AJD41FBJD9010	N. Ferguson "Two, Daisy, Hannah, and Kitten"	5	7	5	1233360000
157841	171152	7310172001	AJD41FBJD9010	N. Ferguson "Two, Daisy, Hannah, and Kitten"	0	0	5	1233360000
157842	171153	7310172001	AJD41FBJD9010	N. Ferguson "Two, Daisy, Hannah, and Kitten"	0	0	5	1233360000
157843	171154	7310172001	AJD41FBJD9010	N. Ferguson "Two, Daisy, Hannah, and Kitten"	0	0	5	1233360000
157876	171189	7310172001	AJD41FBJD9010	N. Ferguson "Two, Daisy, Hannah, and Kitten"	39	51	5	1233360000
157908	171223	7310172001	AJD41FBJD9010	N. Ferguson "Two, Daisy, Hannah, and Kitten"	1	1	5	1233360000
200626	217414	7310172101	AE9ZBY7WW3LIQ	W. K. Ota	5	13	1	1219363200
200618	217405	7310172101	AE9ZBY7WW3LIQ	W. K. Ota	0	0	4	1182902400
200597	217384	7310172101	AJD41FBJD9010	N. Ferguson "Two, Daisy, Hannah, and	0	0	5	1233360000

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
200631	217420	7310172101	AJD41FBJD9010	N. Ferguson "Two, Daisy, Hannah, and Kitten"	39	51	5	1233360000	fo
200598	217385	7310172101	AJD41FBJD9010	N. Ferguson "Two, Daisy, Hannah, and Kitten"	0	0	5	1233360000	b
200663	217454	7310172101	AJD41FBJD9010	N. Ferguson "Two, Daisy, Hannah, and Kitten"	1	1	5	1233360000	b
200667	217459	7310172101	AJD41FBJD9010	N. Ferguson "Two, Daisy, Hannah, and Kitten"	5	7	5	1233360000	
200596	217383	7310172101	AJD41FBJD9010	N. Ferguson "Two, Daisy, Hannah, and Kitten"	0	0	5	1233360000	d -
346048	374351	B00004CI84	A1K94LXX833JTT	Sanpete	1	2	5	1211760000	fa
346106	374412	B00004CI84	A1K94LXX833JTT	Sanpete	8	10	4	1213747200	
346119	374425	B00004CI84	A1K94LXX833JTT	Sanpete	10	14	4	1213747200	
417917	451939	B00004CXX9	A1K94LXX833JTT	Sanpete	8	10	4	1213747200	
417853	451871	B00004CXX9	A1K94LXX833JTT	Sanpete	1	2	5	1211760000	fa
417930	451952	B00004CXX9	A1K94LXX833JTT	Sanpete	10	14	4	1213747200	
212523	230338	B00004RYGX	A1K94LXX833JTT	Sanpete	8	10	4	1213747200	
212465	230277	B00004RYGX	A1K94LXX833JTT	Sanpete	1	2	5	1211760000	fa
212536	230351	B00004RYGX	A1K94LXX833JTT	Sanpete	10	14	4	1213747200	
341832	369818	B000084DWM	A25C5MVVCIYT5D	Natalie Dawn	1	1	5	1304726400	T
341815	369799	B000084DWM	A25C5MVVCIYT5D	Natalie Dawn	2	2	5	1304726400	
341817	369801	B000084DWM	A36JDIN9RAAIEC	Jon	2	2	5	1292976000	G
341818	369802	B000084DWM	A36JDIN9RAAIEC	Jon	2	2	5	1292976000	D
341806	369790	B000084DWM	A36JDIN9RAAIEC	Jon	3	3	5	1292976000	G

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
...	...	...	...	...	...	...	...	...
411612	445161	B009GHI5Q4	A3TVZM3ZIXG8YW	christopher hayes	11	15	1	1291420800
411671	445223	B009GHI5Q4	A3TVZM3ZIXG8YW	christopher hayes	6	15	1	1291420800
411611	445160	B009GHI5Q4	A3TVZM3ZIXG8YW	christopher hayes	7	9	1	1291420800
411608	445157	B009GHI5Q4	A3TVZM3ZIXG8YW	christopher hayes	3	3	1	1291420800
411603	445152	B009GHI5Q4	A3TVZM3ZIXG8YW	christopher hayes	18	24	1	1291420800
411599	445147	B009GHI5Q4	A3TVZM3ZIXG8YW	christopher hayes	19	21	1	1291420800
411621	445170	B009GHI5Q4	A3TVZM3ZIXG8YW	christopher hayes	33	48	1	1291420800
411659	445211	B009GHI5Q4	A3TVZM3ZIXG8YW	christopher hayes	2	4	1	1291420800
411670	445222	B009GHI5Q4	A3TVZM3ZIXG8YW	christopher hayes	6	14	1	1291420800
411613	445162	B009GHI5Q4	A3TVZM3ZIXG8YW	christopher hayes	11	15	1	1291420800
411631	445181	B009GHI5Q4	A966L65JSN8XN	N. Schleif "night owl"	1	1	5	1319241600
411651	445203	B009GHI5Q4	A966L65JSN8XN	N. Schleif "night owl"	0	0	5	1323820800
62140	67512	B009GHI6I6	A2ISKAWUPGGOLZ	M. S. Handley	2	4	1	1310774400
62142	67515	B009GHI6I6	A2ISKAWUPGGOLZ	M. S. Handley	0	1	1	1310774400
62138	67510	B009GHI6I6	A3TVZM3ZIXG8YW	christopher hayes	7	11	1	1291420800
62143	67516	B009GHI6I6	A3TVZM3ZIXG8YW	christopher hayes	0	2	1	1291420800
463853	501546	B009M2LUEW	A2AY7WOD04JYMY	J. Norden	1	1	5	1252195200
463852	501545	B009M2LUEW	A2AY7WOD04JYMY	J. Norden	1	1	5	1252713600
386528	417991	B009RB4GO4	A1FQSVI2WVV5W5	JLF	3	4	1	1319760000

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
386522	417984	B009RB4GO4	A1FQSVI2WVV5W5	JLF	1	1	1	1319760000	
386525	417988	B009RB4GO4	A1N06XIVTDQMP	LadyRae13	1	1	5	1316563200	
386455	417911	B009RB4GO4	A1N06XIVTDQMP	LadyRae13	0	0	4	1316563200	
386483	417942	B009RB4GO4	A21GDMT9JN2A5Y	Wayward Traveller "WaywardT"	0	1	1	1309910400	D
386431	417884	B009RB4GO4	A21GDMT9JN2A5Y	Wayward Traveller "WaywardT"	5	5	1	1309910400	
386530	417993	B009RB4GO4	A353Y7VBQHHW0T	wackygirl "wackygirl"	3	4	5	1318896000	it
386486	417946	B009RB4GO4	A353Y7VBQHHW0T	wackygirl "wackygirl"	5	10	5	1303776000	
386492	417952	B009RB4GO4	A3QVP3B2VVJ9T0	B. Fitzpatrick "BAFXF"	2	2	1	1327017600	
386356	417800	B009RB4GO4	A3QVP3B2VVJ9T0	B. Fitzpatrick "BAFXF"	0	0	1	1332633600	
386460	417917	B009RB4GO4	ANMGYT60QP4CM	Patricia Kagie	0	0	5	1311120000	C
386458	417914	B009RB4GO4	ANMGYT60QP4CM	Patricia Kagie	0	0	5	1315785600	C

11988 rows × 11 columns

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='quicksort', na_po
sition='last')
```

[2.3] Data cleaning: Deduplication - 1

In [0]:

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

Out[0]:

```
(364173, 10)
```

In [0]:

```
data.head(2)
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Su
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	5	939340800	E educ
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	1	4	1194739200	Lo boo th

[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, inplace=False
, kind='quicksort', na_position='last')
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Tim
67574	73444	B0046IISFG	A3OXHLG6DIBRW8	C. F. Hill "CFH"	1	1	5	13429152
287090	311004	B001EO6FPU	A3OXHLG6DIBRW8	C. F. Hill "CFH"	9	9	5	12970368
302818	327982	B0000CEQ6H	A281NPSIMI1C2R	Rebecca of Amazon "The Rebecca Review"	3	3	5	10844928
494235	534333	B0000CEQ72	A281NPSIMI1C2R	Rebecca of Amazon "The Rebecca Review"	1	1	5	10936512
387315	418839	B000FZYSVC	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	1	1	5	11730528
164025	177904	B000PSFW9Q	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	1	1	5	11567232

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Tim
267899	290387	B000S85AVI	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	2	2	5	11730528
443822	479891	B000Z91YTC	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	6	6	5	11567232
442191	478132	B0001GSP9G	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	1	1	5	11567232
177373	192340	B000M7OWLE	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	1	1	5	11730528
349975	378572	B0001GSPC8	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	1	1	5	11568960
308770	334367	B000M7OWMS	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	3	3	5	11730528
432171	467365	B0002W0RX6	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	2	2	5	11572416
306132	331530	B004JJ6ZN4	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	0	0	5	11567232
68214	74193	B000E4AHAK	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	4	4	5	11819520
36692	39874	B000CMIZ0I	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	13	13	5	11816928
204048	221073	B0001N4890	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	0	0	5	11704608
61803	67142	B0000CGFSC	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	1	1	5	11703744
524984	567556	B003ULE0TS	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	2	2	5	12960000
378979	409774	B000QVDP6Y	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	2	2	5	11567232
438391	474076	B000CQC08C	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	2	2	5	12819168

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Tim
442055	477988	B000BOZ6K4	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	2	2	5	11916288(
410856	444343	B0001M0ZTI	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	1	1	5	11916288(
113563	123178	B000CQBZOW	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	2	2	5	12820032(
360276	389666	B000Q61HH8	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	1	1	5	11899872(
488311	528026	B000EUCKF4	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	15	15	5	12170304(
367930	397829	B000PIMWGM	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	1	2	5	11876544(
367928	397826	B000PIMWGM	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	10	11	5	11877408(
227146	246294	B0009F3SB4	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	22	24	5	11894688(
443856	479926	B000VV0512	A1YUL9PCJR3JTY	O. Brown "Ms. O. Khannah-Brown"	0	0	5	11907648(
...	...	...	...	...	...	...	...	
484592	523982	B004JGQ15Y	A1KEK09ZA6J9P8	Colleen M. Schneider	0	0	5	13015296(
156517	169743	B004JGQ16I	A1KEK09ZA6J9P8	Colleen M. Schneider	0	0	5	13007520(
59565	64702	B0002ERVTM	A281NPSIMI1C2R	Rebecca of Amazon "The Rebecca Review"	1	1	5	12302496(
401926	434586	B000F9BCLW	A281NPSIMI1C2R	Rebecca of Amazon "The Rebecca Review"	0	0	5	11690784(
146793	159233	B002IYDXVE	A3R7Q2RWQ8K2S7	MamaCito	0	0	4	13004064(

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Tim
357423	386592	B003TN6ZN6	A3R7Q2RWQ8K2S7	MamaCito	9	9	4	13004928(
503424	544379	B002BCE97K	A3BTL4FV6ODKAT	fredtownward "The Analytical Mind; Have Brain...	1	1	5	13296096(
246458	267240	B000E123IC	A3BTL4FV6ODKAT	fredtownward "The Analytical Mind; Have Brain...	0	0	5	13293504(
505442	546536	B000E148MG	A3BTL4FV6ODKAT	fredtownward "The Analytical Mind; Have Brain...	1	2	5	13290912(
200791	217587	B000N8OLCC	A3BTL4FV6ODKAT	fredtownward "The Analytical Mind; Have Brain...	0	0	5	13291776(
314550	340574	B0018CJYCO	A2AHTUMQC1O3M8	Glenn Wagstaff "GBW"	1	1	5	12973824(
140727	152726	B0018CIPS8	A2AHTUMQC1O3M8	Glenn Wagstaff "GBW"	1	2	5	12971232(
469169	507321	B000EEDJGE	A20EEWWSFMZ1PN	bernie "webviator"	1	1	5	13169088(
35905	39033	B002PXEQCS	A20EEWWSFMZ1PN	bernie "webviator"	1	1	5	13446432(
479858	518889	B003BXOAKE	A3QZ6JT0R1OWEC	M. Goldman "M_gold~"	0	0	1	13494816(
514622	556404	B000IBILV6	A3QZ6JT0R1OWEC	M. Goldman "M_gold~"	0	0	1	13240800(
138146	149927	B0028C44IM	AC8C9PT59CDW1	M.A.R.	0	0	5	13337568(
447742	484120	B001IZ9ME6	AC8C9PT59CDW1	M.A.R.	0	0	5	13307328(
485560	525050	B0010B6IFY	A21B8AV7E3MPXE	Natalie V. Galasso	2	2	5	13041216(
38078	41352	B0096EZHM2	A21B8AV7E3MPXE	Natalie V. Galasso	3	3	4	13044672(

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Tim
54700	59365	B000FBM3RC	A1CVN6FWUCZOMD	A Customer	2	2	5	11695968
151003	163798	B000FBKFRW	A1CVN6FWUCZOMD	his_billyness	2	2	5	11695968
37746	40992	B001T5GHUM	A3PS4V0JQ2003X	PookieThePirate	8	9	5	13134528
118574	128597	B0026A2BS6	A3PS4V0JQ2003X	PookieThePirate	9	10	5	13141440
76868	83624	B005ZBZLT4	A3LL0U6E3QK34A	A Customer	0	1	4	13416192
167105	181178	B007Y59HVM	A3LL0U6E3QK34A	K. Biddle	0	1	4	13416192
242532	263029	B006N3HYYS	A3RFWQMLYSAKI0	Michael Burkett "reader rider"	0	8	4	12985920
99008	107540	B007TJGY4Q	A3RFWQMLYSAKI0	A Customer	0	8	4	12985920
521495	563807	B007JFMH8M	A248UQ9YXAMO9Z	Becky	0	0	5	13417920
521496	563808	B007JFMH8M	A3IMUU0I31XF33	Becky	0	0	5	13417920

630 rows × 10 columns

In [0]:

```
#removing duplicate reviews
data=data.drop_duplicates(subset={"Text"}, keep='first', inplace=False)
data.shape
```

Out[0]:

```
(363836, 10)
```

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.

[2.5] Data cleaning: Removing practically impossible data

In [0]:

```
#also removing those reviews where HelpfulnessNumerator is greater than HelpfulnessDenominator whi
ch is not possible
data=data[data['HelpfulnessNumerator']<=data['HelpfulnessDenominator']]
data.shape
```

Out[0]:

```
(363834, 10)
```

In [0]:

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

In [0]:

```
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)
```

[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 characters
    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't", "shouldn'
t", "wasn", "couldn't", 'mustn', "hadn't", "doesn't",
                      'wouldn', "weren't", "didn", "mustn't", "wasn't", "didn't", "don't", "not"]
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',
'themselves', 'on', 'now', 'own', 'before', 'yourself', 'i', 'did', 'didn', 'from', "weren't", 'ou
t', "that'll", 'during', "hadn't", 'these', 'but', 'nor', 'don', 'his', 'are', 'an', 'hadn', 'beca
use', 'very', "wouldn't", "needn't", 'as', 'where', 'too', 'shan', 'them', 'she', 'was', 'the', 'c
an', 'who', 'y', 'weren', 're', "haven't", 'whom', 'been', "won't", 'you', 'down', 'until', 't', '
d', 'my', 'won', 'through', 'that', "you'll", 'does', 'both', "couldn't", 'himself', 'ours',
'being', 'what', 'for', 'when', 'once', 'were', "doesn't", 'again', 'am', 'then', 'so',
"mightn't", "shan't", 'than', 'how', 'any', 'your', 'doing', 'here', 'ourselves', 'between',
"you're", "should've", 'there', 'myself', "you've", 'hers', 'which', 'under', 'same', 'against', '
will', "shouldn't", 'not', 'each', "wasn't", 'over', 'why', 'those', 'further', 'about', 'me', 'yo
urs', 'should', "you'd", 'shouldn', 'other', "she's", 'herself', "don't", "aren't", 'up',
'wouldn', 'in', 's', 'it', 'be', 'have', 'ma', 'has', 'is', 'her', 'few', 'all', 'such', 'haven',
'we', 'theirs', 'having', 'only', 'do', "mustn't", 'had', 'yourselves', 'after', 'by', 'mightn', '
m', 'its', 'some', 'below', 'most', 'o', 'he', 'above', 'a', 'their', 'wasn', 'isn', 'itself', 'if
', 'or', 'no', 'while', 'at', 'into', "didn't", 'll', 'with', 'to', "isn't", 'ain', 'of', 'doesn',
'him', 'hasn', 'aren', 'they', 'mustn']
Final stopwords: ['more', 'just', 'couldn', 'and', 'havent', 'our', 've', 'this', 'off',
'themselves', 'on', 'now', 'own', 'before', 'yourself', 'did', 'i', 'from', 'out', 'during', 'thes
e', 'but', 'nor', 'don', 'his', 'mightnt', 'are', 'an', 'because', 'very', 'youd', 'isnt',
'where', 'as', 'arent', 'too', 'shan', 'them', 'she', 'was', 'the', 'can', 'shouldve', 'who', 'y',
're', 'youve', 'whom', 'been', 'you', 'down', 'until', 't', 'd', 'my', 'won', 'through', 'that', '
does', 'both', 'himself', 'ours', 'being', 'what', 'wouldnt', 'for', 'when', 'once', 'wont',
'were', 'again', 'am', 'then', 'so', 'than', 'how', 'any', 'your', 'doing', 'here', 'ourselves', '
between', 'there', 'myself', 'hers', 'which', 'under', 'same', 'against', 'will', 'each', 'over',
'why', 'those', 'further', 'about', 'me', 'yours', 'should', 'other', 'herself', 'up', 'in', 's',
'youre', 'it', 'shes', 'be', 'have', 'ma', 'has', 'is', 'her', 'few', 'all', 'such', 'haven',
'we', 'theirs', 'having', 'only', 'do', 'youll', 'had', 'yourselves', 'after', 'by', 'mightn', 'm'
, 'its', 'some', 'below', 'most', 'o', 'he', 'above', 'a', 'their', 'isn', 'itself', 'if', 'or', '
no', 'while', 'at', 'hasnt', 'into', 'll', 'with', 'to', 'ain', 'of', 'him', 'thatll', 'hasn', 'ar
en', 'they']
```

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("utf8")
                    filtered_sentence.append(s)
                    if scores[i] == "positive":
                        all_positive_words.append(s) #list of all words used to describe positive r
eviews
                    if scores[i] == "negative":
                        all_negative_words.append(s) #list of all words used to describe negative r
eviews reviews
                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 data after pre
-processing of the review
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_reviews/final.sqlite')
c=conn.cursor()
conn.text_factory = str
data.to_sql('Reviews', conn,  schema=None, if_exists='replace', index=True, index_label=None, chunk
size=None, dtype=None)
```

In [0]:

```
###########################################################################
###########################################################################
con = sqlite3.connect('/content/gdrive/My
Drive/appliedAI/datasets/amzn_fine_food_reviews/final.sqlite')
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	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summa
0	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	positive	939340800	EVE boo educatio
1	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	1	positive	1194739200	Love book, m the ha co vers
2	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	1	positive	1191456000	chick soup w r mon
									a gc

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
3	150508	0006641040	AZGXZ2UUK6X	Hallberg " (Kate)"	1	1	positive	1076025600	a ge rhythm read alo
4	150509	0006641040	A3CMRKGE0P909G	Teresa	3	4	positive	1018396800	A gr way learn mon

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_position='la
st')
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_position='la
st')
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%
```

```
# 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)
```

[4] Featurization

[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'love',
104938), (b'great', 103358), (b'use', 101467), (b'make', 100401), (b'flavor', 99414), (b'one', 968
00), (b'get', 93100), (b'product', 90812), (b'try', 86221), (b'tea', 82860), (b'coffee', 78957), (
b'find', 78423), (b'buy', 75962), (b'food', 64946), (b'would', 59996), (b'eat', 57438), (b'time',
54081)]
Most Common Negative Words :  [(b'taste', 33523), (b'like', 31734), (b'product', 28122), (b'buy',
20800), (b'one', 20593), (b'would', 20028), (b'get', 20000), (b'flavor', 18123), (b'try', 17575),
(b'make', 16240), (b'use', 14915), (b'good', 14894), (b'coffee', 14764), (b'order', 12792),
(b'food', 12756), (b'think', 11931), (b'tea', 11633), (b'eat', 11013), (b'even', 10947), (b'box',
10812)]
```

In [0]:

```
#saving BoW unigrams
with open("/content/gdrive/My
Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_uni_vec_train.pkl", 'wb') as bow:
    pickle.dump(train_final_counts, bow)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/train_lab.pkl", 'wb'
) as bow:
    pickle.dump(train_df['Score'].values, bow)

with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_uni_vec_cv.pkl",
'wb') as bow:
    pickle.dump(cv_final_counts, bow)
```

```
    pickle.dump(cv_final_counts, bow)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/cv_lab.pkl", 'wb') a
s bow:
    pickle.dump(cv_df['Score'].values, bow)

with open("/content/gdrive/My
Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_uni_vec_test.pkl", 'wb') as bow:
    pickle.dump(test_final_counts, bow)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/test_lab.pkl", 'wb')
as bow:
    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_counts.get_s
hape()[1])
```

```
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_vec_train.pkl", 'wb') as bow:
    pickle.dump(train_bigram_counts, bow)
# with open("/content/gdrive/My
Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_vec_train_lab.pkl", 'wb') as bow:
#     pickle.dump(train_df['Score'].values, bow)

with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_vec_cv.pkl",
'wb') as bow:
    pickle.dump(cv_bigram_counts, bow)
# with open("/content/gdrive/My
Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_vec_cv_lab.pkl", 'wb') as bow:
#     pickle.dump(cv_df['Score'].values, bow)

with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_vec_test.pkl"
, 'wb') as bow:
    pickle.dump(test_bigram_counts, bow)
# with open("/content/gdrive/My
Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_vec_test_lab.pkl", 'wb') as bow:
#     pickle.dump(test_df['Score'].values, bow)
```

[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.get_shape()[
1])
```

```
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 person',
'bpa personally', 'bpa plastic', 'bpa prove', 'bpa really', 'bpa recently', 'bpa rest', 'bpa safe'
]
```

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 names.'''
    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_vec_train.pkl"
, 'wb') as bow:
    pickle.dump(train_tf_idf, bow)
# with open("/content/gdrive/My
Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_vec_train_lab.pkl", 'wb') as bow:
#     pickle.dump(train_df['Score'].values, bow)

with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_vec_cv.pkl", '
wb') as bow:
    pickle.dump(cv_tfidf, bow)
# with open("/content/gdrive/My
Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_vec_cv_lab.pkl", 'wb') as bow:
#     pickle.dump(cv_df['Score'].values, bow)

with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_vec_test.pkl",
'wb') as bow:
    pickle.dump(test_tfidf, bow)
# with open("/content/gdrive/My
Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_vec_test_lab.pkl", 'wb') as bow:
#     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())
```

[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_w2v_vec.model")
```

In [0]:

```
#loading model
w2v_model = Word2Vec.load("/content/gdrive/My
Drive/appliedAI/datasets/amzn_fine_food_reviews/amzn_w2v_vec.model")
```

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',
'drive', 'along', 'always', 'sing', 'refrain', 'learn', 'whale', 'india', 'droop', 'rose', 'love',
'new', 'word', 'classic', 'willing', 'bet', 'still', 'able', 'memory', 'college', 'grow', 'read',
'sendak', 'watch', 'really', 'rosie', 'movie', 'incorporate', 'however', 'miss', 'hard', 'cover',
```

'version', 'paperback', 'seem', 'kind', 'flimsy', 'take', 'two', 'hand', 'keep']

[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_train.pkl",
'wb') as w2v_pickle:
    pickle.dump(avg_w2v_train, w2v_pickle)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/avg_w2v_cv.pkl", 'wb
') as w2v_pickle:
    pickle.dump(avg_w2v_cv, w2v_pickle)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/avg_w2v_test.pkl", '
wb') as w2v_pickle:
    pickle.dump(avg_w2v_test, w2v_pickle)
```

[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 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 = final_tf_idf[row, tfidf_feat.index(word)]
                sent_vec += (vec * tf_idf)
                weight_sum += tf_idf
        if weight_sum != 0:
```

```python
            sent_vec /= weight_sum
        tfidf_sent_vectors.append(sent_vec)
        row += 1
        #print(row, end=" ")
    return tfidf_sent_vectors
```

In [0]:

```python
from tqdm import tqdm
```

In [0]:

```python
helper_numba = jit()(helper)
```

In [0]:

```python
# 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]:

```python
# 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 = tfidf

    tfidf_sent_vectors = []; # 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 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]:

```python
# 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 list
#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
enumerate(list_of_sent)]
#     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_weighted_w2v_train.pkl", 'wb') as
tfidf_w2v_pickle:
    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_weighted_w2v_cv.pkl", 'wb') as
tfidf_w2v_pickle:
    pickle.dump(tfidf_sent_vectors, tfidf_w2v_pickle)
```

In [0]:

```
with open("/content/gdrive/My
Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_weighted_w2v_test.pkl", 'wb') as
tfidf_w2v_pickle:
    pickle.dump(tfidf_sent_vectors, tfidf_w2v_pickle)
```

[5] KNN Assignment

[5.1] KNN Brute Force

[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_vec_train.pkl", 'rb') as bow:
    bow_train = pickle.load(bow)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/train_lab.pkl", 'rb'
) as bow:
    bow_train_lab = pickle.load(bow)

with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_uni_vec_cv.pkl",
'rb') as bow:
    bow_cv = pickle.load(bow)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/cv_lab.pkl", 'rb') a
s bow:
    bow_cv_lab = pickle.load(bow)

with open("/content/gdrive/My
Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_uni_vec_test.pkl", 'rb') as bow:
    bow_test = pickle.load(bow)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/test_lab.pkl", 'rb')
as bow:
    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]
```

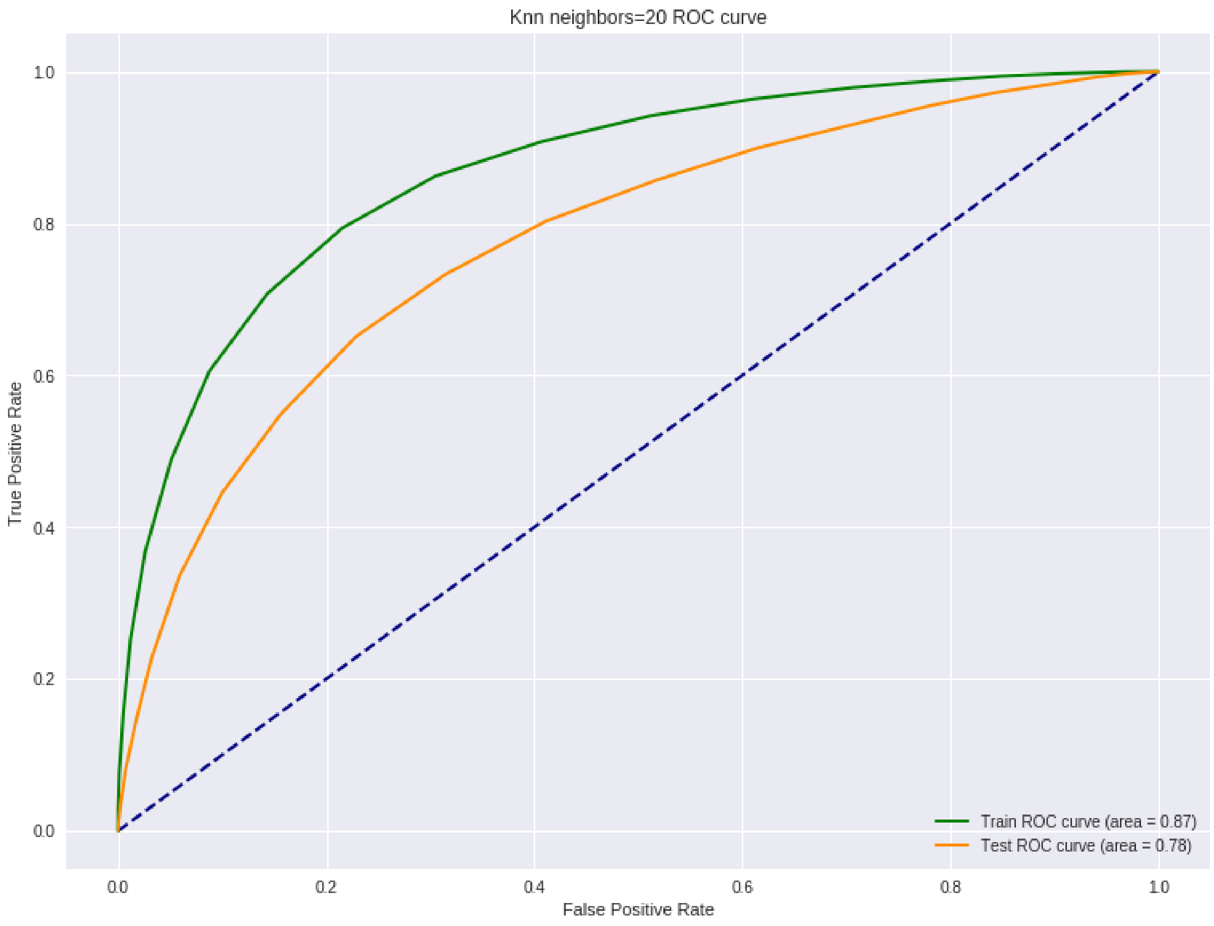
In [0]:

```
# 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[:,1])
    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]:

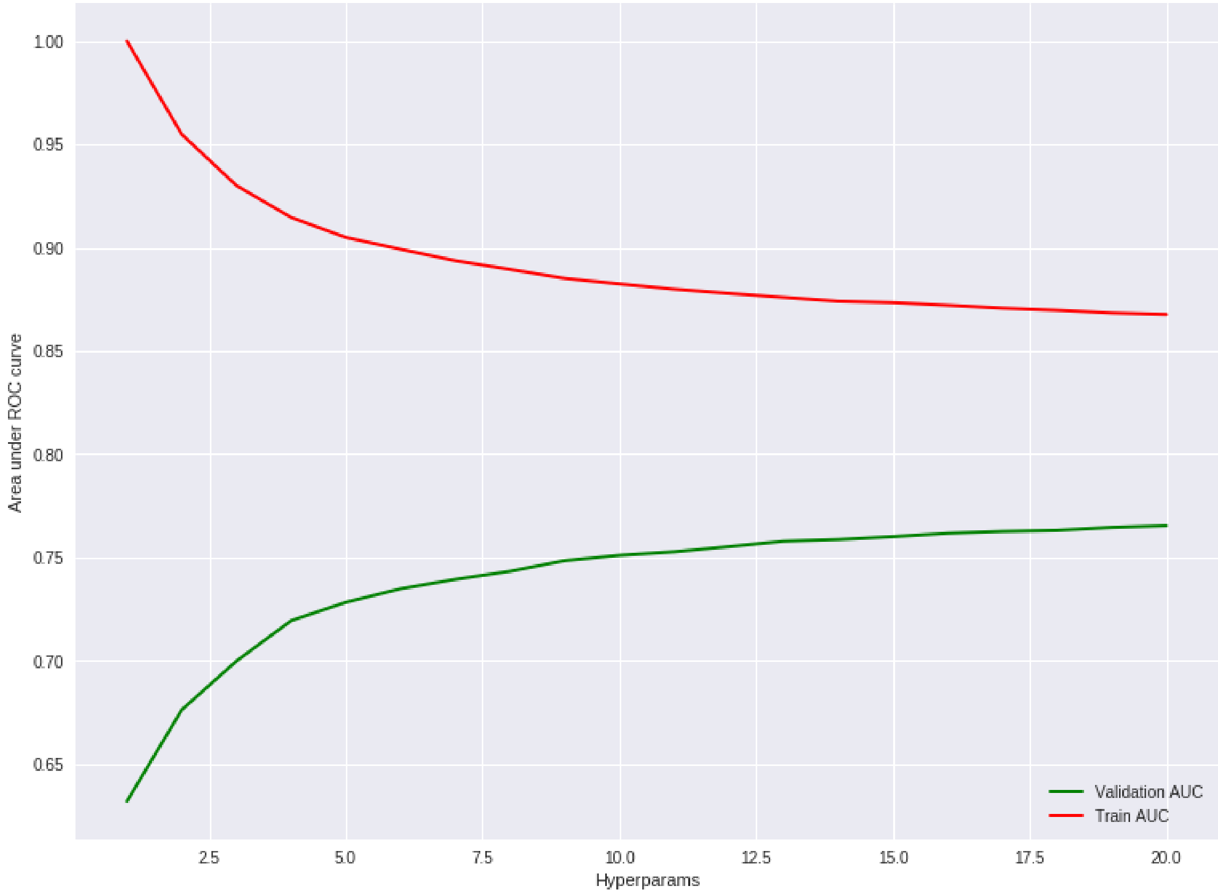
```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_cv.index(max(auc_cv))
plt.plot(fpr_train[max_idx], tpr_train[max_idx], color='green', lw=lw, label='Train ROC curve
(area = %0.2f)' % auc_train[max_idx])
plt.plot(fpr_test[max_idx], tpr_test[max_idx], color='darkorange', lw=lw, label='Test ROC curve
(area = %0.2f)' % auc_test[max_idx])
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/plot_roc.html

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]:

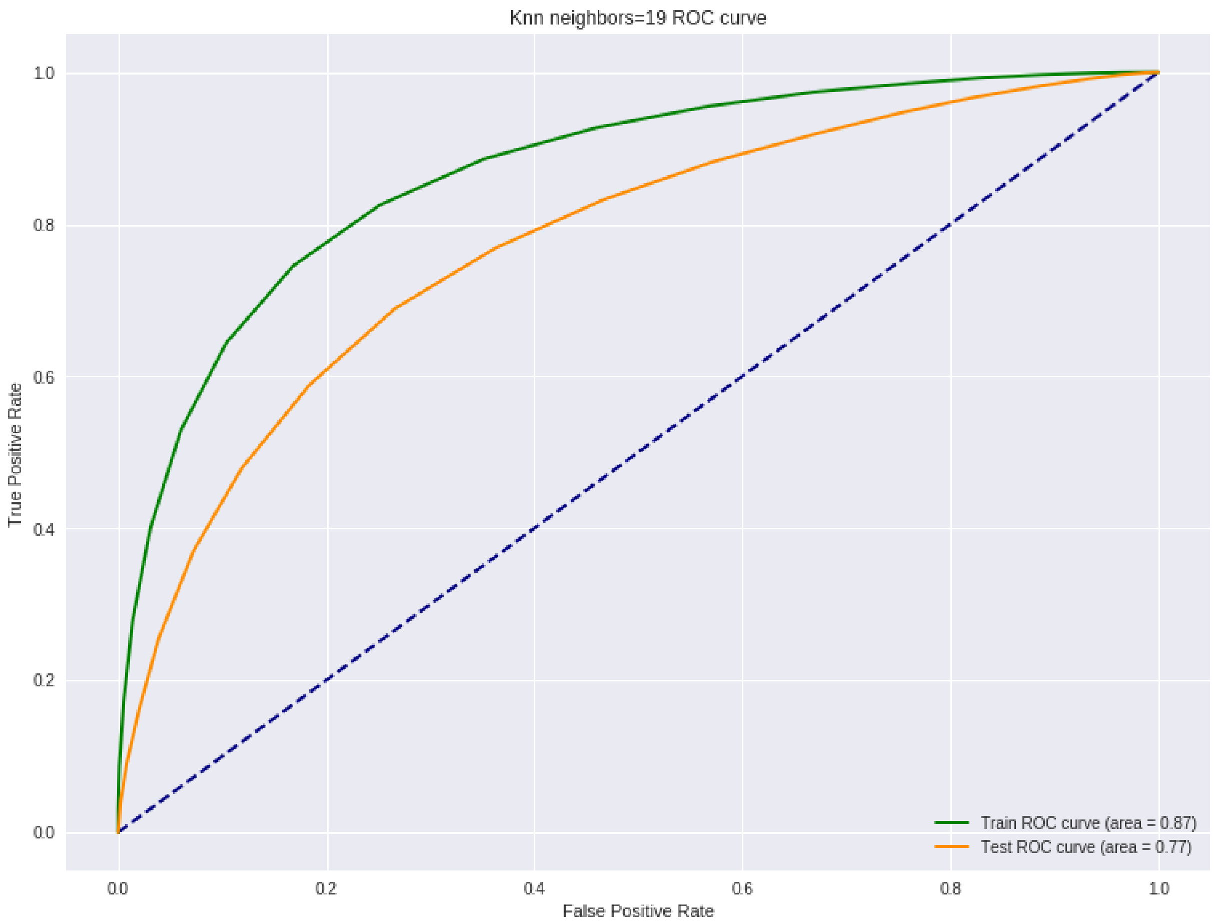
```
# 19-NN
knn_classifier = KNeighborsClassifier(n_neighbors=19, algorithm='brute')
knn_classifier.fit(bow_train, bow_train_lab)
bow_cv_predict = knn_classifier.predict(bow_cv)
print(classification_report(bow_cv_lab, bow_cv_predict))
train_proba = knn_classifier.predict_proba(bow_train)
fpr_train, tpr_train, _ = 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/plot_roc.html
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 = %0.2f)' % auc_t
rain)
plt.plot(fpr_test, tpr_test, color='darkorange', lw=lw, label='Test ROC curve (area = %0.2f)' % auc
_test)
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()
```



[5.1.2] TFIDF

In [0]:

```
# loading tfidf vectors
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_vec_train.pkl"
, 'rb') as bow:
```

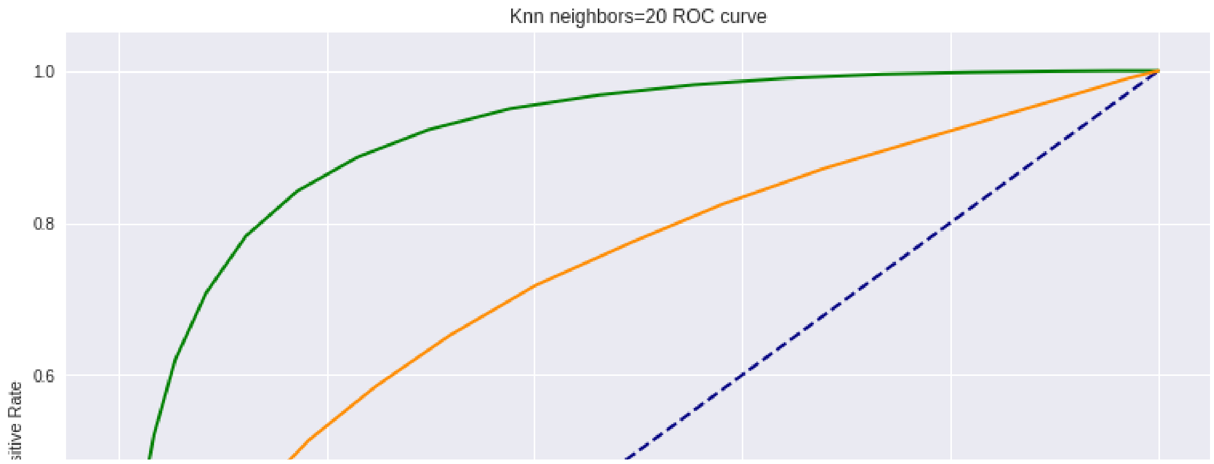
```
', 'rb') as bow:
    train_data = pickle.load(bow)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_vec_cv.pkl", '
rb') as bow:
    cv_data = pickle.load(bow)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_vec_test.pkl",
'rb') as bow:
    test_data = pickle.load(bow)
```

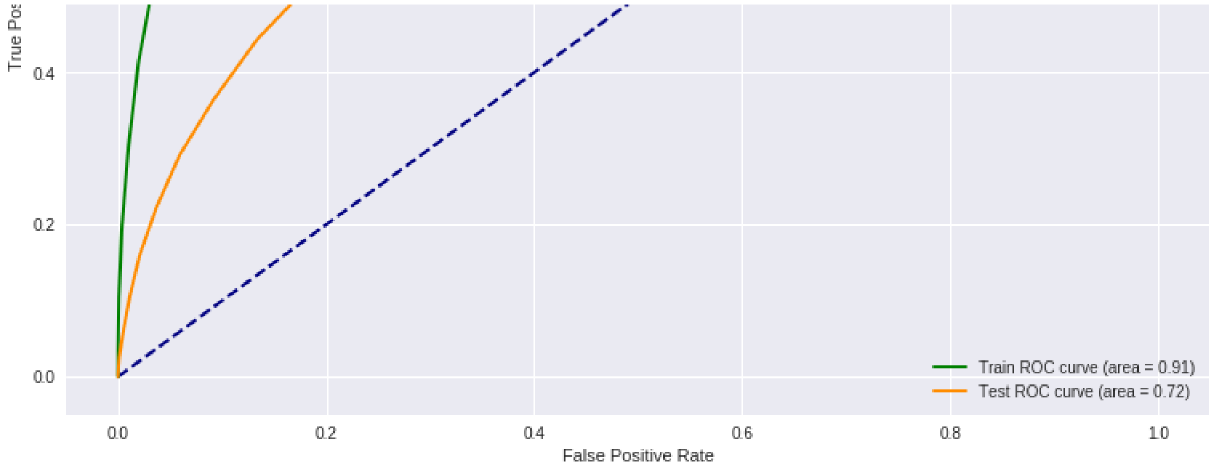
In [0]:

```
# 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]:

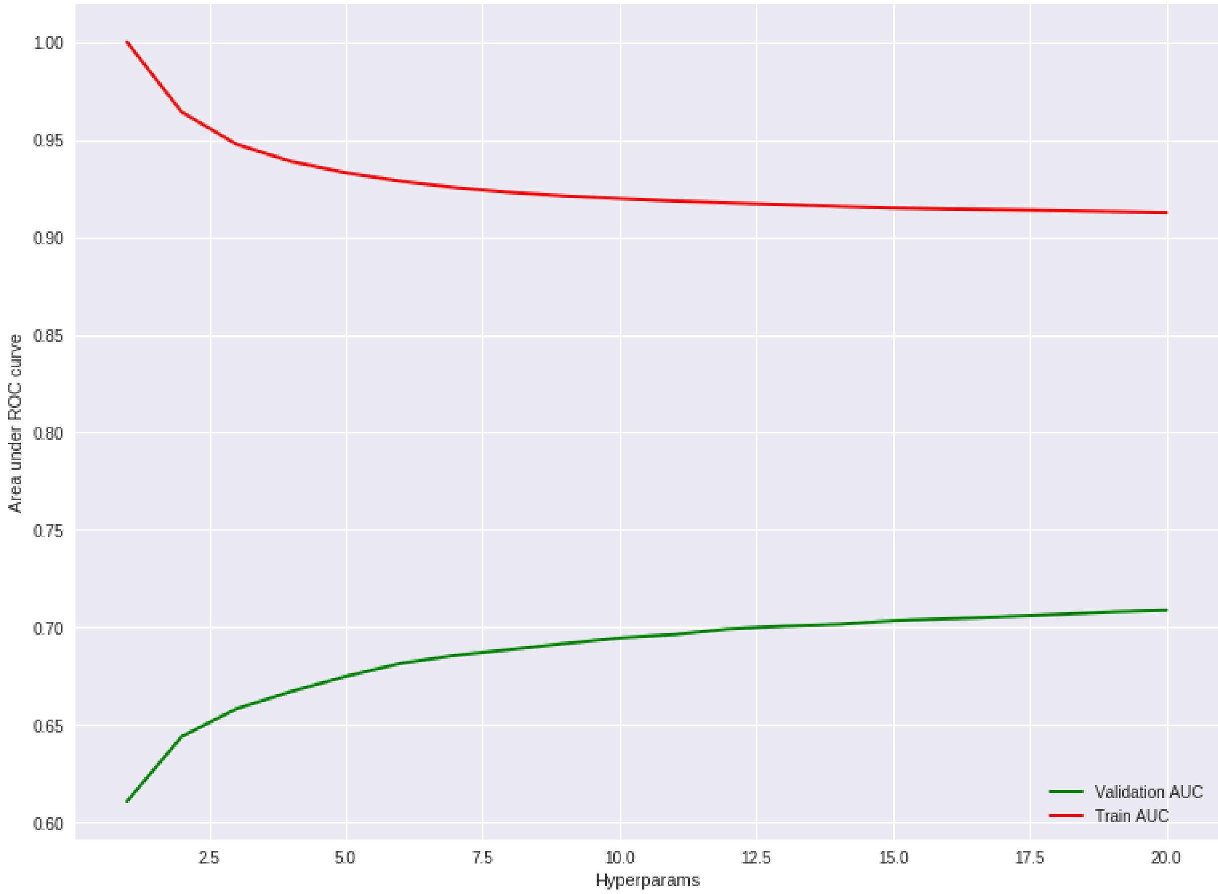
```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_cv.index(max(auc_cv))
plt.plot(fpr_train[max_idx], tpr_train[max_idx], color='green', lw=lw, label='Train ROC curve
(area = %0.2f)' % auc_train[max_idx])
plt.plot(fpr_test[max_idx], tpr_test[max_idx], color='darkorange', lw=lw, label='Test ROC curve
(area = %0.2f)' % auc_test[max_idx])
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/plot_roc.html

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}
```

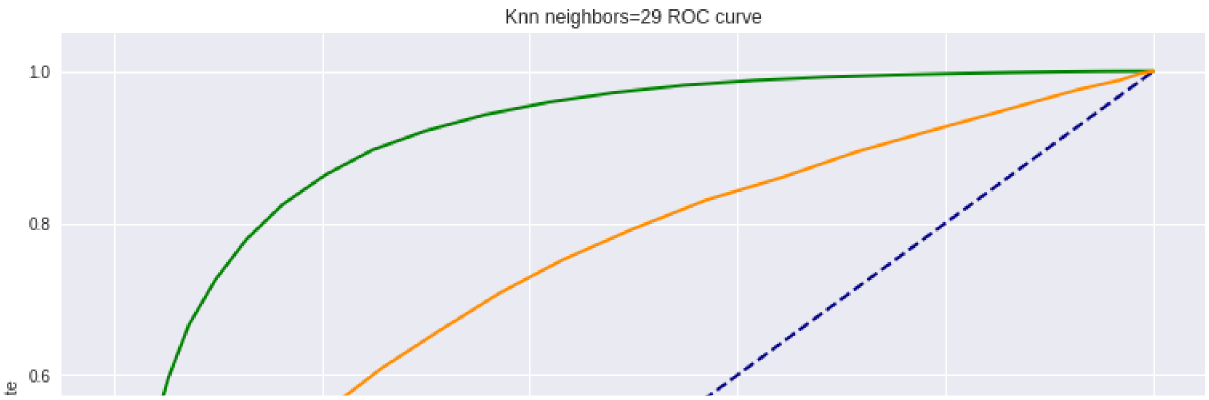
In [0]:

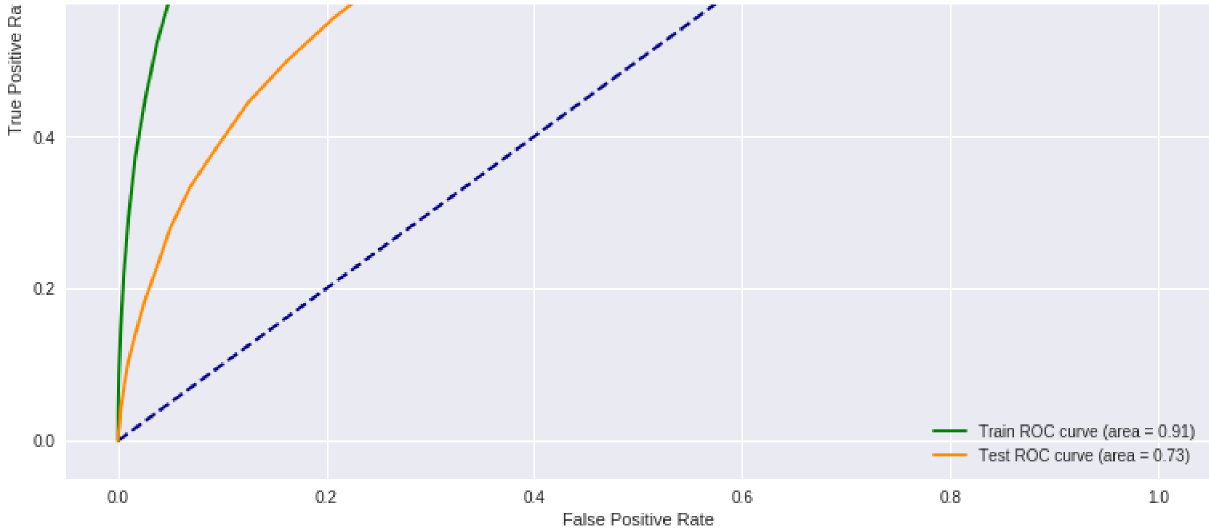
```
# 29-NN
knn_classifier = KNeighborsClassifier(n_neighbors=29, algorithm='brute')
knn_classifier.fit(train_data, bow_train_lab)
cv_predict = knn_classifier.predict(cv_data)
print(classification_report(bow_cv_lab, cv_predict))
train_proba = knn_classifier.predict_proba(train_data)
fpr_train, tpr_train, _ = 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/plot_roc.html
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 = %0.2f)' % auc_t
rain)
plt.plot(fpr_test, tpr_test, color='darkorange', lw=lw, label='Test ROC curve (area = %0.2f)' % auc
_test)
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()
```





[5.1.3] Word2Vec

In [0]:

```
#loading word2vec vectors
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/avg_w2v_train.pkl",
'rb') as w2v_pickle:
    train_data = pickle.load(w2v_pickle)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/avg_w2v_cv.pkl", 'rb
') as w2v_pickle:
    cv_data = pickle.load(w2v_pickle)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/avg_w2v_test.pkl", '
rb') as w2v_pickle:
    test_data = pickle.load(w2v_pickle)
```

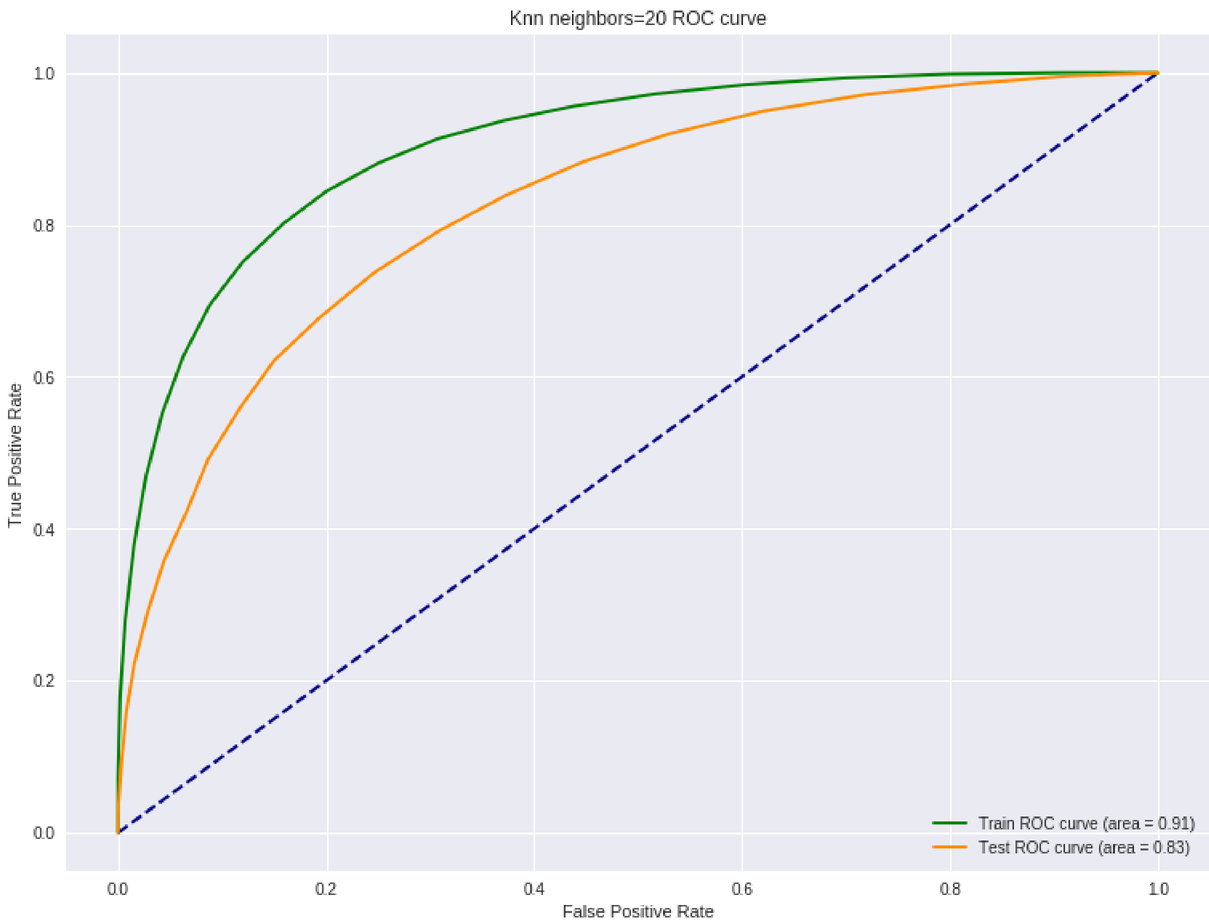
In [0]:

```
# 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]:

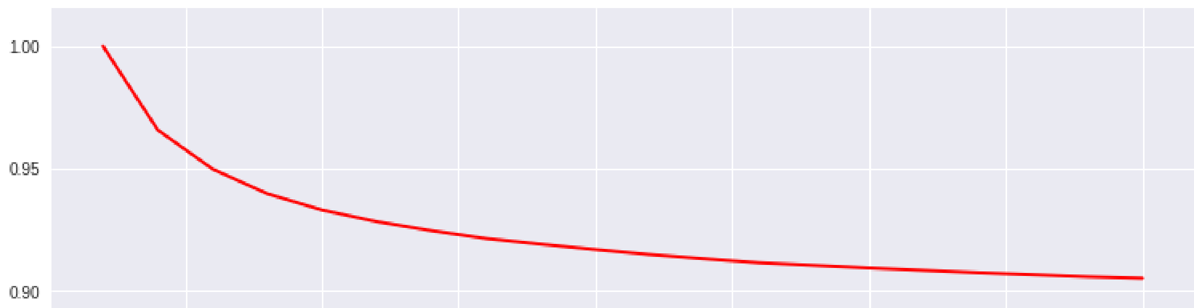
```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
plt.figure(figsize=(12.8, 9.6))
```

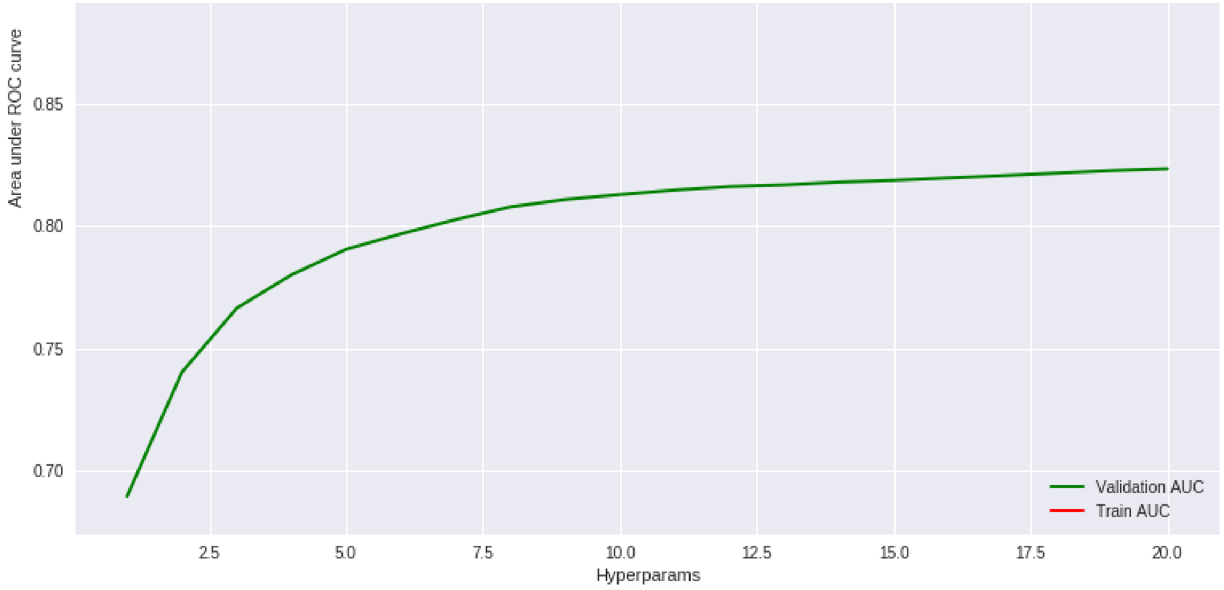
```
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_cv.index(max(auc_cv))
plt.plot(fpr_train[max_idx], tpr_train[max_idx], color='green', lw=lw, label='Train ROC curve
(area = %0.2f)' % auc_train[max_idx])
plt.plot(fpr_test[max_idx], tpr_test[max_idx], color='darkorange', lw=lw, label='Test ROC curve
(area = %0.2f)' % auc_test[max_idx])
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/plot_roc.html

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]:

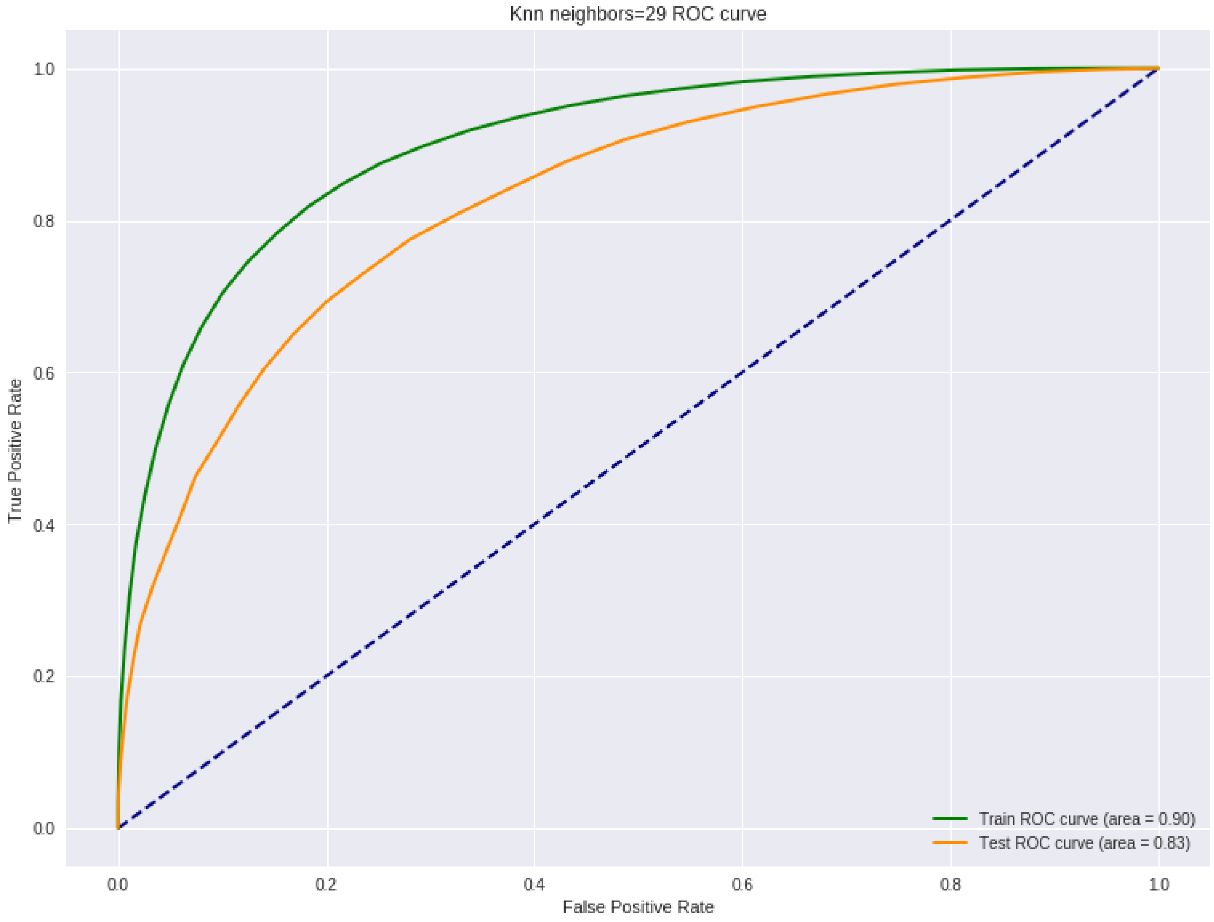
```
# 29-NN
knn_classifier = KNeighborsClassifier(n_neighbors=29, algorithm='brute')
knn_classifier.fit(train_data, bow_train_lab)
cv_predict = knn_classifier.predict(cv_data)
print(classification_report(bow_cv_lab, cv_predict))
train_proba = knn_classifier.predict_proba(train_data)
fpr_train, tpr_train,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

```
lw=2
```

```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
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 = %0.2f)' % auc_t
rain)
plt.plot(fpr_test, tpr_test, color='darkorange', lw=lw, label='Test ROC curve (area = %0.2f)' % auc
_test)
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()
```



[5.1.4] TFIDF Word2Vec

In [0]:

```
#loading tfidf word2vec

with open("/content/gdrive/My
Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_weighted_w2v_train.pkl", 'rb') as
tfidf_w2v_pickle:
    train_data = pickle.load(tfidf_w2v_pickle)
with open("/content/gdrive/My
Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_weighted_w2v_cv.pkl", 'rb') as
tfidf_w2v_pickle:
    cv_data = pickle.load(tfidf_w2v_pickle)
with open("/content/gdrive/My
Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_weighted_w2v_test.pkl", 'rb') as
tfidf_w2v_pickle:
    test_data = pickle.load(tfidf_w2v_pickle)
```

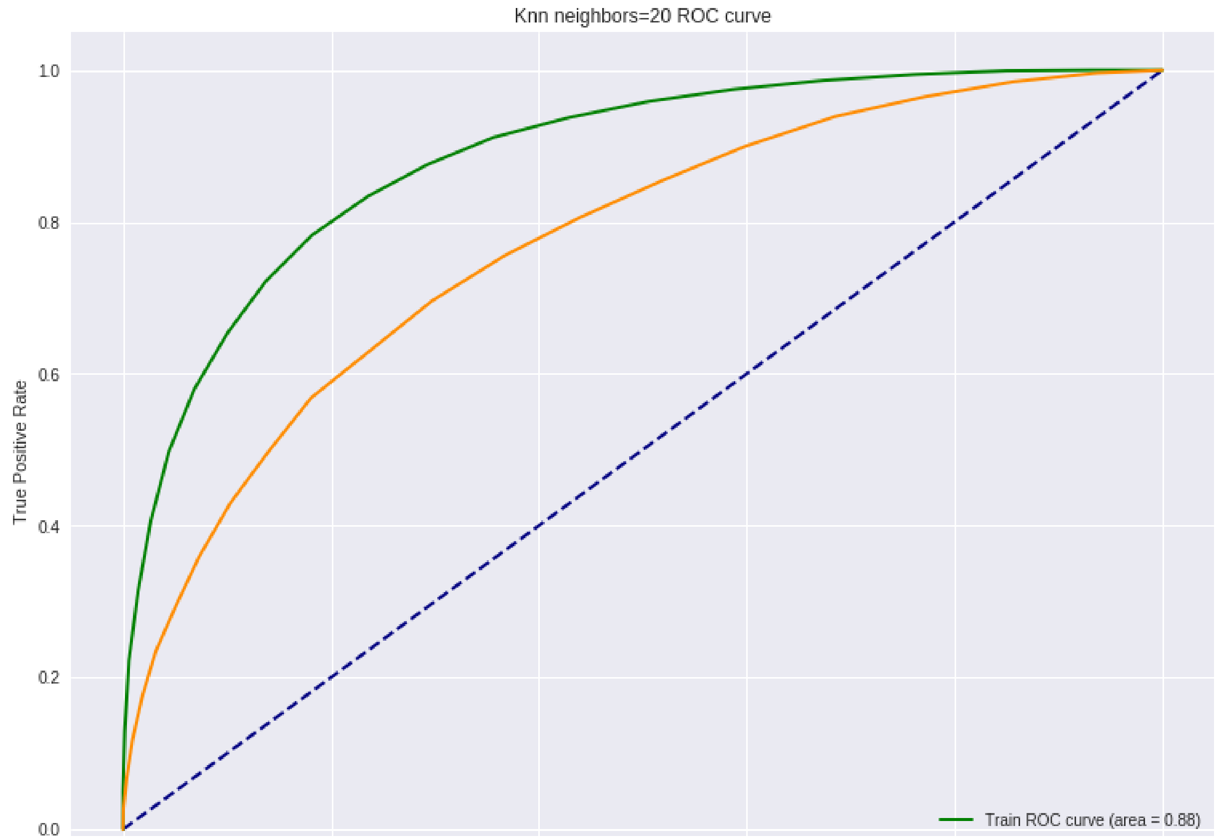
In [0]:

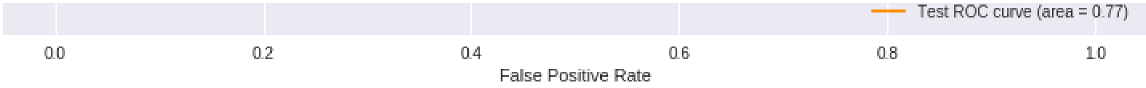
```
# finding best k using AUC
lw = 2
auc_train = []
```

```
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]:

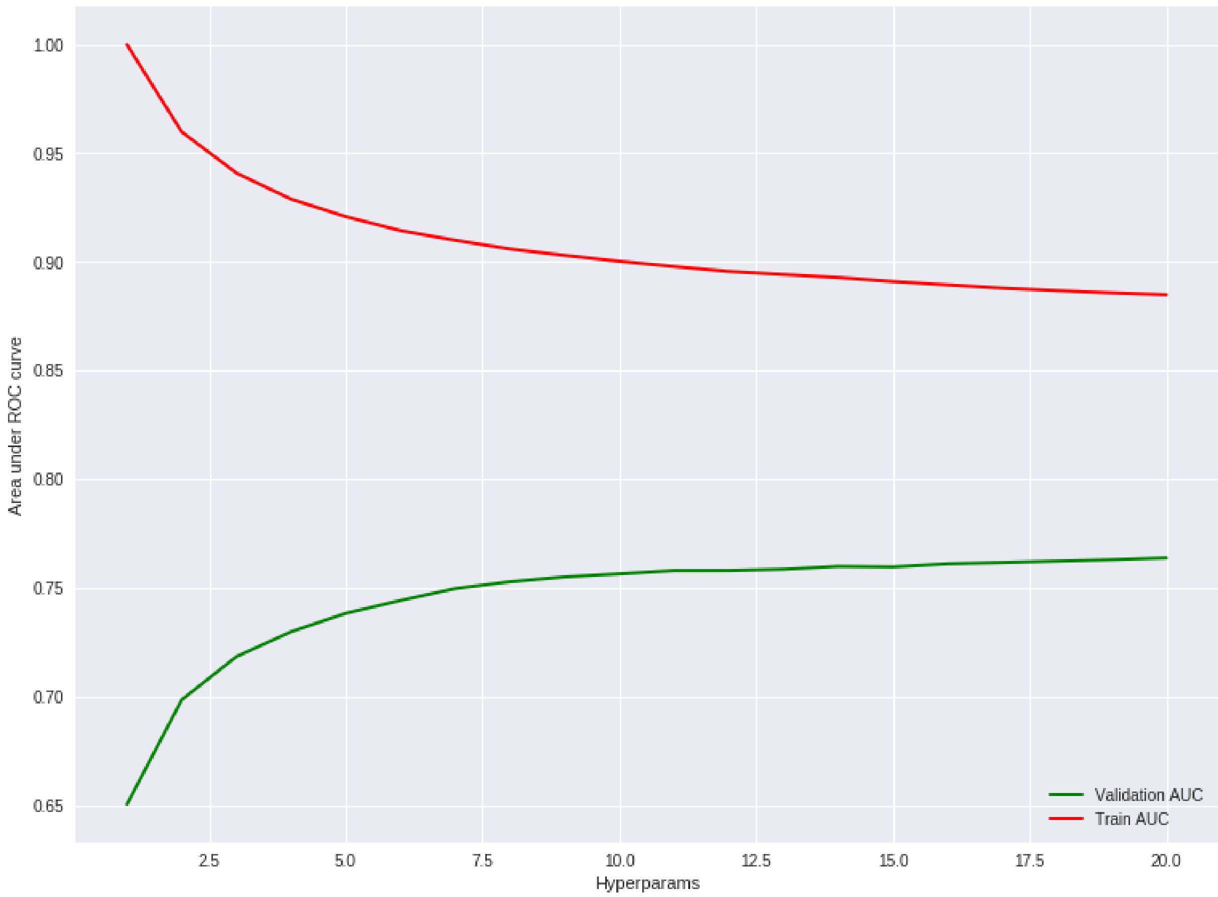
```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_cv.index(max(auc_cv))
plt.plot(fpr_train[max_idx], tpr_train[max_idx], color='green', lw=lw, label='Train ROC curve
(area = %0.2f)' % auc_train[max_idx])
plt.plot(fpr_test[max_idx], tpr_test[max_idx], color='darkorange', lw=lw, label='Test ROC curve
(area = %0.2f)' % auc_test[max_idx])
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/plot_roc.html

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}
```

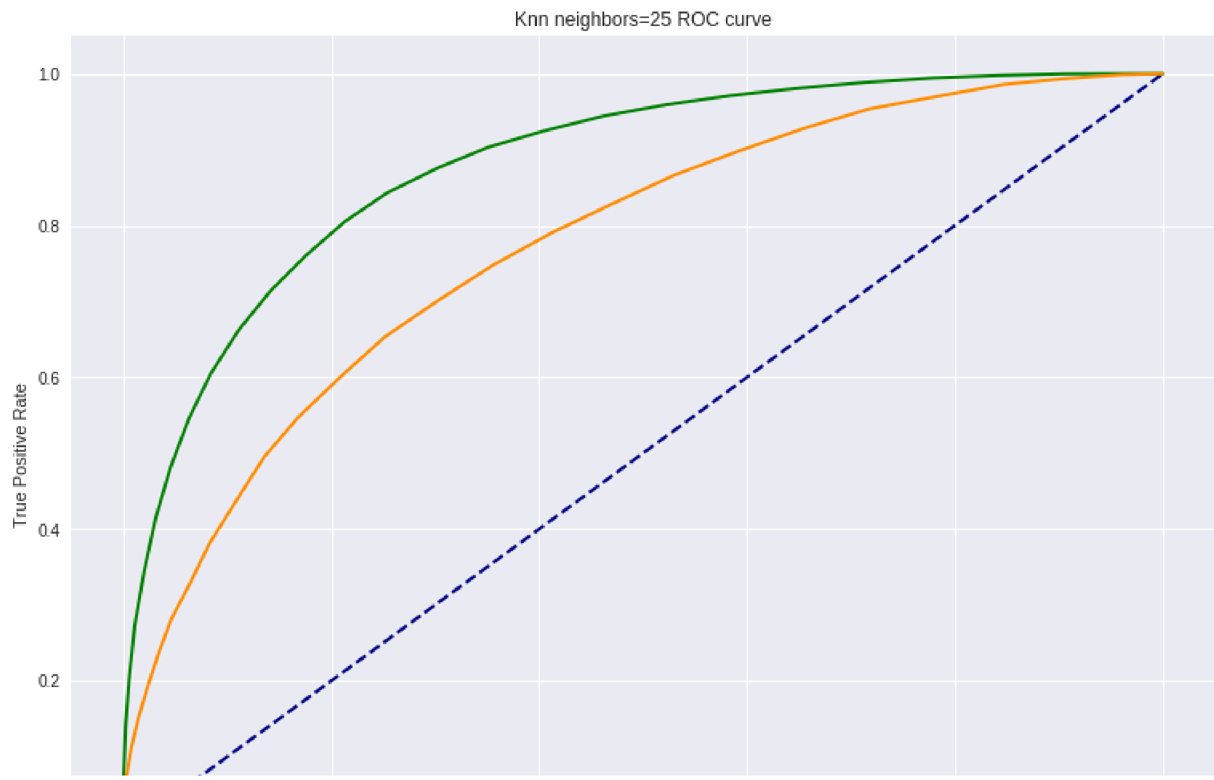
In [0]:

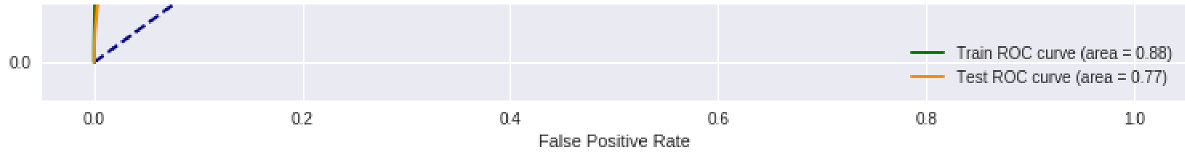
```
# 27-NN
knn_classifier = KNeighborsClassifier(n_neighbors=27, algorithm='brute')
knn_classifier.fit(train_data, train_lab_bin)
cv_predict = knn_classifier.predict(cv_data)
print(classification_report(cv_lab_bin, cv_predict))
train_proba = knn_classifier.predict_proba(train_data)
fpr_train, tpr_train, _ = 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/plot_roc.html
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 = %0.2f)' % auc_t
rain)
plt.plot(fpr_test, tpr_test, color='darkorange', lw=lw, label='Test ROC curve (area = %0.2f)' % auc
_test)
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()
```





[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]
```

[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]:

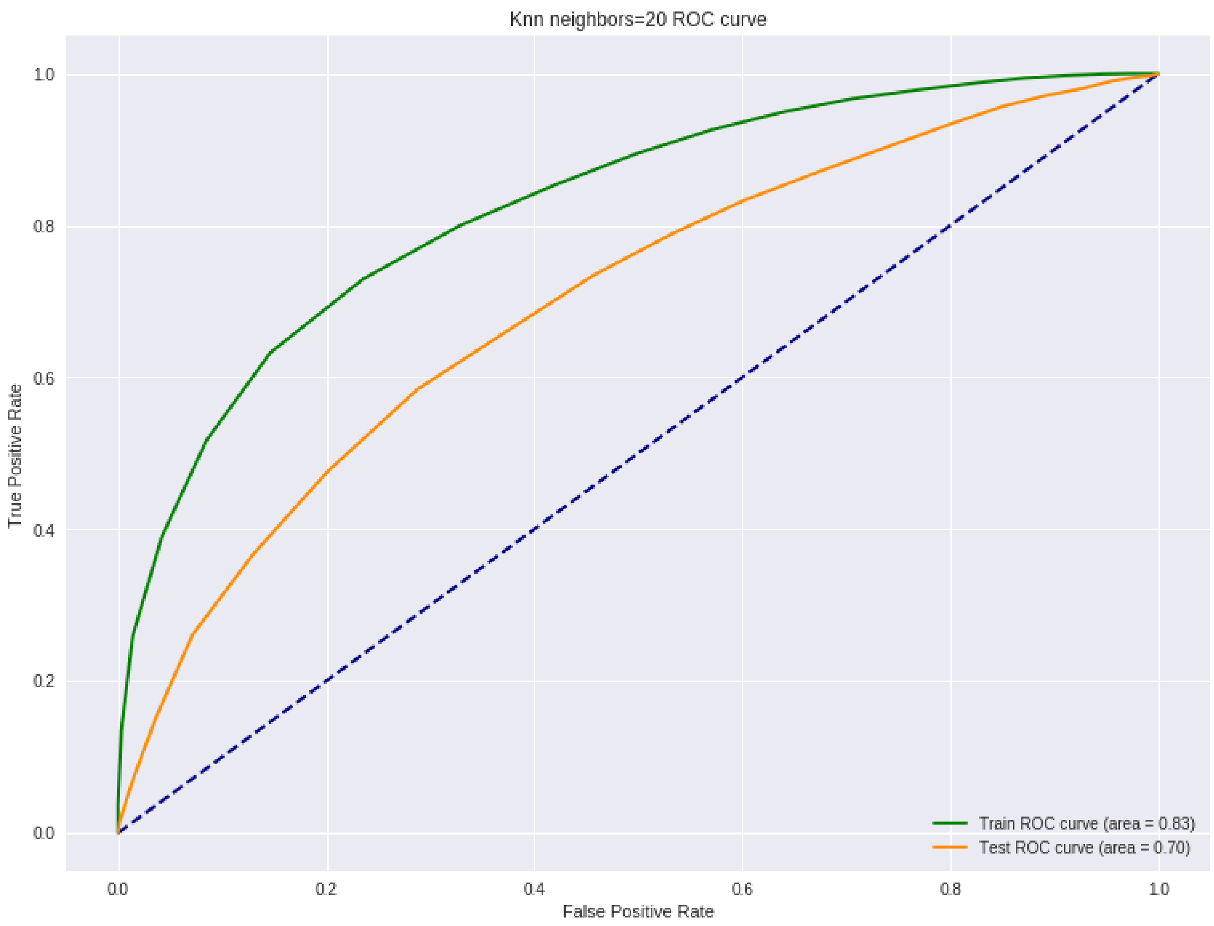
```
# finding best k using AUC
lw = 2
auc_train = []
auc_cv = []
auc_test = []
fpr_train = dict()
tpr_train = dict()
fpr_test = dict()
tpr_test = dict()
fpr_cv = dict()
tpr_cv = dict()

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]:

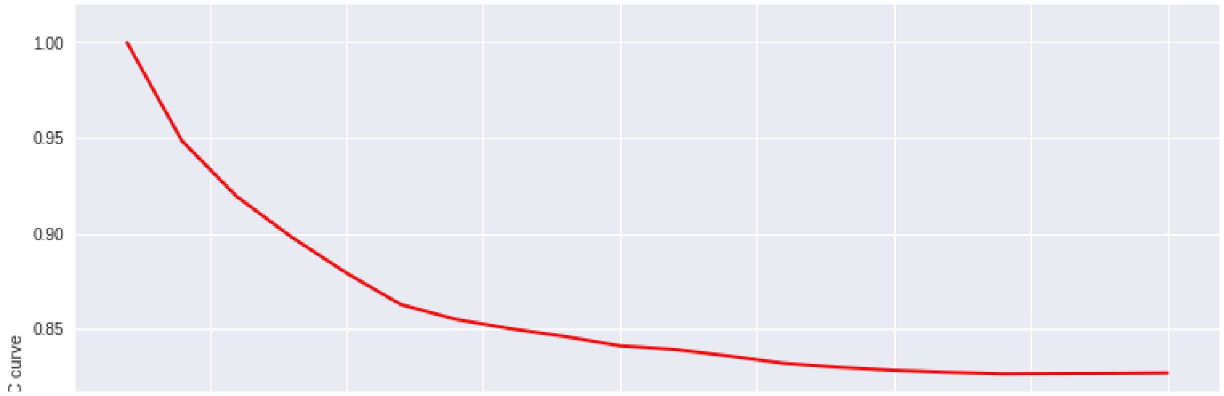
```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_cv.index(max(auc_cv))
plt.plot(fpr_train[max_idx], tpr_train[max_idx], color='green', lw=lw, label='Train ROC curve
```

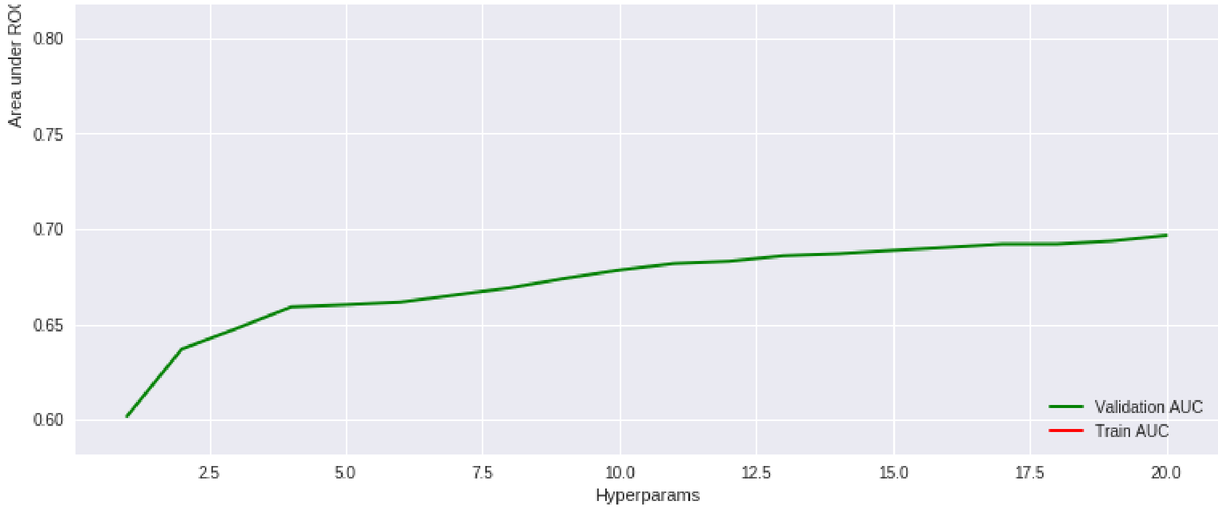
```
(area = %0.2f)' % auc_train[max_idx])
plt.plot(fpr_test[max_idx], tpr_test[max_idx], color='darkorange', lw=lw, label='Test ROC curve
(area = %0.2f)' % auc_test[max_idx])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
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/plot_roc.html

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]:

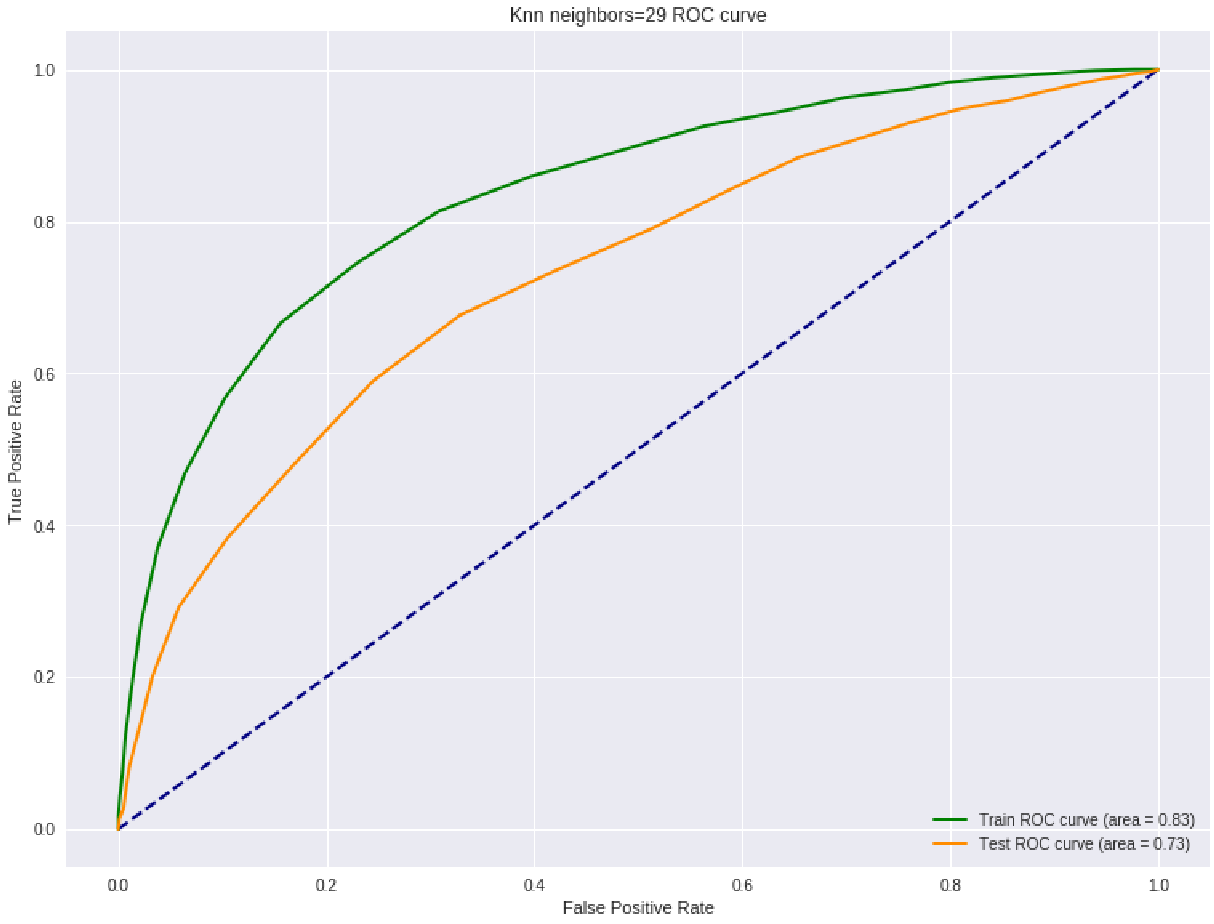
```
# 28-NN
knn_classifier = KNeighborsClassifier(n_neighbors=28, algorithm='brute')
knn_classifier.fit(train_data, train_lab_bin)
cv_predict = knn_classifier.predict(cv_data)
print(classification_report(cv_lab_bin, cv_predict))
train_proba = knn_classifier.predict_proba(train_data)
fpr_train, tpr_train, _ = 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/plot_roc.html
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
```

```
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 = %0.2f)' % auc_t
rain)
plt.plot(fpr_test, tpr_test, color='darkorange', lw=lw, label='Test ROC curve (area = %0.2f)' % auc
_test)
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()
```



[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.get_shape()[1
])
```

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

In [0]:

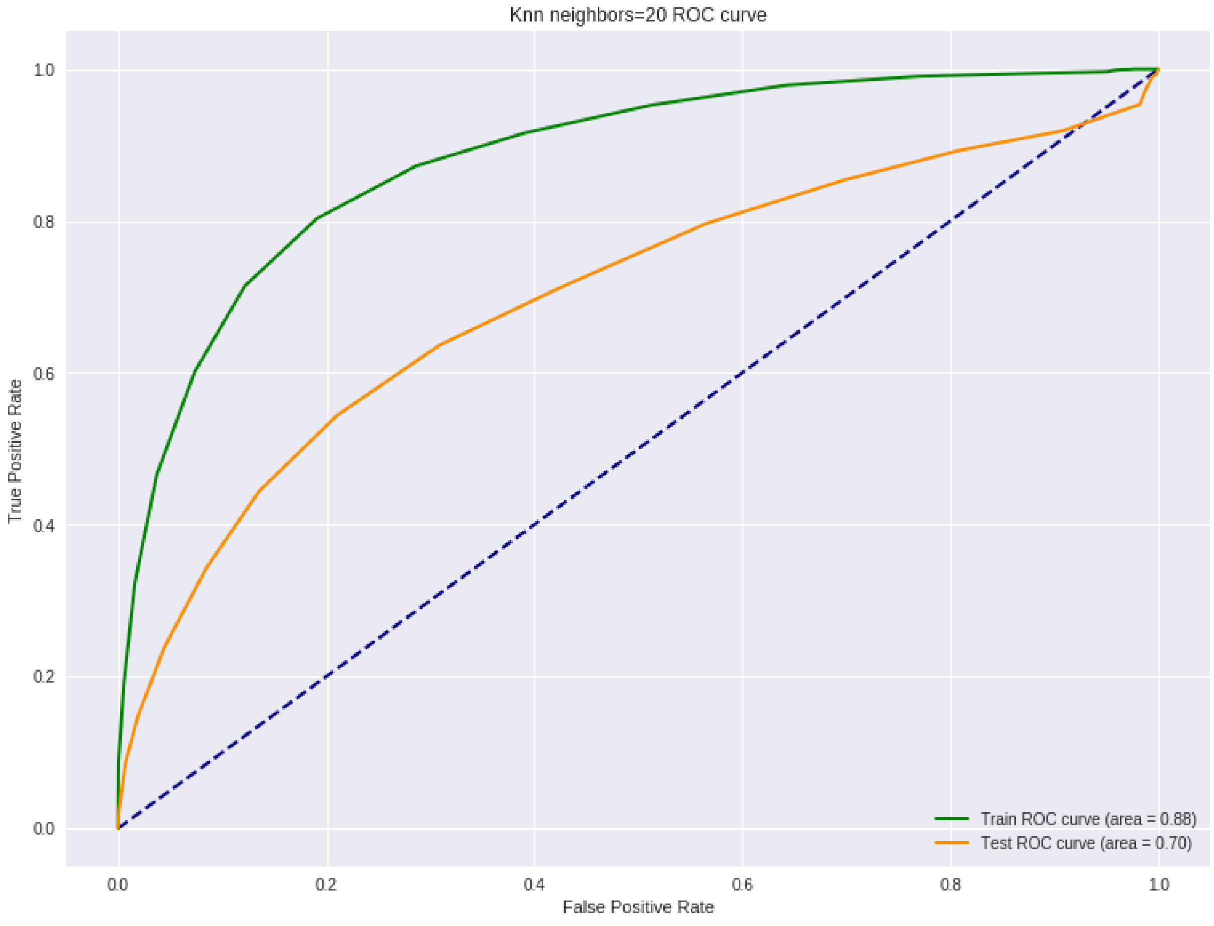
```
# finding best k using AUC
lw = 2
auc_train = []
auc_cv = []
auc_test = []
fpr_train = dict()
tpr_train = dict()
fpr_test = dict()
```

```
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
```

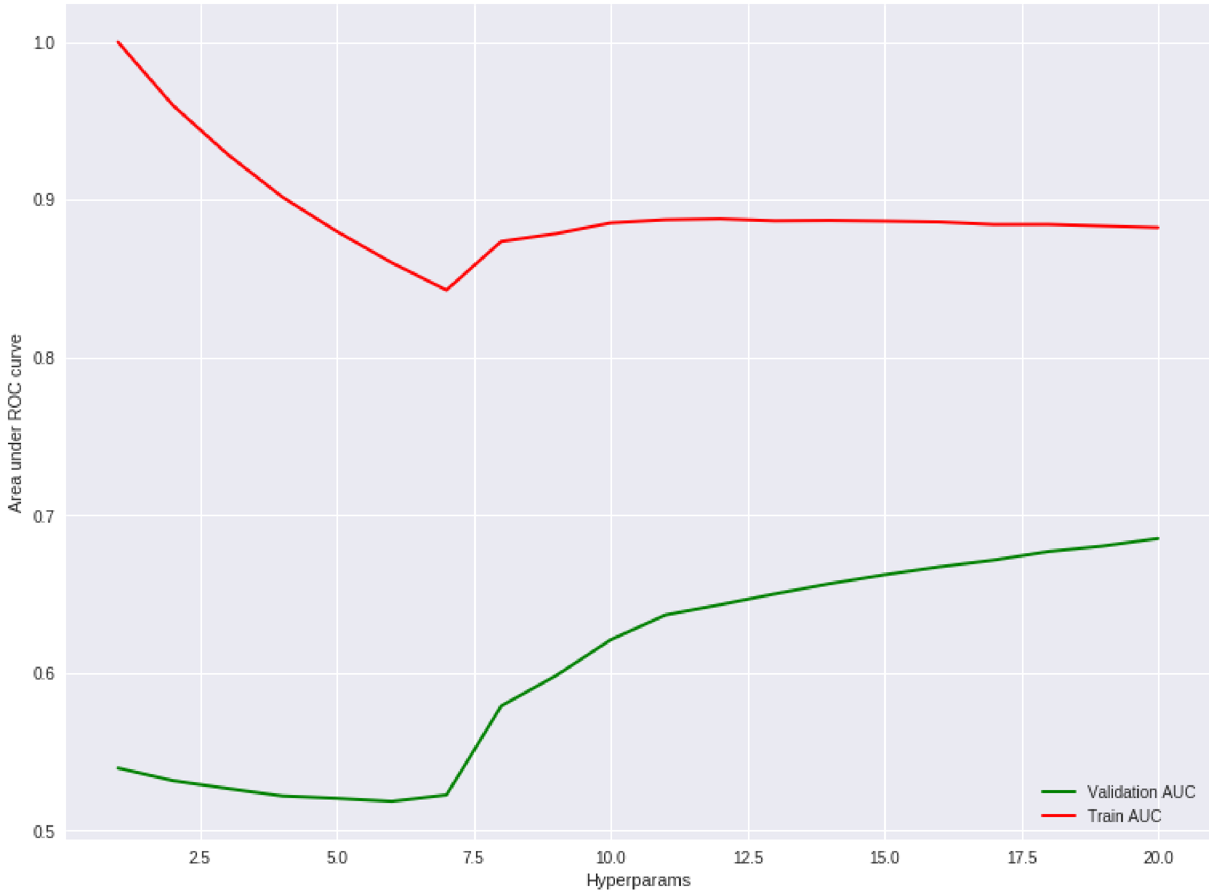
In [0]:

```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
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 curve
(area = %0.2f)' % auc_train[max_idx])
plt.plot(fpr_test[max_idx], tpr_test[max_idx], color='darkorange', lw=lw, label='Test ROC curve
(area = %0.2f)' % auc_test[max_idx])
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/plot_roc.html

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]:

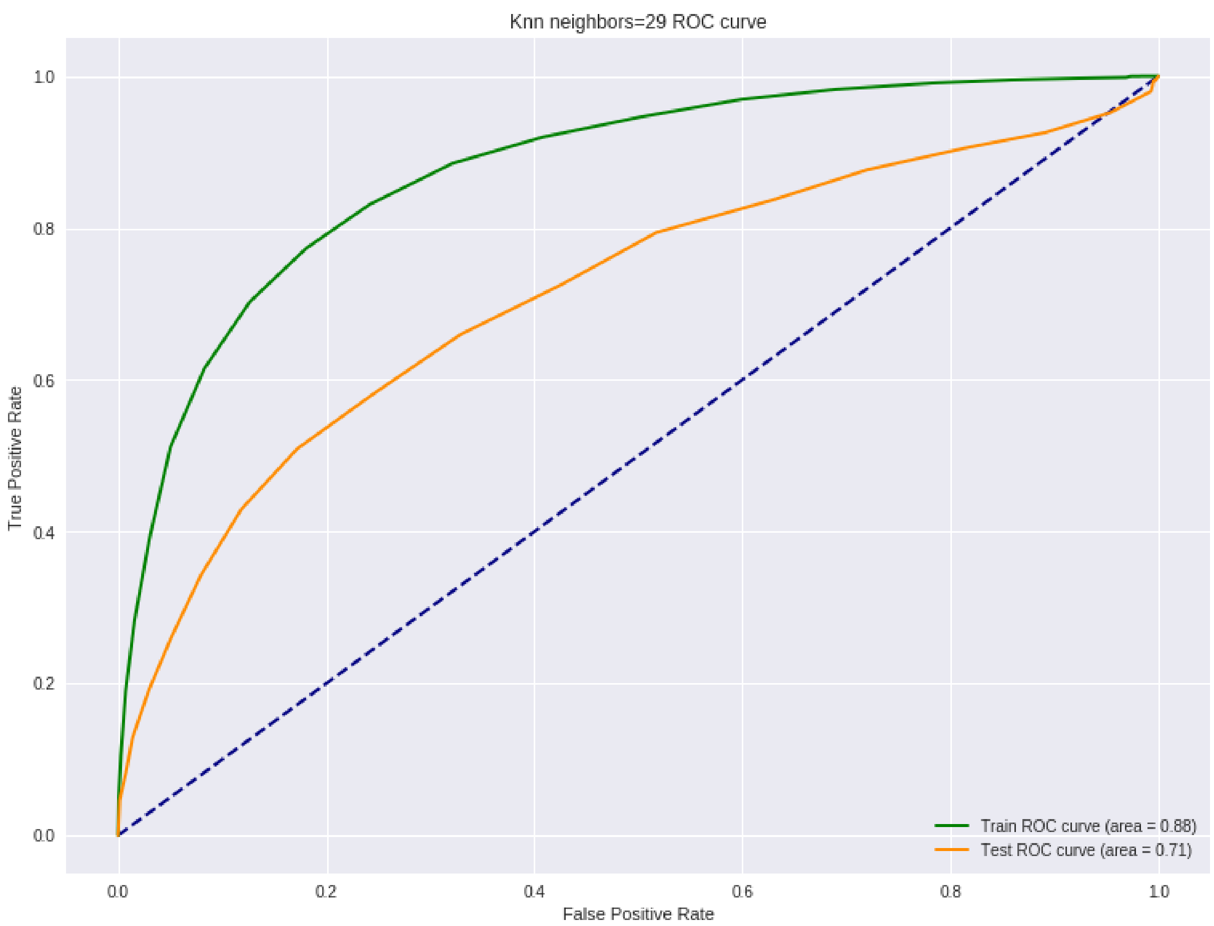
```
# 25-NN
```

```
# 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/plot_roc.html
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 = %0.2f)' % auc_t
rain)
plt.plot(fpr_test, tpr_test, color='darkorange', lw=lw, label='Test ROC curve (area = %0.2f)' % auc
_test)
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()
```



[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
```

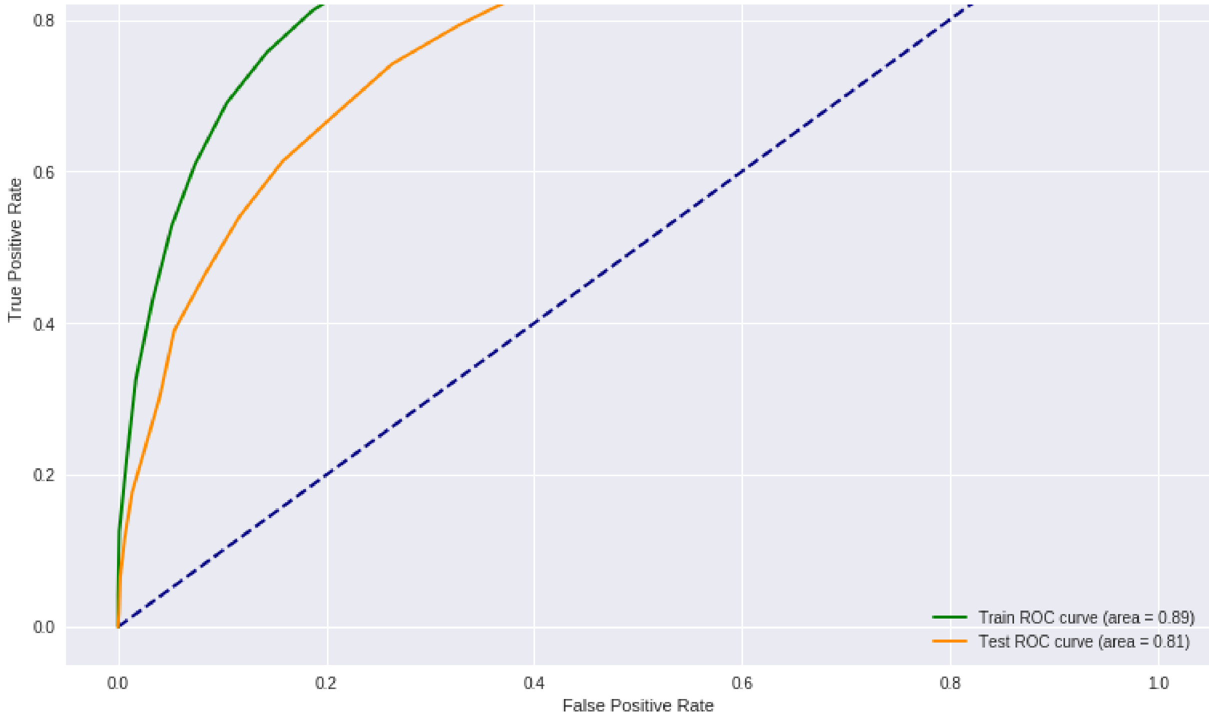
In [0]:

```
# 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
```

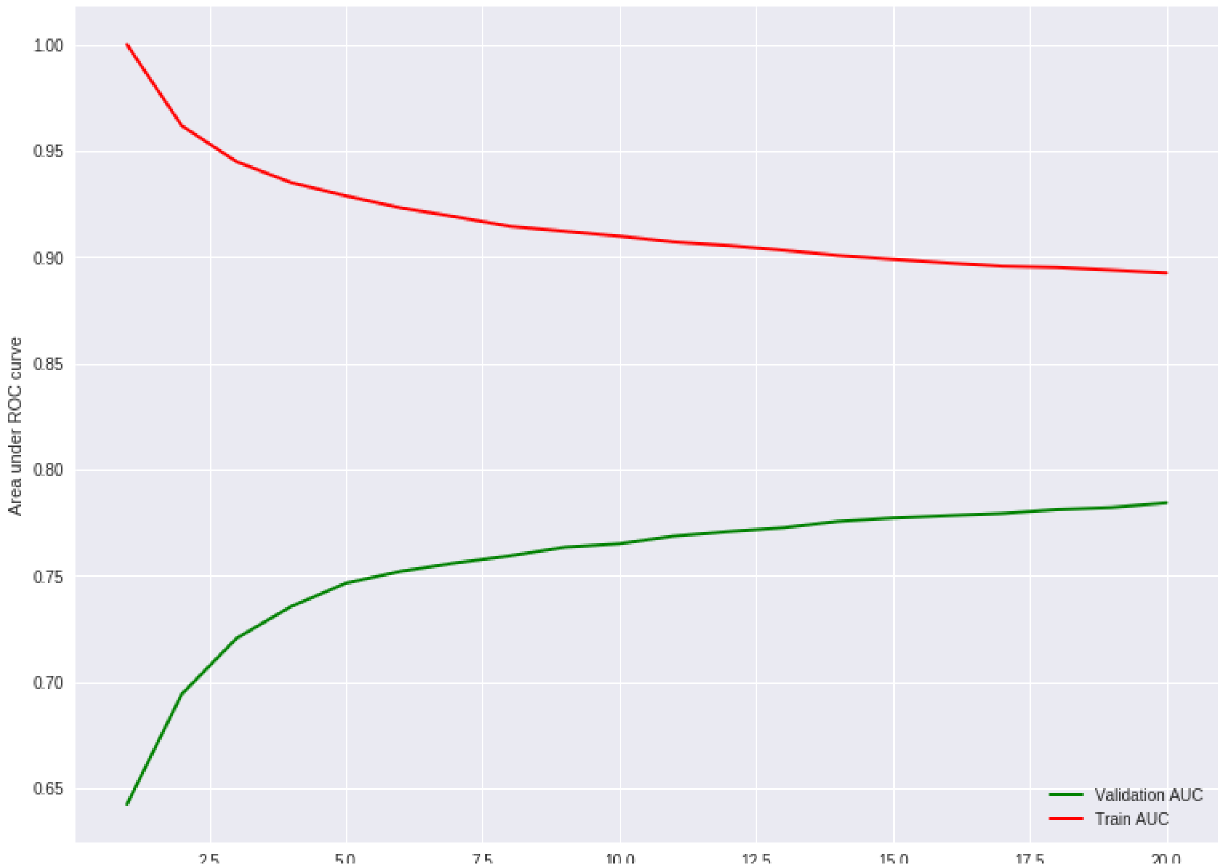
In [0]:

```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
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 curve
(area = %0.2f)' % auc_train[max_idx])
plt.plot(fpr_test[max_idx], tpr_test[max_idx], color='darkorange', lw=lw, label='Test ROC curve
(area = %0.2f)' % auc_test[max_idx])
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/plot_roc.html

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()
```



Hyperparams

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]:

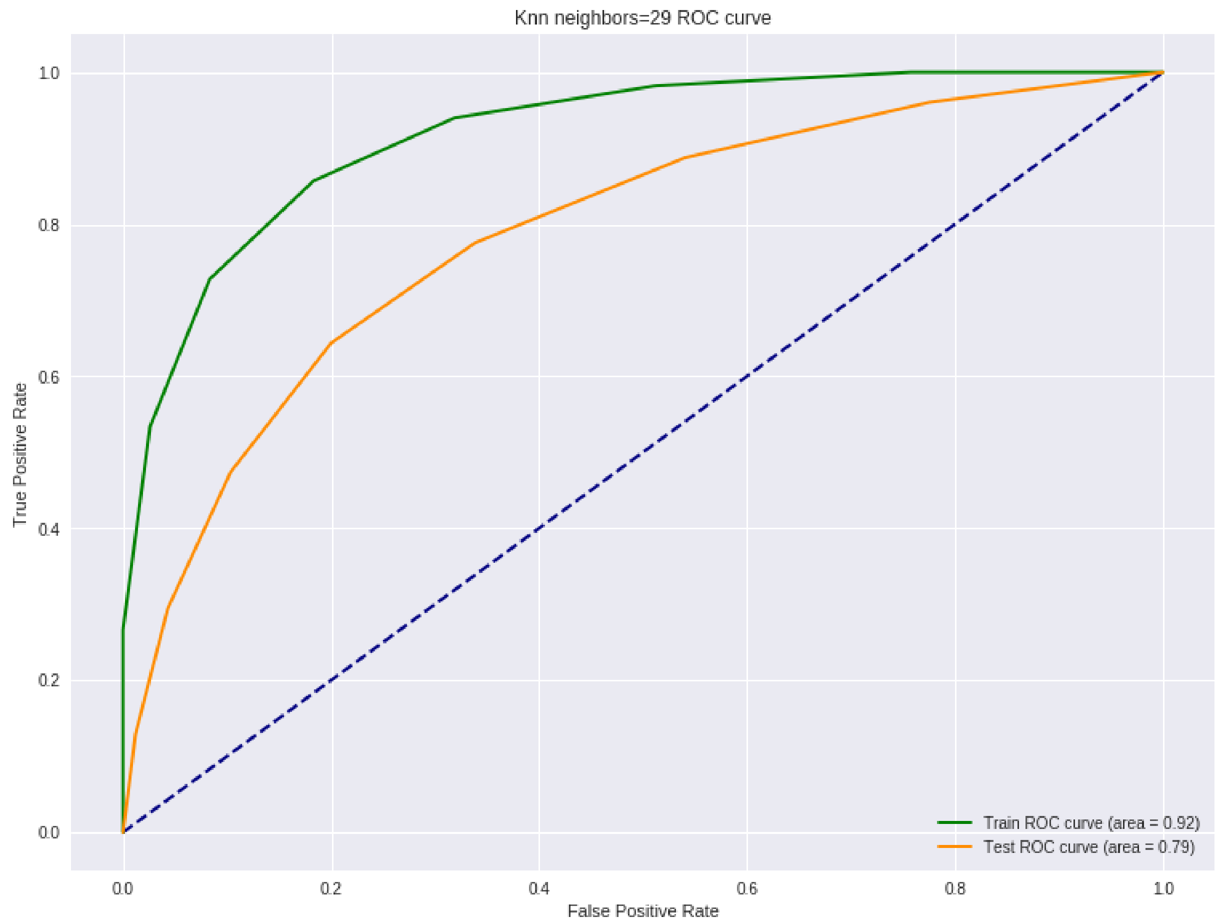
```
# 7-NN
knn_classifier = KNeighborsClassifier(n_neighbors=7, algorithm='kd_tree')
knn_classifier.fit(train_data, train_lab_bin)
cv_predict = knn_classifier.predict(cv_data)
print(classification_report(cv_lab_bin, cv_predict))
train_proba = knn_classifier.predict_proba(train_data)
fpr_train, tpr_train, _ = 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/plot_roc.html
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 = %0.2f)' % auc_t
rain)
plt.plot(fpr_test, tpr_test, color='darkorange', lw=lw, label='Test ROC curve (area = %0.2f)' % auc
_test)
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()
```



[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]:

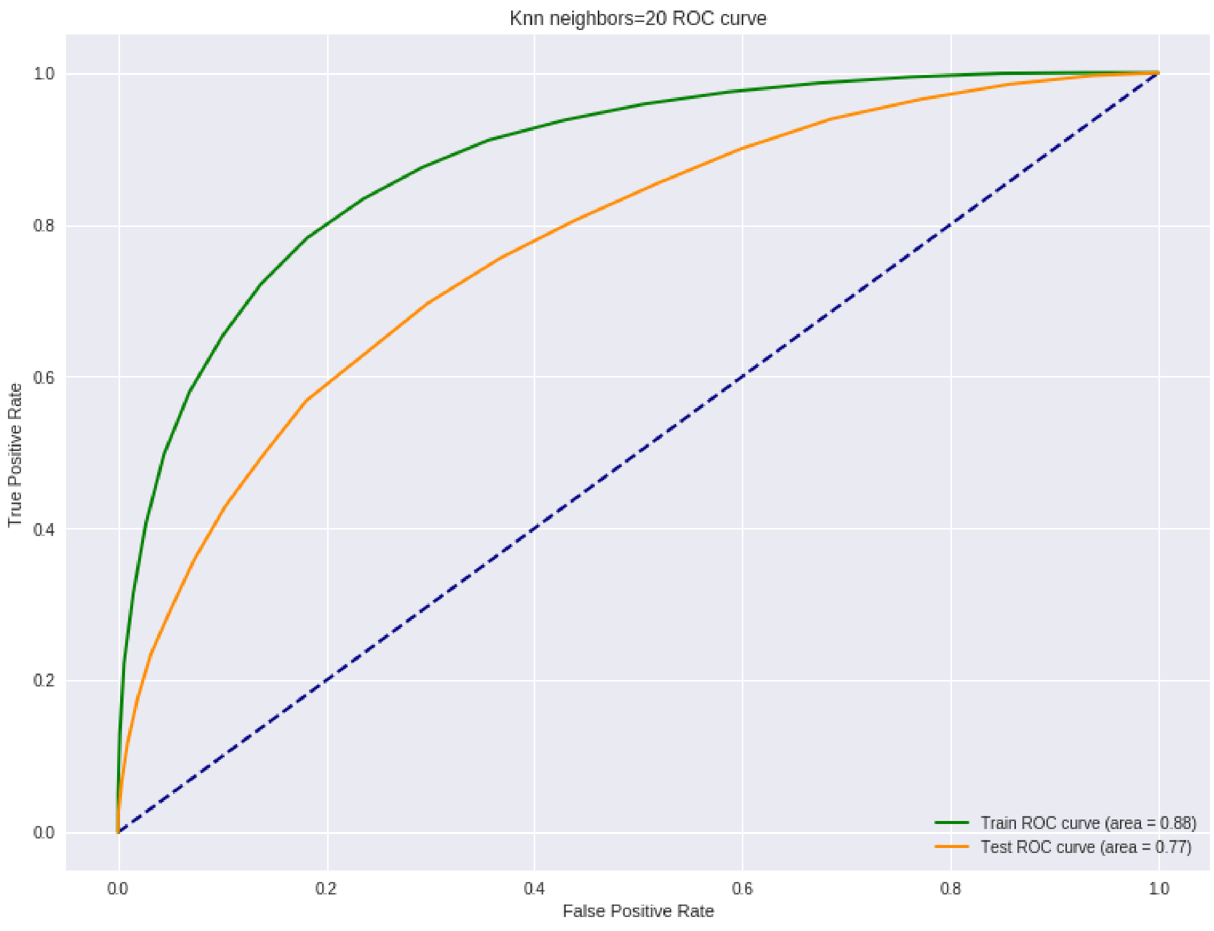
```
# finding best k using AUC
lw = 2
auc_train = []
auc_cv = []
auc_test = []
fpr_train = dict()
```

```
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
```

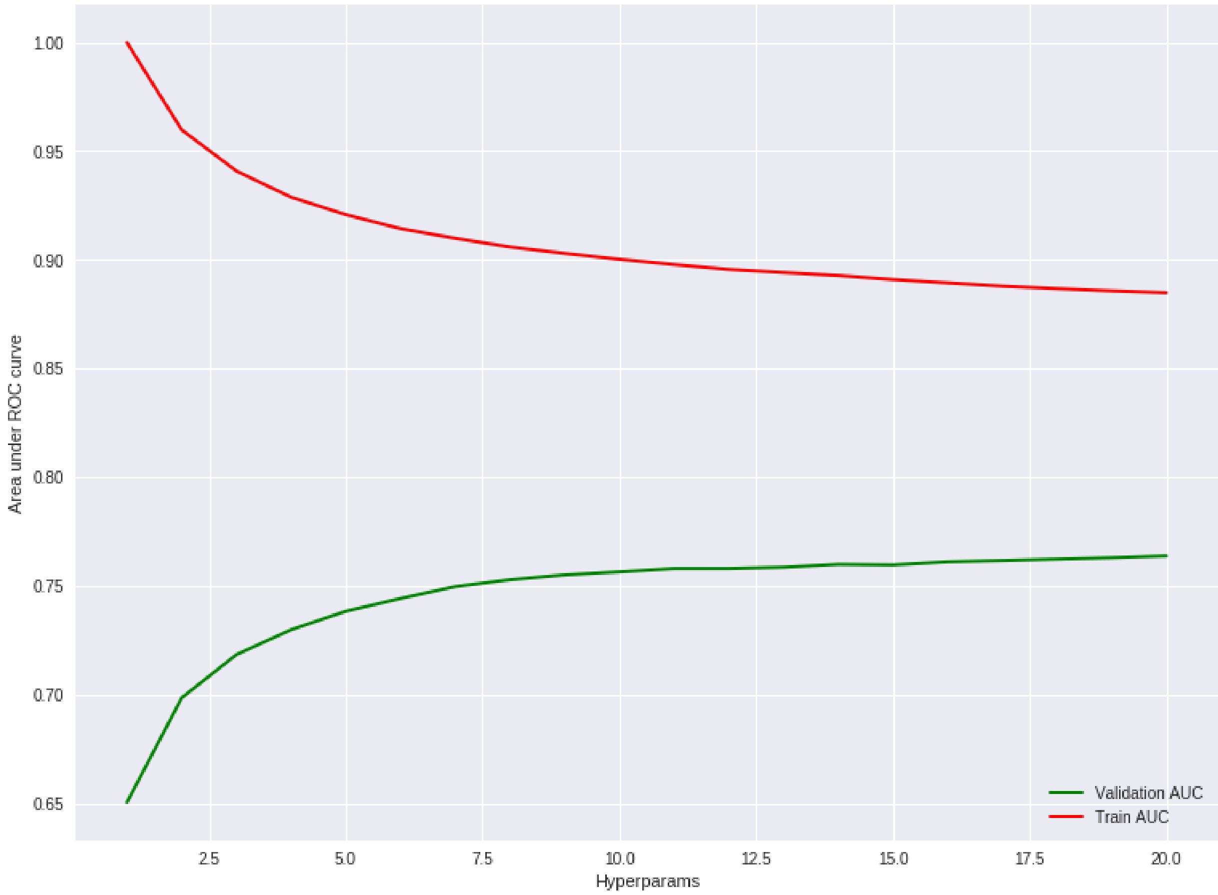
In [0]:

```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
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 curve
(area = %0.2f)' % auc_train[max_idx])
plt.plot(fpr_test[max_idx], tpr_test[max_idx], color='darkorange', lw=lw, label='Test ROC curve
(area = %0.2f)' % auc_test[max_idx])
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/plot_roc.html

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}
```

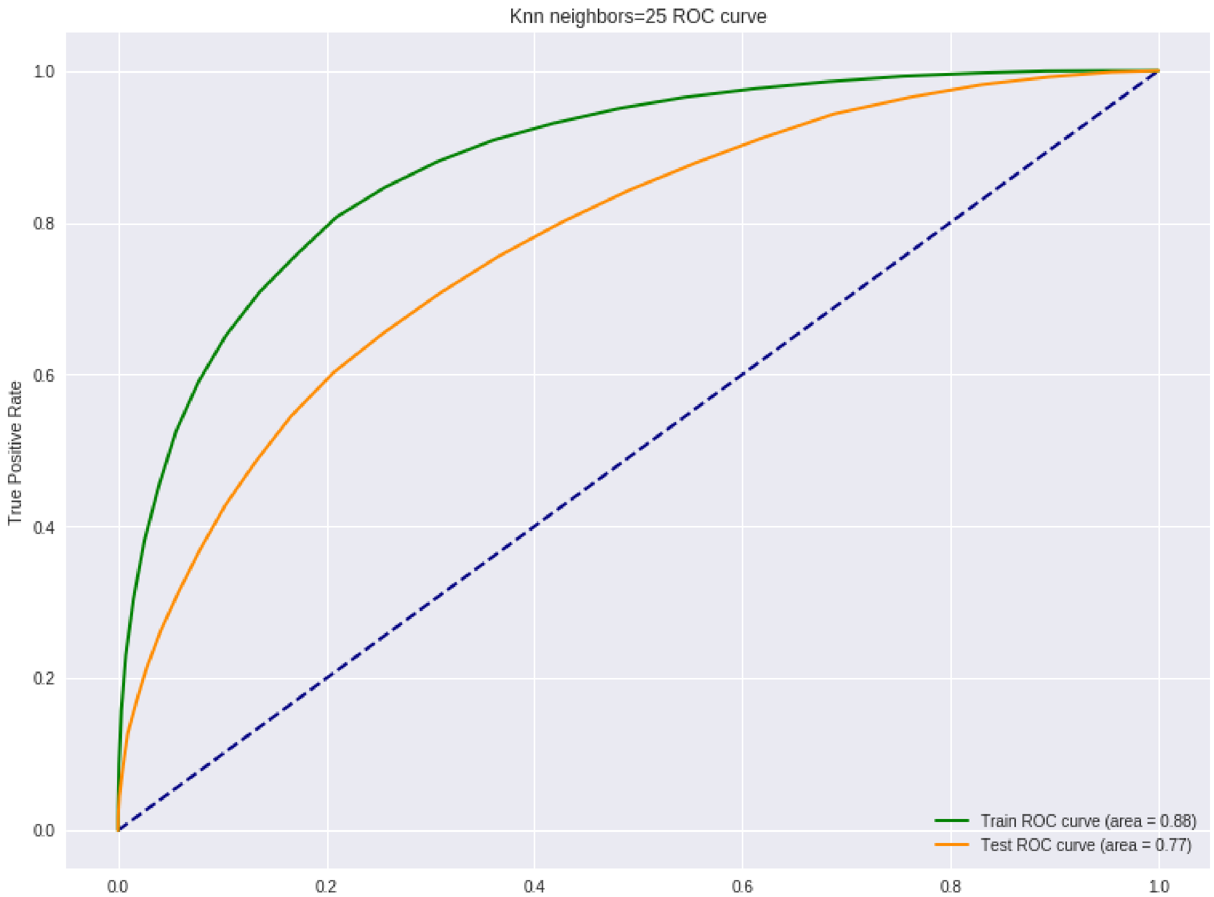
In [0]:

```
# 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/plot_roc.html
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 = %0.2f)' % auc_t
rain)
plt.plot(fpr_test, tpr_test, color='darkorange', lw=lw, label='Test ROC curve (area = %0.2f)' % auc
_test)
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()
```



False Positive Rate

[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   |
+----------------+---------+-----------------+----------+
```