Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

Objective:

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

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

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

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [9]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


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

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

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

from sklearn.tree import DecisionTreeClassifier

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

from tqdm import tqdm
import os
```

In [10]:

```
# using the SQLite Table to read data.
con = sqlite3.connect(r'C:\Sandy\privy\AI\Data Sets\Amazon Food rev dataset\datal

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

#Took 3000 points from each Category i.e from Positive reviews and Negative Revie
#Negative Data
Neg_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score < 3 LIMIT 100(
#Positive Data
Pos_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score > 3 LIMIT 100(

Neg_data.head()
preprocessed_data =pd.concat([Neg_data,Pos_data])
print("Total Sample Points : ",preprocessed_data.shape)
#filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 ""
print("\n Sample Points : ")
preprocessed_data.head()
```

```
Total Sample Points :  (20000, 10)

 Sample Points :
```

Out[10]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomi
0	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
1	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	
2	13	B0009XLVG0	A327PCT23YH90	LT	1	
3	17	B001GVISJM	A3KLWF6WQ5BNYO	Erica Neathery	0	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomi
4	27	B001GVISJM	A3RXAU2N8KV45G	lady21	0	

In [11]:
```
preprocessed_data=preprocessed_data[preprocessed_data['HelpfulnessNumerator']<=pr
```

In [12]:
```
preprocessed_data.shape
```

Out[12]:
```
(20000, 10)
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

In [13]:
```
#Sorting data according to ProductId in ascending order
sorted_data=preprocessed_data.sort_values('ProductId', axis=0, ascending=True, in
```

In [14]:
```
# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negat
def partition(x):
    if x < 3:
        return 0
    return 1

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

```
Number of data points in our dataset (20000, 10)
```

Out[14]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominat
0	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
1	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	
2	13	B0009XLVG0	A327PCT23YH90	LT	1	

In [15]:
```
#Deduplication of entries
final_data=preprocessed_data.drop_duplicates(subset={"UserId","ProfileName","Tim
final_data.shape
```

Out[15]:
```
(18678, 10)
```

In [16]:
```
#Checking to see how much % of data still remains

(final_data['Id'].size*1.0)/(preprocessed_data['Id'].size*1.0)*100
```

Out[16]:
```
93.39
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [17]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[17]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomir
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	

In [18]:
```
final=final_data[final_data.HelpfulnessNumerator<=final_data.HelpfulnessDenomina
```

In [19]:
```
final=final.sort_values('Time',axis=0, ascending=True , inplace=False, kind='qui
```

In [20]:
```
#Before starting the next phase of preprocessing lets see the number of entries
print(final.shape)

#How many positive and negative reviews are present in our dataset?
print(final['Score'].value_counts())
```

```
(18678, 10)
1    9600
0    9078
Name: Score, dtype: int64
```

In [21]:
```
#Before starting the next phase of preprocessing lets see the number of entries l
print(final.shape)

y=final[['Score']]
print(len(y))
```

```
(18678, 10)
18678
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [22]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [23]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st st

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'our
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itse
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'tha
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'ha
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'tl
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than'
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should'v
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "d
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma'
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn
            'won', "won't", 'wouldn', "wouldn't"])
```

In [24]:
```
# Combining all the above stundents
from tqdm import tqdm
from bs4 import BeautifulSoup
preprocessed_reviews_text = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'html.parser').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in s
    preprocessed_reviews_text.append(sentance.strip())
```

```
100%|
| 18678/18678 [00:14<00:00, 1261.08it/s]
```

In [25]:
```
preprocessed_reviews_text[1]
```

Out[25]:
```
'received shipment could hardly wait try product love slickers call instead sti
ckers removed easily daughter designed signs printed reverse use car windows pr
inted beautifully print shop program going lot fun product windows everywhere s
urfaces like tv screens computer monitors'
```

[3.2] Preprocessing Review Summary

In [26]:
```
from tqdm import tqdm
from bs4 import BeautifulSoup
preprocessed_reviews_summary = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Summary'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'html.parser').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in s
    preprocessed_reviews_summary.append(sentance.strip())
```

```
100%|
| 18678/18678 [00:18<00:00, 1001.96it/s]
```

Concatenating Summary and text reviews

In [27]:
```
preprocessed_reviews=[]
for i in tqdm(range(len(preprocessed_reviews_text))):
    preprocessed_reviews.append(preprocessed_reviews_text[i]+' '+preprocessed_rev
```

```
100%|                                                                        |
18678/18678 [00:00<00:00, 359417.76it/s]
```

In [28]:
```
preprocessed_reviews[0]
```

Out[28]:
```
'really good idea final product outstanding use decals car window everybody ask
s bought decals made two thumbs great product'
```

[4] Featurization

In [29]:
```
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold",
    return t

def predict_with_best_t(proba, threshold):
    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

[4.2] TF-IDF

In [30]:
```
from sklearn.model_selection import train_test_split

x_train_tf_idf_sent,x_test_tf_idf_sent,y_train_tf_idf,y_test_tf_idf=train_test_s|
```

Top features from Tf-idf Vectorizer

In [61]:
```

tf_idf_vect = TfidfVectorizer(ngram_range=(1,1),min_df=10,max_features=5000,anal
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_
print('='*50)

x_train_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(x_train_tf_idf))
print("the shape of out text TFIDF vectorizer ",x_train_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", x_train_
print('='*50)


#x_test_tf_idf = tf_idf_vect.transform(x_test_tf_idf_sent)
#print("the type of count vectorizer ",type(x_test_tf_idf))
#print("the shape of out text TFIDF vectorizer ",x_test_tf_idf.get_shape())
#print("the number of unique words including both unigrams and bigrams ", x_test_
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'absolut
e', 'absolutely', 'absorb', 'absorbed', 'acai', 'accept', 'acceptable', 'accept
ed']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (18678, 5000)
the number of unique words including both unigrams and bigrams  5000
==================================================
```

In [62]:
```
temp_features=np.argsort(tf_idf_vect.idf_)
print(len(temp_features))
```

```
5000
```

In [63]:
```
top_features=np.take(tf_idf_vect.get_feature_names(),temp_features[len(tf_idf_ve
```

In [64]:
```
df_co_occ=pd.DataFrame(0,index=top_features,columns=top_features)
```

Co-Occurance Matrix

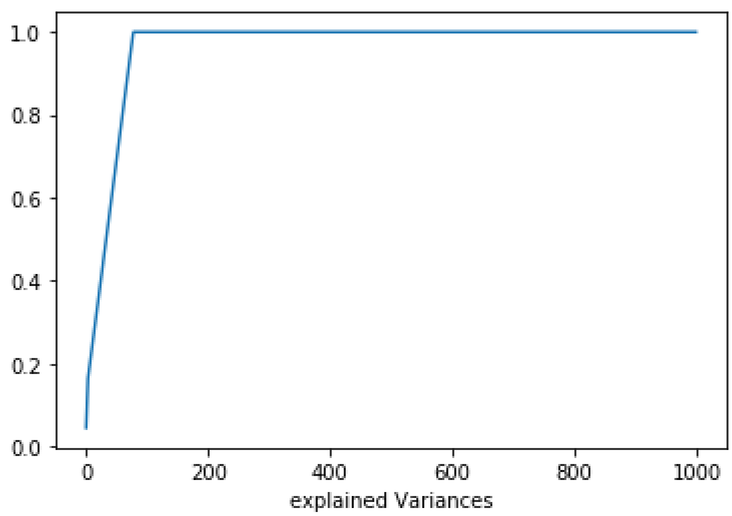
In [37]:
```
#updating co-occurance matrix
cnt=0
for i in preprocessed_reviews:
    x=i.split()
    for j in range(len(x)):
        if j < len(x)-1:
            if x[j] in list(df_co_occ.index.values) and x[j+1] in list(df_co_occ
                df_co_occ.loc[x[j],x[j+1]]+=1

co_occ_mtrx=df_co_occ.as_matrix()
```

In [38]:
```
from sklearn.decomposition import TruncatedSVD
svd=TruncatedSVD(n_components=1000)
svd_vec=svd.fit_transform(co_occ_mtrx)
```

Selcting optimal Components

In [41]:
```
explained_variances=[i/np.sum(svd.explained_variance_ratio_) for i in svd.explai
variances=[]
temp=0
for i in explained_variances:
    temp=temp+i
    variances.append(temp)
plt.plot(variances,label='Explained Variances')
plt.xlabel("explained Variances")
plt.show()
```



In [42]:
```
svd_opt=TruncatedSVD(n_components=130)
x_data=svd_opt.fit_transform(co_occ_mtrx)
```

K-Means

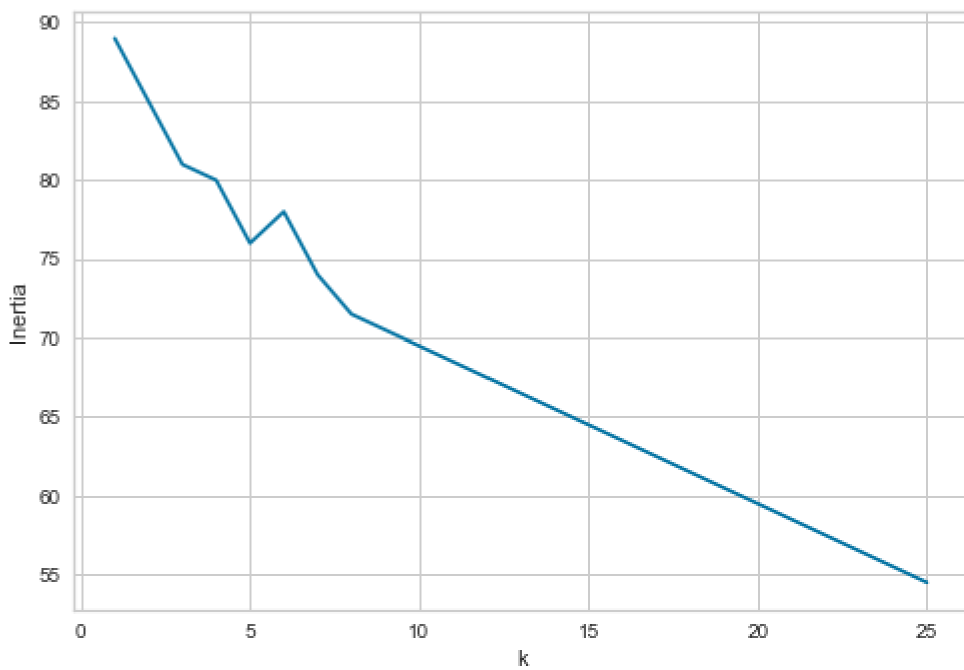
[5.2] Applying K-means on TFIDF, SET 2

In [43]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve, auc,roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer


n_clusters=[i for i in range(1,26)]
#n_clusters=[1,2]
inertia=[]
#K-Means Model
for k in n_clusters:
    k_means_clstr=KMeans(n_clusters=k,init='k-means++')
    k_means_clstr.fit(x_data)
    inertia.append(k_means_clstr.inertia_)



plt.plot(n_clusters,inertia,label='K vs Inertia')
plt.xlabel('k')
plt.ylabel('Inertia')
plt.show()
```



In [44]:
```
opt_k_tf_idf=10
k_means_tf_idf=KMeans(n_clusters=opt_k_tf_idf,init='k-means++')
k_means_tf_idf.fit(x_data)


#print(cluster_data)
```

Out[44]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=10, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [45]:
```
print(set(k_means_tf_idf.labels_))
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

WordCloud per cluster

In [46]:
```
cluster_data_tf_idf={}
for i in range(0,opt_k_tf_idf):
    cluster_data_tf_idf['clust{0}'.format(i)]=[]
#print(cluster_data)
#Words extraction per cluster
for i in range(0,len(k_means_tf_idf.labels_)):
    features=np.take(tf_idf_vect.get_feature_names(),x_train_tf_idf[i].indices).
    cluster_data_tf_idf['clust{0}'.format(k_means_tf_idf.labels_[i])] =  cluster_
#print(len(k_means_tf_idf.labels_))
```

In [47]:
```
from  wordcloud  import WordCloud
for i in range(0,len(cluster_data_tf_idf)):
    #print(cluster_data_tf_idf['clust{0}'.format(i)])
    if cluster_data_tf_idf['clust{0}'.format(i)] != []:
        print("WordCloud for Cluster : clust{0}  ".format(i))
        text=' '.join([str(elem) for elem in cluster_data_tf_idf['clust{0}'.form
        pos_rev=WordCloud().generate(text)
        plt.imshow(pos_rev,interpolation='bilinear')
        plt.show()
```

```
WordCloud for Cluster : clust0
```


```
WordCloud for Cluster : clust1
```

In [50]:
```
from sklearn.metrics.pairwise import cosine_similarity
```

In [51]:
```
#to know what arguments does function or module take as input parameters
import inspect
inspect.getargspec(cosine_similarity)
inspect.getargspec(cosine_similarity.__init__)
```

Out[51]:
```
ArgSpec(args=['self'], varargs='args', keywords='kwargs', defaults=None)
```

In [76]:
```
#https://github.com/Manish-12/Truncated-SVD-on-Amazon-fine-food-reviews-/blob/mas
def cos_sim(word):
    cos_sim_vec=cosine_similarity(svd_vec)
    sim_words_vec=cos_sim_vec[top_features.tolist().index(word)]
    print("similar words of {0} are {1}".format(word,np.take(top_features,sim_wor
```

Top 10 similar words for a given word

In [77]:
```
cos_sim(top_features[0])
#top_features
```

```
similar words of owners are ['dash' 'wasteful' 'winner' 'plug' 'cuisinart' 'bat
ter' 'chopped' 'glycol'
 'marzano' 'owners']
```

Conclusion :

- taken top 3000 features based on idf values.
- Constructed a Co-occurance Matrix with these 3000 features
- Applied Truncated SVD on co-occurance matrix with optimal number of components.
- Applied Kmeans on truncated SVD to analyse the clusters.
- We got the optimal k value from knee method is 10
- for tf_idf vectorizer k-means developed wide range of clusters to group the data
- We can observe from wordclouds that words per cluster are much similar
- We can also observe that some features are included into different clusters this displays that these clusters are nearer to each other.

In [ ]: