Amazon Food Reviews

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

This dataset consists of reviews of fine foods from Amazon. The data span a period of more than 10 years, including all ~500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories.

Data includes:

- Reviews from Oct 1999 - Oct 2012
- 568,454 reviews
- 256,059 users
- 74,258 products
- 260 users with > 50 reviews

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

Number of people who found the review helpful

Number of people who indicated whether or not the review was helpful

Summary

Objective:- Review Polarity

Given a review, determine the review is positive or neagative

Using text review to decide the polarity

Take the summary and text of review and analyze it using NLP whether the customer feedback/review is positive or negative

In [2]:

```
!pip install --upgrade pip
!pip install qtconsole ipywidgets widgetsnbextension
!pip install seaborn
!pip install nltk
# import nltk
# nltk.download("stopwords")
!pip install gensim
```

```
Requirement already up-to-date: pip in /opt/conda/envs/py3.6/lib/python3.6/site-packages (10.0.1)
ipywidgets 7.0.3 has requirement widgetsnbextension~=3.0.0, but you'll have widgetsnbextension 3.2
.1 which is incompatible.
Requirement already satisfied: qtconsole in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(4.3.1)
Requirement already satisfied: ipywidgets in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(7.0.3)
Requirement already satisfied: widgetsnbextension in /opt/conda/envs/py3.6/lib/python3.6/site-
packages (3.2.1)
Requirement already satisfied: pygments in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from
qtconsole) (2.2.0)
Requirement already satisfied: jupyter-core in /opt/conda/envs/py3.6/lib/python3.6/site-packages (
from qtconsole) (4.4.0)
Requirement already satisfied: traitlets in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(from qtconsole) (4.3.2)
Requirement already satisfied: jupyter-client>=4.1 in /opt/conda/envs/py3.6/lib/python3.6/site-
packages (from qtconsole) (5.2.3)
Requirement already satisfied: ipykernel>=4.1 in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(from qtconsole) (4.8.0)
Requirement already satisfied: ipython-genutils in /opt/conda/envs/py3.6/lib/python3.6/site-
packages (from qtconsole) (0.2.0)
Requirement already satisfied: nbformat>=4.2.0 in /opt/conda/envs/py3.6/lib/python3.6/site-
packages (from ipywidgets) (4.4.0)
Requirement already satisfied: ipython>=4.0.0; python_version >= "3.3" in
/opt/conda/envs/py3.6/lib/python3.6/site-packages (from ipywidgets) (6.4.0)
Requirement already satisfied: notebook>=4.4.1 in /opt/conda/envs/py3.6/lib/python3.6/site-
packages (from widgetsnbextension) (5.5.0)
Requirement already satisfied: six in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from
traitlets->qtconsole) (1.11.0)
Requirement already satisfied: decorator in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(from traitlets->qtconsole) (4.3.0)
Requirement already satisfied: pyzmq>=13 in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(from jupyter-client>=4.1->qtconsole) (17.0.0)
Requirement already satisfied: python-dateutil>=2.1 in /opt/conda/envs/py3.6/lib/python3.6/site-
packages (from jupyter-client>=4.1->qtconsole) (2.7.3)
Requirement already satisfied: tornado>=4.1 in /opt/conda/envs/py3.6/lib/python3.6/site-packages
```

```
(from jupyter-client>=4.1->qtconsole) (5.0.2)
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in
/opt/conda/envs/py3.6/lib/python3.6/site-packages (from nbformat>=4.2.0->ipywidgets) (2.6.0)
Requirement already satisfied: pickleshare in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(from ipython>=4.0.0; python_version >= "3.3"->ipywidgets) (0.7.4)
Requirement already satisfied: jedi>=0.10 in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(from ipython>=4.0.0; python_version >= "3.3"->ipywidgets) (0.11.0)
Requirement already satisfied: pexpect; sys_platform != "win32" in
/opt/conda/envs/py3.6/lib/python3.6/site-packages (from ipython>=4.0.0; python_version >= "3.3"->i
pywidgets) (4.3.0)
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.15 in
/opt/conda/envs/py3.6/lib/python3.6/site-packages (from ipython>=4.0.0; python_version >= "3.3"->i
pywidgets) (1.0.15)
Requirement already satisfied: backcall in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from
ipython>=4.0.0; python_version >= "3.3"->ipywidgets) (0.1.0)
Requirement already satisfied: simplegeneric>0.8 in /opt/conda/envs/py3.6/lib/python3.6/site-
packages (from ipython>=4.0.0; python_version >= "3.3"->ipywidgets) (0.8.1)
Requirement already satisfied: setuptools>=18.5 in /opt/conda/envs/py3.6/lib/python3.6/site-
packages (from ipython>=4.0.0; python_version >= "3.3"->ipywidgets) (36.4.0)
Requirement already satisfied: terminado>=0.8.1 in /opt/conda/envs/py3.6/lib/python3.6/site-
packages (from notebook>=4.4.1->widgetsnbextension) (0.8.1)
Requirement already satisfied: Send2Trash in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(from notebook>=4.4.1->widgetsnbextension) (1.5.0)
Requirement already satisfied: nbconvert in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(from notebook>=4.4.1->widgetsnbextension) (5.3.1)
Requirement already satisfied: jinja2 in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from
notebook>=4.4.1->widgetsnbextension) (2.10)
Requirement already satisfied: parso==0.1.* in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(from jedi>=0.10->ipython>=4.0.0; python_version >= "3.3"->ipywidgets) (0.1.1)
Requirement already satisfied: wcwidth in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from
prompt-toolkit<2.0.0,>=1.0.15->ipython>=4.0.0; python_version >= "3.3"->ipywidgets) (0.1.7)
Requirement already satisfied: pandocfilters>=1.4.1 in /opt/conda/envs/py3.6/lib/python3.6/site-
packages (from nbconvert->notebook>=4.4.1->widgetsnbextension) (1.4.2)
Requirement already satisfied: bleach in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from
nbconvert->notebook>=4.4.1->widgetsnbextension) (1.5.0)
Requirement already satisfied: mistune>=0.7.4 in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(from nbconvert->notebook>=4.4.1->widgetsnbextension) (0.8.3)
Requirement already satisfied: entrypoints>=0.2.2 in /opt/conda/envs/py3.6/lib/python3.6/site-
packages (from nbconvert->notebook>=4.4.1->widgetsnbextension) (0.2.3)
Requirement already satisfied: testpath in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from
nbconvert->notebook>=4.4.1->widgetsnbextension) (0.3.1)
Requirement already satisfied: MarkupSafe>=0.23 in /opt/conda/envs/py3.6/lib/python3.6/site-
packages (from jinja2->notebook>=4.4.1->widgetsnbextension) (1.0)
Requirement already satisfied: html5lib!=0.9999,!=0.99999,<0.99999999,>=0.999 in
/opt/conda/envs/py3.6/lib/python3.6/site-packages (from bleach->nbconvert->notebook>=4.4.1-
>widgetsnbextension) (0.9999999)
ipywidgets 7.0.3 has requirement widgetsnbextension~=3.0.0, but you'll have widgetsnbextension 3.2
.1 which is incompatible.
Requirement already satisfied: seaborn in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(0.8.1)
ipywidgets 7.0.3 has requirement widgetsnbextension~=3.0.0, but you'll have widgetsnbextension 3.2
.1 which is incompatible.
Requirement already satisfied: nltk in /opt/conda/envs/py3.6/lib/python3.6/site-packages (3.3)
Requirement already satisfied: six in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from
nltk) (1.11.0)
ipywidgets 7.0.3 has requirement widgetsnbextension~=3.0.0, but you'll have widgetsnbextension 3.2
.1 which is incompatible.
Requirement already satisfied: gensim in /opt/conda/envs/py3.6/lib/python3.6/site-packages (3.5.0)
Requirement already satisfied: smart-open>=1.2.1 in /opt/conda/envs/py3.6/lib/python3.6/site-
packages (from gensim) (1.6.0)
Requirement already satisfied: scipy>=0.18.1 in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(from gensim) (0.19.1)
Requirement already satisfied: numpy>=1.11.3 in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(from gensim) (1.12.1)
Requirement already satisfied: six>=1.5.0 in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(from gensim) (1.11.0)
Requirement already satisfied: bz2file in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from
smart-open>=1.2.1->gensim) (0.98)
Requirement already satisfied: boto3 in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from
smart-open>=1.2.1->gensim) (1.7.52)
Requirement already satisfied: boto>=2.32 in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(from smart-open>=1.2.1->gensim) (2.48.0)
Requirement already satisfied: requests in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from
smart-open>=1.2.1->gensim) (2.18.4)
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in /opt/conda/envs/py3.6/lib/python3.6/site-
packages (from boto3->smart-open>=1.2.1->gensim) (0.9.3)
Requirement already satisfied: botocore<1.11.0,>=1.10.52 in
```

```
/opt/conda/envs/py3.6/lib/python3.6/site-packages (from boto3->smart-open>=1.2.1->gensim)
(1.10.52)
Requirement already satisfied: s3transfer<0.2.0,>=0.1.10 in
/opt/conda/envs/py3.6/lib/python3.6/site-packages (from boto3->smart-open>=1.2.1->gensim) (0.1.13)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /opt/conda/envs/py3.6/lib/python3.6/site-p
ackages (from requests->smart-open>=1.2.1->gensim) (3.0.4)
Requirement already satisfied: idna<2.7,>=2.5 in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(from requests->smart-open>=1.2.1->gensim) (2.6)
Requirement already satisfied: urllib3<1.23,>=1.21.1 in /opt/conda/envs/py3.6/lib/python3.6/site-p
ackages (from requests->smart-open>=1.2.1->gensim) (1.22)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/envs/py3.6/lib/python3.6/site-
packages (from requests->smart-open>=1.2.1->gensim) (2018.4.16)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1; python_version >= "2.7" in
/opt/conda/envs/py3.6/lib/python3.6/site-packages (from botocore<1.11.0,>=1.10.52->boto3->smart-op
en>=1.2.1->gensim) (2.7.3)
Requirement already satisfied: docutils>=0.10 in /opt/conda/envs/py3.6/lib/python3.6/site-packages
(from botocore<1.11.0,>=1.10.52->boto3->smart-open>=1.2.1->gensim) (0.14)
ipywidgets 7.0.3 has requirement widgetsnbextension~=3.0.0, but you'll have widgetsnbextension 3.2
.1 which is incompatible.
```

In [3]:

```
#Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sqlite3 as sql
import seaborn as sns
from time import time
import random
import gensim
import warnings

warnings.filterwarnings("ignore")

%matplotlib inline
# sets the backend of matplotlib to the 'inline' backend:
#With this backend, the output of plotting commands is displayed inline within frontends like the
Jupyter notebook,
#directly below the code cell that produced it. The resulting plots will then also be stored in th
e notebook document.

#Functions to save objects for later use and retireve it
import pickle
def savetofile(obj,filename):
    pickle.dump(obj,open(filename+".p","wb"))
def openfromfile(filename):
    temp = pickle.load(open(filename+".p","rb"))
    return temp
```

In [4]:

```
# !wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (Windows NT 10.0
; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.139 Safari/537.36" --header=
"Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8" --
header="Accept-Language: en-US,en;q=0.9" "https://storage.googleapis.com/kaggle-
datasets/18/2157/database.sqlite.zip?GoogleAccessId=web-data@kaggle-
161607.iam.gserviceaccount.com&Expires=1526375292&Signature=G95OD7LgGsnAoencBUsNHa3R2iIGiXOhdITLbh
9IGS3JA9ETgbJRa3tHTguzL0ignoIz2sjQUxyY2YbcD98XR8immdcAmrFlQVA6Jm%2BBju%2BpDGjF05FpW0wGeMq6utKq2Qy8
NW%2FA%2F7m557B%2Bi3kGcBP4uaEzMk6F%2BpGaZnxcroDAcjpSj9VzU03INKPwpkbxtM%2FrWCaX748Bpgx9uKqwfrRakGR%
nMHcUukj%2FhaKKRi9QoQaTNpdRjmVB%2FqewKwDXTN8sr701yMkmqItQXBJI9Y312GqSP3Vd%2B3oleta5HZ2L9xlBFyUcLoy
xI4pTjukwu1A%3D%3D" -O "database.sqlite.zip" -c
```

Loading the data

In [40]:

```
#Using sqlite3 to retrieve data from sqlite file

con = sql.connect("final.sqlite")#Loading Cleaned/ Preprocesed text that we did in Text
Preprocessing

#Using pandas functions to query from sql table
```

```
df = pd.read_sql_query("""
SELECT * FROM Reviews
""",con)

#Reviews is the name of the table given
#Taking only the data where score != 3 as score 3 will be neutral and it won't help us much
df.head()
```

Out[40]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
0	0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	Positive	13
1	1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	Negative	13
2	2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	Positive	12
3	3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	Negative	13
4	4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	Positive	13

In [41]:

```
df.describe()
```

Out[41]:

	index	Id	HelpfulnessNumerator	HelpfulnessDenominator	Time
count	364171.000000	364171.000000	364171.000000	364171.000000	3.641710e+05
mean	241825.377603	261814.561014	1.739021	2.186841	1.296135e+09
std	154519.869452	166958.768333	6.723921	7.348482	4.864772e+07
min	0.000000	1.000000	0.000000	0.000000	9.393408e+08
25%	104427.500000	113379.500000	0.000000	0.000000	1.270858e+09
50%	230033.000000	249445.000000	0.000000	1.000000	1.311379e+09
75%	376763.500000	407408.500000	2.000000	2.000000	1.332893e+09
max	525813.000000	568454.000000	866.000000	878.000000	1.351210e+09

In [42]:

```
df.shape
df['Score'].size
```

Out[42]:

```
364171
```

-> For EDA and Text Preprecessing Refer other ipynb notebook

Score as positive or negative

In [43]:

```
def polarity(x):
    if x == "Positive":
        return 0
    else:
        return 1
df["Score"] = df["Score"].map(polarity) #Map all the scores as the function polarity i.e. positive or negative
df.head()
```

Out[43]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
0	0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	0	130
1	1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	134
2	2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	0	121
3	3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	1	130
4	4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	0	135

In [44]:

```
#Taking Sample Data
n_samples = 25000
df_sample = df.sample(n_samples)

###Sorting as we want according to time series
```

```
df_sample.sort_values('Time',inplace=True)
df_sample.head(10)
```

Out[44]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulne
117924	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0
1144	1146	1245	B00002Z754	A29Z5PI9BW2PU3	Robbie	7	7
117265	137932	149700	B00006L2ZT	A19JWUIRF6DXLV	Andrew J Monzon	2	4
316048	443664	479725	B00005U2FA	A270SG4UVKEO3X	Susanna "suzattorney"	23	23
77727	87386	95119	B0000DIYIJ	A3S4XR84R8S0TV	Brook Lindquist	0	1
218787	284749	308481	B0000DIVUR	AAFD4W6P5XWNT	Nick Watson	7	8
334889	477821	516699	B0000DG87B	AF5EKQ4I9NHJ4	Smitty Peete	1	21
262407	359912	389289	B0000DYZCG	A1U4PHVIQPBCD2	Dan Murphy	2	4
77127	86598	94281	B0000CNU2Q	A1NOWEOLKMRRXM	T. Reinhardt "olivia lee"	27	27
178039	224637	243579	B0000DIYKD	AYHW6HJSUCSAE	"insolent_shoeshine_grrl"	11	13

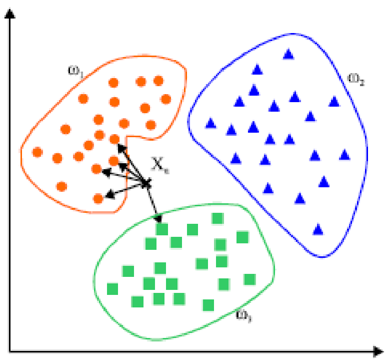
In [45]:

```
#Saving 25000 samples in disk to as to test to test on the same sample for each of all Algo
savetofile(df_sample,"sample_25000_knn")
```

In [4]:

```
#Opening from samples from file
df_sample = openfromfile("sample_25000_knn")
```

KNN Models using Different Vectorizing Techniques in NLP



Bag of Words (BoW)

A commonly used model in methods of Text Classification. As part of the BOW model, a piece of text (sentence or a document) is represented as a bag or multiset of words, disregarding grammar and even word order and the frequency or occurrence of each word is used as a feature for training a classifier.
OR
Simply,Converting a collection of text documents to a matrix of token counts

In [10]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

#Text -> Uni gram Vectors
uni_gram = CountVectorizer() #in scikit-learn
uni_gram_vectors = uni_gram.fit_transform(df_sample['CleanedText'].values)
uni_gram_vectors.shape
```

Out[10]:

```
(25000, 34014)
```

In [11]:

```
from sklearn import preprocessing
#Normalizing the data
uni_gram_vectors_norm = preprocessing.normalize(uni_gram_vectors)
print(uni_gram_vectors_norm.min())
print(uni_gram_vectors_norm.max())
#Not shuffling the data as we want it on time basis
X_train, X_test, y_train, y_test = train_test_split(uni_gram_vectors_norm,df_sample['Score'].values
,test_size=0.3,shuffle=False)
```

```
0.0
0.937042571332
```

In [12]:

```
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=10)
for train, cv in tscv.split(X_train):
```

```
#     print("%s %s" % (train, cv))
    print(X_train[train].shape, X_train[cv].shape)
```

```
(1600, 34014) (1590, 34014)
(3190, 34014) (1590, 34014)
(4780, 34014) (1590, 34014)
(6370, 34014) (1590, 34014)
(7960, 34014) (1590, 34014)
(9550, 34014) (1590, 34014)
(11140, 34014) (1590, 34014)
(12730, 34014) (1590, 34014)
(14320, 34014) (1590, 34014)
(15910, 34014) (1590, 34014)
```

Finding the best 'k' value using Forward Chaining Cross Validation or Time Series CV

1. Without Grid Search CV

In [8]:

```
%%time
from sklearn.model_selection import TimeSeriesSplit
from sklearn.neighbors import KNeighborsClassifier

#No of splits for Forward Chaining Cross Validation
n_splits = 10
#Max no. of neighbours for KNN
neigh_max = 100

tscv = TimeSeriesSplit(n_splits=n_splits)
#To store accuracy of different k values
k_acc = []

for k in range(1,neigh_max,2):
    #To store accuracy of different fold
    acc_list = []
    for train, cv in tscv.split(X_train):
          if(train.size > k):
            knn = KNeighborsClassifier(n_neighbors=k,algorithm='brute',n_jobs=-1)
            knn.fit(X_train[train],y_train[train])
            acc_list.append(knn.score(X_train[cv],y_train[cv])*100)
    if(acc_list):
        acc_nparr = np.array(acc_list)
    k_acc.append(acc_nparr.mean())
k_acc = np.array(k_acc)
```

```
CPU times: user 2min 50s, sys: 1min 23s, total: 4min 14s
Wall time: 6min 3s
```

In [9]:

```
savetofile(k_acc,"k_acc_uni_gram")
```

In [9]:

```
k_acc_uni_gram = openfromfile("k_acc_uni_gram")
k_acc_uni_gram
```

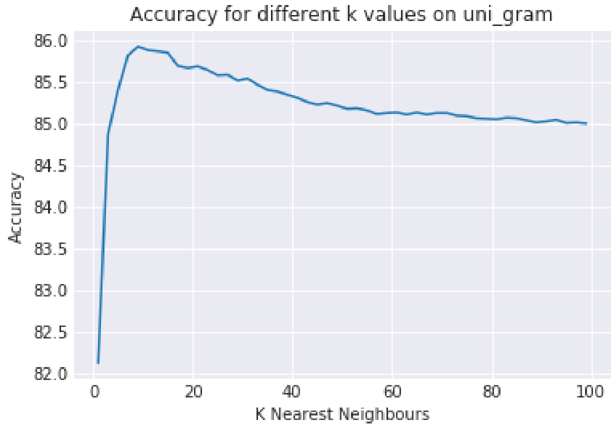
Out[9]:

```
array([ 82.12578616,  84.87421384,  85.40251572,  85.81761006,
        85.9245283 ,  85.88050314,  85.86792453,  85.8490566 ,
        85.6918239 ,  85.66666667,  85.68553459,  85.64150943,
        85.57861635,  85.58490566,  85.51572327,  85.5408805 ,
        85.46540881,  85.40251572,  85.3836478 ,  85.34591195,
        85.3081761 ,  85.25786164,  85.22641509,  85.24528302,
        85.21383648,  85.17610063,  85.18238994,  85.1572327 ,
        85.11320755,  85.12578616,  85.13207547,  85.10691824,
```

```
       85.11320755,  85.12578616,  85.13207547,  85.10691824,
       85.13207547,  85.10691824,  85.12578616,  85.12578616,
       85.09433962,  85.08805031,  85.06289308,  85.05660377,
       85.05031447,  85.06918239,  85.06289308,  85.03773585,
       85.01257862,  85.02515723,  85.04402516,  85.00628931,
       85.01257862,  85.        ])
```

In [11]:

```
sns.set_style("darkgrid")
plt.plot(np.arange(1,100,2),k_acc_uni_gram)
plt.xlabel("K Nearest Neighbours")
plt.ylabel("Accuracy")
plt.title("Accuracy for different k values on uni_gram")
plt.show()
```



With k=11-13 uni_gram has the highest accuracy of 86%
As we can see after a no. of neighbours the accuracy dips hence the no. of neighbours is restricted to 100 neighbours

2. With Grid Search CV

The above code for finding best value of 'k' can be condensed using Grid Search CV it tries all the possible params which tell it to try on and returns the best params and best accuracy

A.Brute Algorithm

In [29]:

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(algorithm='brute',n_jobs=-1)
# neigh = np.arange(1,100,2)
param_grid = {'n_neighbors':np.arange(1,100,2)} #params we need to try on classifier
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(knn,param_grid,cv=tscv,verbose=1)
gsv.fit(X_train,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 12.2 µs
Fitting 10 folds for each of 50 candidates, totalling 500 fits
Best HyperParameter:  {'n_neighbors': 9}
Best Accuracy: 85.92%
```

```
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed: 32.3min finished
```

In [30]:

```
#Testing Accuracy on Test data
```

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=9)
knn.fit(X_train,y_train)
score = knn.score(X_test,y_test)
print("Accuracy on test set: %0.3f%%"%(score*100))
```

```
Accuracy on test set: 83.707%
```

B. Kd tree Algorithm

In [31]:

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(algorithm='kd_tree',n_jobs=-1)
# neigh = np.arange(1,100,2)
param_grid = {'n_neighbors':np.arange(1,100,2)} #params we need to try on classifier
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(knn,param_grid,cv=tscv,verbose=1)
gsv.fit(X_train,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 7.87 µs
Fitting 10 folds for each of 50 candidates, totalling 500 fits
Best HyperParameter:  {'n_neighbors': 9}
Best Accuracy: 85.92%
```

```
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed: 32.8min finished
```

In [13]:

```
#Testing Accuracy on Test data
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=9,algorithm='kd_tree')
knn.fit(X_train,y_train)
score = knn.score(X_test,y_test)
print("Accuracy on test set: %0.3f%%"%(score*100))
```

```
Accuracy on test set: 84.080%
```

bi-gram

In [14]:

```
from sklearn.feature_extraction.text import CountVectorizer
#taking one words and two consecutive words together
bi_gram = CountVectorizer(ngram_range=(1,2))
bi_gram_vectors = bi_gram.fit_transform(df_sample['CleanedText'].values)
bi_gram_vectors.shape
```

Out[14]:

```
(25000, 501388)
```

In [15]:

```
from sklearn import preprocessing
bi_gram_vectors_norm = preprocessing.normalize(bi_gram_vectors)
```

In [16]:

```
from sklearn.model_selection import train_test_split
#Not shuffling the data as we want it on time basis
X_train, X_test, y_train, y_test = train_test_split(bi_gram_vectors_norm,df_sample['Score'].values,
test_size=0.3,shuffle=False)
```

In [17]:

```
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=10)
for train, cv in tscv.split(X_train):
#     print("%s %s" % (train, cv))
    print(X_train[train].shape, X_train[cv].shape)
```

```
(1600, 501388) (1590, 501388)
(3190, 501388) (1590, 501388)
(4780, 501388) (1590, 501388)
(6370, 501388) (1590, 501388)
(7960, 501388) (1590, 501388)
(9550, 501388) (1590, 501388)
(11140, 501388) (1590, 501388)
(12730, 501388) (1590, 501388)
(14320, 501388) (1590, 501388)
(15910, 501388) (1590, 501388)
```

A.Brute Algorithm

In [39]:

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(algorithm='brute',n_jobs=-1)
# neigh = np.arange(1,100,2)
param_grid = {'n_neighbors':np.arange(1,100,2)} #params we need to try on classifier
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(knn,param_grid,cv=tscv,verbose=1)
gsv.fit(X_train,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 7.63 µs
Fitting 10 folds for each of 50 candidates, totalling 500 fits
Best HyperParameter:  {'n_neighbors': 13}
Best Accuracy: 85.99%
```

```
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed: 33.9min finished
```

With k=13 bi_gram has the highest accuracy of 85.99% in Cross Validation

In [42]:

```
#Testing Accuracy on Test data
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=13)
knn.fit(X_train,y_train)
score = knn.score(X_test,y_test)
print("Accuracy on test set: %0.3f%%"%(score*100))
```

```
Accuracy on test set: 84.120%
```

B. Kd tree Algorithm

In [43]:

```
%time
```

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(algorithm='kd_tree',n_jobs=-1)
# neigh = np.arange(1,100,2)
param_grid = {'n_neighbors':np.arange(1,100,2)} #params we need to try on classifier
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(knn,param_grid,cv=tscv,verbose=1)
gsv.fit(X_train,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 10.7 µs
Fitting 10 folds for each of 50 candidates, totalling 500 fits
Best HyperParameter:  {'n_neighbors': 13}
Best Accuracy: 85.99%
```

```
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed: 33.8min finished
```

With k=13 bi_gram has the highest accuracy of 85.99% in Cross Validation

In [19]:

```
#Testing Accuracy on Test data
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=13,algorithm='kd_tree')
knn.fit(X_train,y_train)
score = knn.score(X_test,y_test)
print("Accuracy on training set: %0.3f%%"%(score*100))
```

```
Accuracy on training set: 84.747%
```

tf-idf

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

In [20]:

```
%%time
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(ngram_range=(1,2)) #Using bi-grams
tfidf_vec = tfidf.fit_transform(df_sample['CleanedText'].values)
tfidf_vec.shape
```

```
CPU times: user 4.34 s, sys: 20 ms, total: 4.36 s
Wall time: 4.36 s
```

In [21]:

```
tfidf_vec.shape
```

Out[21]:

```
(25000, 501388)
```

In [22]:

```python
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

tfidf_vec_norm = preprocessing.normalize(tfidf_vec)

#Not shuffling the data as we want it on time basis
X_train, X_test, y_train, y_test = train_test_split(tfidf_vec_norm,df_sample['Score'].values,test_s
ize=0.3,shuffle=False)
```

A.Brute Algorithm

In [15]:

```python
%time
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(algorithm='brute')
# neigh = np.arange(1,100,2)
param_grid = {'n_neighbors':np.arange(1,100,2)} #params we need to try on classifier
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(knn,param_grid,cv=tscv,verbose=1)
gsv.fit(X_train,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 13.1 µs
Fitting 10 folds for each of 50 candidates, totalling 500 fits
Best HyperParameter:  {'n_neighbors': 11}
Best Accuracy: 85.68%
```

```
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed: 47.4min finished
```

With k=13 bi_gram has the highest accuracy of 85.99% in Cross Validation

In [17]:

```python
#Testing Accuracy on Test data
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=11)
knn.fit(X_train,y_train)
score = knn.score(X_test,y_test)
print("Accuracy on test set: %0.3f%%"%(score*100))
```

```
Accuracy on test set: 84.187%
```

B. Kd tree Algorithm

In [18]:

```python
%time
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(algorithm='kd_tree')
# neigh = np.arange(1,100,2)
param_grid = {'n_neighbors':np.arange(1,100,2)} #params we need to try on classifier
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(knn,param_grid,cv=tscv,verbose=1)
gsv.fit(X_train,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 319 µs
Fitting 10 folds for each of 50 candidates, totalling 500 fits
```

```
Fitting 10 folds for each of 50 candidates, totalling 500 fits
Best HyperParameter:  {'n_neighbors': 11}
Best Accuracy: 85.68%
```

```
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed: 47.1min finished
```

With k=13 bi_gram has the highest accuracy of 85.99% in Cross Validation

In [23]:

```python
#Testing Accuracy on Test data
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=11,algorithm='kd_tree')
knn.fit(X_train,y_train)
score = knn.score(X_test,y_test)
print("Accuracy on test set: %0.3f%%"%(score*100))
```

```
Accuracy on test set: 84.373%
```

Gensim

Gensim is a robust open-source vector space modeling and topic modeling toolkit implemented in Python. It uses NumPy, SciPy and optionally Cython for performance. Gensim is specifically designed to handle large text collections, using data streaming and efficient incremental algorithms, which differentiates it from most other scientific software packages that only target batch and in-memory processing.

Word2Vec

[Refer Docs] :https://radimrehurek.com/gensim/models/word2vec.html

In [27]:

```python
from gensim.models import KeyedVectors

#Loading the model from file in the disk
w2vec_model = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
```

In [28]:

```python
w2v_vocub = w2vec_model.wv.vocab
len(w2v_vocub)
```

Out[28]:

```
3000000
```

Avg Word2Vec

- **One of the most naive but good ways to convert a sentence into a vector**
- **Convert all the words to vectors and then just take the avg of the vectors the resulting vector represent the sentence**

In [29]:

```python
%%time
avg_vec_google = [] #List to store all the avg w2vec's
# no_datapoints = 364170
# sample_cols = random.sample(range(1, no_datapoints), 20001)
for sent in df_sample['CleanedText_NoStem']:
    cnt = 0 #to count no of words in each reviews
    sent_vec = np.zeros(300) #Initializing with zeroes
#     print("sent:",sent)
    sent = sent.decode("utf-8")
    for word in sent.split():
```

```
        try:
#             print(word)
            wvec = w2vec_model.wv[word] #Vector of each using w2v model
#             print("wvec:",wvec)
            sent_vec += wvec #Adding the vectors
#             print("sent_vec:",sent_vec)
            cnt += 1
        except:
            pass #When the word is not in the dictionary then do nothing
#     print(sent_vec)
    sent_vec /= cnt #Taking average of vectors sum of the particular review
#     print("avg_vec:",sent_vec)
    avg_vec_google.append(sent_vec) #Storing the avg w2vec's for each review
#     print("*********************************************************************")
# print(avg_vec_google)
avg_vec_google = np.array(avg_vec_google)
```

```
CPU times: user 12 s, sys: 32 ms, total: 12 s
Wall time: 12 s
```

In [30]:

```
np.isnan(avg_vec_google).any()
```

Out[30]:

```
False
```

In [31]:

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

avg_vec_norm = preprocessing.normalize(avg_vec_google)

#Not shuffling the data as we want it on time basis
X_train, X_test, y_train, y_test = train_test_split(avg_vec_norm,df_sample['Score'].values,test_siz
e=0.3,shuffle=False)
```

In [32]:

```
avg_vec_norm.shape
```

Out[32]:

```
(25000, 300)
```

In [33]:

```
avg_vec_norm.max()
```

Out[33]:

```
0.26854231895936098
```

A.Brute Algorithm

In [58]:

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(algorithm='brute')
# neigh = np.arange(1,100,2)
param_grid = {'n_neighbors':np.arange(1,40,2)} #params we need to try on classifier
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(knn,param_grid,cv=tscv,verbose=1)
gsv.fit(X_train,y_train)
print("Best HyperParameter: ",gsv.best_params_)
```

```
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 10 µs
Fitting 10 folds for each of 20 candidates, totalling 200 fits
Best HyperParameter:  {'n_neighbors': 11}
Best Accuracy: 85.61%
```

```
[Parallel(n_jobs=1)]: Done 200 out of 200 | elapsed: 14.0min finished
```

In [60]:

```
#Testing Accuracy on Test data
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=11)
knn.fit(X_train,y_train)
score = knn.score(X_test,y_test)
print("Accuracy on test set: %0.3f%%"%(score*100))
```

```
Accuracy on test set: 85.107%
```

B. Kd tree Algorithm

In [59]:

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(algorithm='kd_tree')
# neigh = np.arange(1,100,2)
param_grid = {'n_neighbors':np.arange(1,40,2)} #params we need to try on classifier
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(knn,param_grid,cv=tscv,verbose=1,n_jobs=-1)
gsv.fit(X_train,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 8.82 µs
Fitting 10 folds for each of 20 candidates, totalling 200 fits
```

```
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:   19.4s
[Parallel(n_jobs=-1)]: Done 160 tasks      | elapsed: 15.4min
[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed: 23.3min finished
```

```
Best HyperParameter:  {'n_neighbors': 11}
Best Accuracy: 85.61%
```

In [34]:

```
#Testing Accuracy on Test data
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=11,algorithm='kd_tree')
knn.fit(X_train,y_train)
score = knn.score(X_test,y_test)
print("Accuracy on test set: %0.3f%%"%(score*100))
```

```
Accuracy on test set: 85.107%
```

Tf-idf W2Vec

- **Another way to covert sentence into vectors**
- **Take weighted sum of the vectors divided by the sum of all the tfidf's**

i.e. (tfidf(word) x w2v(word))/sum(tfidf's)

In [62]:

```
%%time
###Sorting as we want according to time series
df_sample.sort_values('Time',inplace=True)

###tf-idf with No Stemming
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(ngram_range=(1,2)) #Using bi-grams

tfidf_vec_new = tfidf.fit_transform(df_sample['CleanedText_NoStem'].values)

print(tfidf_vec_new.shape)

features = tfidf.get_feature_names()
```

```
(25000, 586319)
CPU times: user 6.37 s, sys: 92 ms, total: 6.46 s
Wall time: 6.93 s
```

In [67]:

```
%%time
tfidf_w2v_vec_google = []
review = 0

for sent in df_sample['CleanedText_NoStem'].values:
    cnt = 0
    weighted_sum  = 0
    sent_vec = np.zeros(300)
    sent = sent.decode("utf-8")
    for word in sent.split():
        try:
#              print(word)
            wvec = w2vec_model.wv[word] #Vector of each using w2v model
#              print("w2vec:",wvec)
#              print("tfidf:",tfidf_vec_ns[review,features.index(word)])
            tfidf_vec = tfidf_vec_new[review,features.index(word)]
            sent_vec += (wvec * tfidf_vec)
            weighted_sum += tfidf_vec
        except:
#              print(review)
            pass
    sent_vec /= weighted_sum
#     print(sent_vec)
    tfidf_w2v_vec_google.append(sent_vec)
    review += 1
tfidf_w2v_vec_google = np.array(tfidf_w2v_vec_google)
```

```
CPU times: user 5h 58min 35s, sys: 2.69 s, total: 5h 58min 38s
Wall time: 5h 58min 39s
```

In [73]:

```
savetofile(tfidf_w2v_vec_google,"tfidf_w2v_vec_google")
```

In [35]:

```
tfidf_w2v_vec_google = openfromfile("tfidf_w2v_vec_google")
```

In [36]:

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

tfidfw2v_vecs_norm = preprocessing.normalize(tfidf_w2v_vec_google)

#Not shuffling the data as we want it on time basis
X_train, X_test, y_train, y_test = train_test_split(tfidfw2v_vecs_norm,df_sample['Score'].values,te
st_size=0.3,shuffle=False)
```

```
st_size=0.3,shuffle=False)
```

A.Brute Algorithm

In [7]:

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import TimeSeriesSplit

knn = KNeighborsClassifier(algorithm='brute')
# neigh = np.arange(1,100,2)
param_grid = {'n_neighbors':np.arange(1,40,2)} #params we need to try on classifier
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(knn,param_grid,cv=tscv,verbose=1)
gsv.fit(X_train,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 12.2 µs
Fitting 10 folds for each of 20 candidates, totalling 200 fits
Best HyperParameter:  {'n_neighbors': 21}
Best Accuracy: 84.48%
```

```
[Parallel(n_jobs=1)]: Done 200 out of 200 | elapsed: 13.8min finished
```

In [6]:

```
#Testing Accuracy on Test data
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=21)
knn.fit(X_train,y_train)
score = knn.score(X_test,y_test)
print("Accuracy on test set: %0.3f%%"%(score*100))
```

```
Accuracy on test set: 82.667%
```

B. Kd tree Algorithm

In [7]:

```
%time
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import TimeSeriesSplit

knn = KNeighborsClassifier(algorithm='kd_tree')
# neigh = np.arange(1,100,2)
param_grid = {'n_neighbors':np.arange(1,40,2)} #params we need to try on classifier
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
gsv = GridSearchCV(knn,param_grid,cv=tscv,verbose=1)
gsv.fit(X_train,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 7.63 µs
Fitting 10 folds for each of 20 candidates, totalling 200 fits
```

```
[Parallel(n_jobs=1)]: Done 200 out of 200 | elapsed: 336.8min finished
```

```
Best HyperParameter:  {'n_neighbors': 35}
Best Accuracy: 84.48%
```

In [37]:

```
#Testing Accuracy on Test data
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=35,algorithm='kd_tree')
knn.fit(X_train,y_train)
score = knn.score(X_test,y_test)
print("Accuracy on test set: %0.3f%%"%(score*100))
```

```
Accuracy on test set: 82.667%
```

KNN (with 25k points)

Featurization	Algo	Accuracy	
Uni - gram	brute	83.707	
	kd-tree	84.08	
Bi -gram	brute	84.12	
	kd-tree	84.747	
tfidf	brute	84.187	
	kd-tree	84.374	
Avg Word2Vec	brute	85.107	
	kd-tree	85.107	
tfidf - Word2vec	brute	82.667	
	kd-tree	82.667	

Conclusions

1. **Best Accuracy is achieved by Avg Word2Vec Featurization and both algorithm got the same accuracy**
2. **As I have taken only 25k points(due to huge training time) the accuracy will not be the representive of the real accuracy**