

```
In [1]: %matplotlib inline
import warnings
warnings.filterwarnings("ignore")

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

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

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

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

from tqdm import tqdm
import os
```

```
In [229]: from google.colab import drive
drive.mount('/content/gdrive')
```

```
-----
ImportError                                Traceback (most recent call last)
<ipython-input-229-6db90e9297aa> in <module>()
----> 1 from google.colab import drive
      2 drive.mount('/content/gdrive')

ImportError: No module named 'google.colab'
```

```

In [230]: #connecting to sqlite db
# con = sqlite3.connect('/content/gdrive/My Drive/Colab Notebooks/Assignment 4/data/database.sqlite')
con = sqlite3.connect('database.sqlite')

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

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000 """, con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 """, con)

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rating(-1)
def partition(x):
    if x < 3:
        return 0
    else:
        return 1

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

```

Number of data points in our data (525814, 10)

Out[230]:

|   | Id | ProductId  | UserId         | ProfileName                              | HelpfulnessNumerator | HelpfulnessDenominator |
|---|----|------------|----------------|--|----------------------|------------------------|
| 0 | 1  | B001E4KFG0 | A3SGXH7AUHU8GW | delmartian                               | 1                    | 1                      |
| 1 | 2  | B00813GRG4 | A1D87F6ZCVE5NK | dll pa                                   | 0                    | 0                      |
| 2 | 3  | B000LQOCH0 | ABXLMWJIXXAIN  | Natalia<br>Corres<br>"Natalia<br>Corres" | 1                    | 1                      |

```
In [231]: display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

```
In [232]: print(display.shape)
display.head()
```

```
(80668, 7)
```

```
Out[232]:
```

|   | UserId             | ProductId  | ProfileName            | Time       | Score | Text  | COUNT |
|---|--------------------|------------|------------------------|------------|-------|---|-------|
| 0 | #oc-R115TNMSPFT9I7 | B007Y59HVM | Breyton                | 1331510400 | 2     | Overall its just OK when considering the price... | 2     |
| 1 | #oc-R11D9D7SHXIJB9 | B005HG9ET0 | Louis E. Emory "hoppy" | 1342396800 | 5     | My wife has recurring extreme muscle spasms, u... | 3     |
| 2 | #oc-R11DNU2NBKQ23Z | B007Y59HVM | Kim Cieszykowski       | 1348531200 | 1     | This coffee is horrible and unfortunately not ... | 2     |
| 3 | #oc-R11O5J5ZVQE25C | B005HG9ET0 | Penguin Chick          | 1346889600 | 5     | This will be the bottle that you grab from the... | 3     |
| 4 | #oc-R12KPBODL2B5ZD | B007OSBE1U | Christopher P. Presta  | 1348617600 | 1     | I didnt like this coffee. Instead of telling y... | 2     |

```
In [233]: # Removing duplicate reviews
final=filtered_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"})
print(final.shape)
```

```
(364173, 10)
```

```
In [234]: (final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
Out[234]: 69.25890143662969
```

```
In [235]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
In [236]: #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?
final['Score'].value_counts()
```

```
(364171, 10)
```

```
Out[236]: 1    307061
0     57110
Name: Score, dtype: int64
```

```
In [237]: final["cleanReview"] = final["Summary"].map(str) + ". " + final["Text"]
```

```
In [238]: final['cleanReview'].head()
```

```
Out[238]: 0    Good Quality Dog Food. I have bought several o...
1    Not as Advertised. Product arrived labeled as ...
2    "Delight" says it all. This is a confection th...
3    Cough Medicine. If you are looking for the sec...
4    Great taffy. Great taffy at a great price. Th...
Name: cleanReview, dtype: object
```

```
In [239]: final['lengthOfReview'] = final['cleanReview'].str.split().str.len()
final['lengthOfReview'].head()
```

```
Out[239]: 0    52
1    34
2    98
3    43
4    29
Name: lengthOfReview, dtype: int64
```

```
In [19]: #remove urls from text python
from tqdm import tqdm
lst = []
removed_urls_list = []
for text in tqdm(final['cleanReview']):
    removed_urls_text = re.sub(r"http\S+", "", text)
    lst.append(removed_urls_text)
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 364171/364171 [00:01<00:00
0, 348689.13it/s]
```



```
In [25]: strip_list = []
         for to_strip in tqdm(decat_lst):
             text = re.sub("\S*d\S*", "", to_strip).strip()
             strip_list.append(text)
```

```
In [29]: print(len(preprocessed_reviews))
preprocessed_reviews[-1]
```

364171

```
Out[29]: 'great honey satisfied product advertised use cereal raw vinegar general sweetn
er'
```

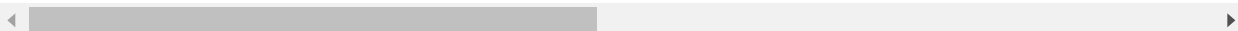
```
In [30]: final['cleanReview'] = preprocessed_reviews
```

```
In [117]: print(len(final))
final.tail(5)
```

364171

```
Out[117]:
```

|               | Id     | ProductId  | UserId         | ProfileName                | HelpfulnessNumerator | HelpfulnessDenominator |
|---------------|--------|------------|----------------|----------------------------|----------------------|------------------------|
| <b>525809</b> | 568450 | B001EO7N10 | A28KG5XORO54AY | Lettie D. Carter           | 0                    | 0                      |
| <b>525810</b> | 568451 | B003S1WTCU | A3I8AFVPEE8KI5 | R. Sawyer                  | 0                    | 0                      |
| <b>525811</b> | 568452 | B004I613EE | A121AA1GQV751Z | pksd<br>"pk_007"           | 2                    | 2                      |
| <b>525812</b> | 568453 | B004I613EE | A3IBEVCTXKNOH  | Kathy A. Welch<br>"katwel" | 1                    | 1                      |
| <b>525813</b> | 568454 | B001LR2CU2 | A3LGQPJCZVL9UC | srfell17                   | 0                    | 0                      |



```
In [118]: final['cleanReview'][0]
```

```
Out[118]: 'good quality dog food bought several vitality canned dog food products found g
ood quality product looks like stew processed meat smells better labrador finic
ky appreciates product better'
```

```
In [119]: final['lengthOfReview'][0]
```

```
Out[119]: 27
```

```
In [2]: dir_path = os.getcwd()
# conn = sqlite3.connect(os.path.join(dir_path, '/content/gdrive/My Drive/Colab N
conn = sqlite3.connect(os.path.join(dir_path, 'final.sqlite'))
# final.to_sql('Reviews', conn, if_exists='replace', index=False)
```

```
In [3]: review_3 = pd.read_sql_query(""" SELECT count(*) FROM Reviews""", conn)
print(review_3)
```

```
count(*)
0      364171
```

```
In [7]: filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews""", conn)
```

```
In [8]: filtered_data.shape
```

```
Out[8]: (364171, 13)
```

```
In [9]: filtered_data["Time"] = pd.to_datetime(filtered_data["Time"], unit = "s")
filtered_data = filtered_data.sort_values(by = "Time")
```



In [10]: `filtered_data.head(5)`

Out[10]:

|               | <b>Id</b> | <b>ProductId</b> | <b>UserId</b>  | <b>ProfileName</b>             | <b>HelpfulnessNumerator</b> | <b>Hel</b> |
|---------------|-----------|------------------|----------------|--------------------------------|-----------------------------|------------|
| <b>117924</b> | 150524    | 0006641040       | ACITT7DI6IDDL  | shari<br>zychinski             | 0                           | 0          |
| <b>117901</b> | 150501    | 0006641040       | AJ46FKXOVC7NR  | Nicholas A<br>Mesiano          | 2                           | 2          |
| <b>298792</b> | 451856    | B00004CXX9       | AIUWLEQ1ADEG5  | Elizabeth<br>Medina            | 0                           | 0          |
| <b>169281</b> | 230285    | B00004RYGX       | A344SMIA5JECGM | Vincent P.<br>Ross             | 1                           | 2          |
| <b>298791</b> | 451855    | B00004CXX9       | AJH6LUC1UT1ON  | The<br>Phantom of<br>the Opera | 0                           | 0          |

```
In [11]: print(len(filtered_data))
filtered_data.info()
filtered_data = filtered_data.head(100000)
print(len(filtered_data))
```

```
364171
<class 'pandas.core.frame.DataFrame'>
Int64Index: 364171 entries, 117924 to 107253
Data columns (total 13 columns):
Id                364171 non-null int64
ProductId         364171 non-null object
UserId           364171 non-null object
ProfileName       364171 non-null object
HelpfulnessNumerator 364171 non-null int64
HelpfulnessDenominator 364171 non-null int64
Score            364171 non-null int64
Time             364171 non-null datetime64[ns]
Summary          364171 non-null object
Text             364171 non-null object
cleanReview       364171 non-null object
lengthOfReview    364171 non-null int64
Preprocessed_text 364171 non-null object
dtypes: datetime64[ns](1), int64(5), object(7)
memory usage: 38.9+ MB
100000
```

```
In [12]: filtered_data['Score'].value_counts()
```

```
Out[12]: 1    87729
0     12271
Name: Score, dtype: int64
```

```
In [13]: X = filtered_data["cleanReview"]
print(print("shape of X:", X.head(5)))
y = filtered_data["Score"]
print("shape of y:", y.head(5))
X_len = filtered_data['lengthOfReview']
```

```
shape of X: 117924    EVERY book is educational. this witty little b...
117901    This whole series is great way to spend time w...
298792    Entertainingl Funny!. Beetlejuice is a well wr...
169281    A modern day fairy tale. A twist of rumplestis...
298791    FANTASTIC!. Beetlejuice is an excellent and fu...
Name: cleanReview, dtype: object
None
shape of y: 117924    1
117901    1
298792    1
169281    1
298791    1
Name: Score, dtype: int64
```

```
In [14]: len(filtered_data['lengthOfReview'])
```

```
Out[14]: 100000
```

```
In [15]: X_train = X[0:100000]
        Y_train = y[0:100000]
```

```
In [16]: print(len(X_train))
        print(len(Y_train))
```

```
100000
100000
```

```
In [17]: print(X_train.shape)
        print(X_train.shape)
```

```
(100000,)
(100000,)
```

## Bag of Words

```
In [15]: from sklearn.feature_extraction.text import CountVectorizer

        count_vect = CountVectorizer()
        X_train_vect = count_vect.fit_transform(X_train)
        X_test_vect = count_vect.transform(X_test)
        X_val_vect = count_vect.transform(X_val)
        feature_names = count_vect.get_feature_names()
        # Bow_dict = {'X_train_vect':X_train_vect, 'X_test_vect': X_test_vect, 'X_val_vect': X_val_vect}
        print(X_train_vect.shape)
        # print(feature_names)

        (60000, 48270)
```

```
In [16]: X_train_vect.shape
```

```
Out[16]: (60000, 48270)
```

```
In [18]: len(filtered_data['lengthOfReview'])
```

```
Out[18]: 100000
```

```
In [191]: from scipy.sparse import hstack
          # len_review = final['lengthOfReview'].to_sparse()
          concat_data = hstack((X_train_vect,np.array(filtered_data['lengthOfReview'])[0:60000]))
          concat_data_val = hstack((X_val_vect,np.array(filtered_data['lengthOfReview'])[60000:80000]))
          concat_data_test = hstack((X_test_vect,np.array(filtered_data['lengthOfReview'])[80000:100000]))
```

```
In [193]: print(concat_data.shape)
          print(concat_data_val.shape)
          print(concat_data_test.shape)
```

```
(60000, 47536)
(20000, 47536)
(20000, 47536)
```

```
In [194]: print(len(feature_names))
```

```
47535
```

```
In [195]: BoW_dict = {'X_train_vect':concat_data, 'X_test_vect': concat_data_test, 'X_val_v
print(BoW_dict['X_train_vect'].shape)
```

```
(60000, 47536)
```

```
In [196]: import pickle
with open('BoW.pkl', 'wb') as handle:
    pickle.dump(BoW_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

## TF-IDF

```
In [18]: tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
train_tf_idf = tf_idf_vect.fit_transform(X_train)

print("the shape of out text TFIDF vectorizer ",train_tf_idf.get_shape())
print("the type of count vectorizer ",type(train_tf_idf))
print("the number of unique words including both unigrams and bigrams ", train_tf
```

```
the shape of out text TFIDF vectorizer (100000, 94772)
the type of count vectorizer <class 'scipy.sparse.csr.csr_matrix'>
the number of unique words including both unigrams and bigrams 94772
```

```
In [19]: tf_idf_dict = {'train_tf_idf': train_tf_idf}
```

```
In [19]: import pickle
with open('tf_idf.pkl', 'wb') as handle:
    pickle.dump(tf_idf_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

## TruncatedSVD on tfidf

```
In [20]: import pickle
# with open(r"/content/gdrive/My Drive/Colab Notebooks/Assignment 4/tf_idf.pkl",
with open(r"tf_idf.pkl", "rb") as input_file:
    tfidf_dict = pickle.load(input_file)
```

```
In [21]: from sklearn.decomposition import TruncatedSVD
```

```
svd = TruncatedSVD()
svd.fit(train_tf_idf)
```

```
Out[21]: TruncatedSVD(algorithm='randomized', n_components=2, n_iter=5,
    random_state=None, tol=0.0)
```

```
In [22]: idf = tf_idf_vect.idf_
y = dict(zip(tf_idf_vect.get_feature_names(), idf))
```

```
Out[26]: ['greasiness',
          'aloe vera',
          'these except',
          'verona is',
          'style with',
          'light tuna',
          'bottle up',
          'can amazon',
          'has seen',
          'to mac',
          'after heating',
          'pan in',
          'she isn',
          'even offer',
          'antioxidants it',
          'and quiet',
          'was holding',
          'have traveled',
          'probably still',
          'brown mustard']
```

## co-occurrence matrix

```
100%|███████████████████████████████████████████████████████████████████████████|
██████████████████████████████████████████████████████████████████████████████| 100000/100000 [1:16:08<00:00, 21.89it/s]
```

```
In [30]: import pickle
file = open('my_dataframe', 'wb')
pickle.dump(df, file)
file.close()
```

```
In [33]: cov_df.head(5)
```

```
Out[33]:
```

|              | greasiness | aloe vera | these except | verona is | style with | light tuna | bottle up | can amazon | has seen | to mac | ... | pl |
|--------------|------------|-----------|--------------|-----------|------------|------------|-----------|------------|----------|--------|-----|----|
| greasiness   | 6          | 0         | 0            | 0         | 0          | 0          | 0         | 0          | 0        | 0      | ... | 0  |
| aloe vera    | 0          | 0         | 0            | 0         | 0          | 0          | 0         | 0          | 0        | 0      | ... | 0  |
| these except | 0          | 0         | 0            | 0         | 0          | 0          | 0         | 0          | 0        | 0      | ... | 0  |
| verona is    | 0          | 0         | 0            | 0         | 0          | 0          | 0         | 0          | 0        | 0      | ... | 0  |
| style with   | 0          | 0         | 0            | 0         | 0          | 0          | 0         | 0          | 0        | 0      | ... | 0  |

5 rows × 2000 columns

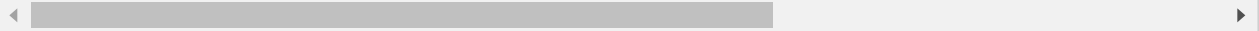


```
In [34]: cov_df.shape
```

```
Out[34]: (2000, 2000)
```

```
In [27]: from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
```

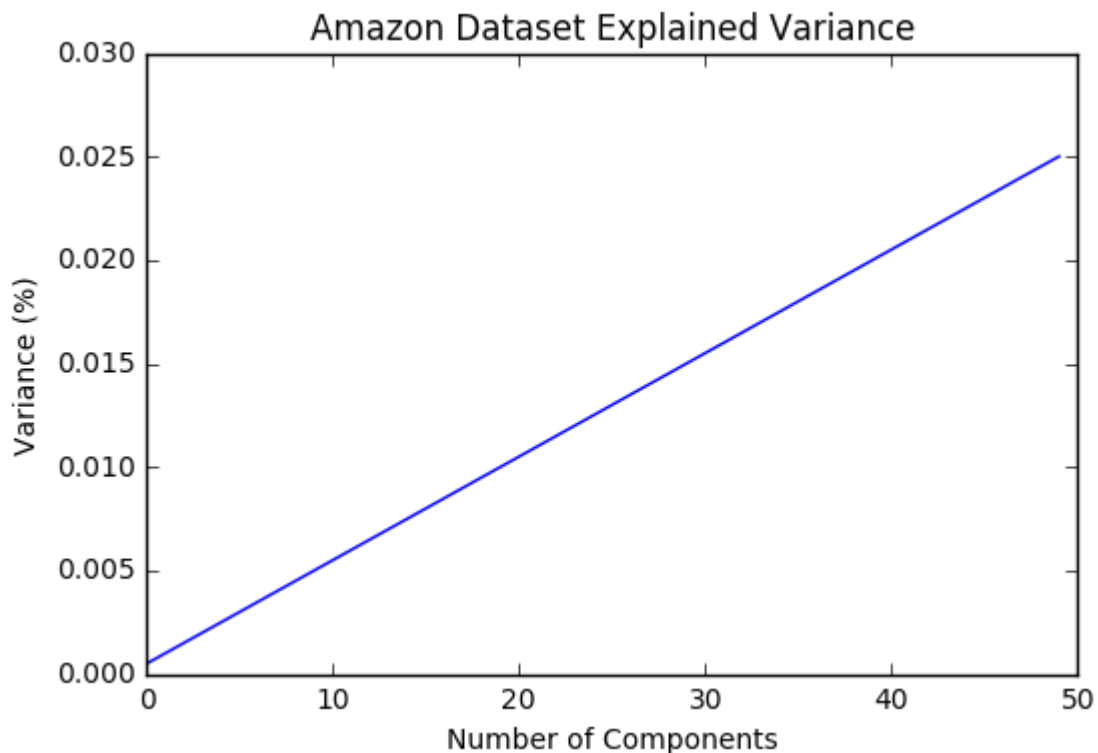
```
In [85]: #https://towardsdatascience.com/an-approach-to-choosing-the-number-of-components-
#Fitting the Truncated SVD algorithm with our Data
tsvd = TruncatedSVD(n_components=50).fit(m)
```



```
In [86]: tsvd.explained_variance_ratio_
```

```
Out[86]: array([0.00050035, 0.00049886, 0.00050025, 0.00049901, 0.0005005 ,
0.00050034, 0.00050009, 0.00050044, 0.00050049, 0.0005001 ,
0.00050048, 0.0005005 , 0.00050046, 0.00049928, 0.00049991,
0.0005001 , 0.00050003, 0.00050045, 0.00050049, 0.00050034,
0.00049951, 0.00050048, 0.00050047, 0.00050014, 0.00050049,
0.00050048, 0.00050019, 0.00050013, 0.00049987, 0.00050045,
0.00050048, 0.0005005 , 0.00050023, 0.00050046, 0.00050039,
0.00050038, 0.00050012, 0.00050041, 0.00049976, 0.00050033,
0.00050045, 0.00050037, 0.00050025, 0.00050045, 0.00050028,
0.00050035, 0.0005005 , 0.00050032, 0.00049854, 0.00049977])
```

```
In [87]: #Plotting the Cumulative Summation of the Explained Variance
plt.figure()
plt.plot(np.cumsum(tsvd.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Variance (%)') #for each component
plt.title('Amazon Dataset Explained Variance')
plt.show()
```



```
In [31]: best_svd = TruncatedSVD(n_components=1).fit(train_tf_idf)
```

```
In [ ]: from sklearn.cluster import KMeans
from sklearn.model_selection import GridSearchCV
from tqdm import tqdm
tfidf_k_inertia_train = dict()

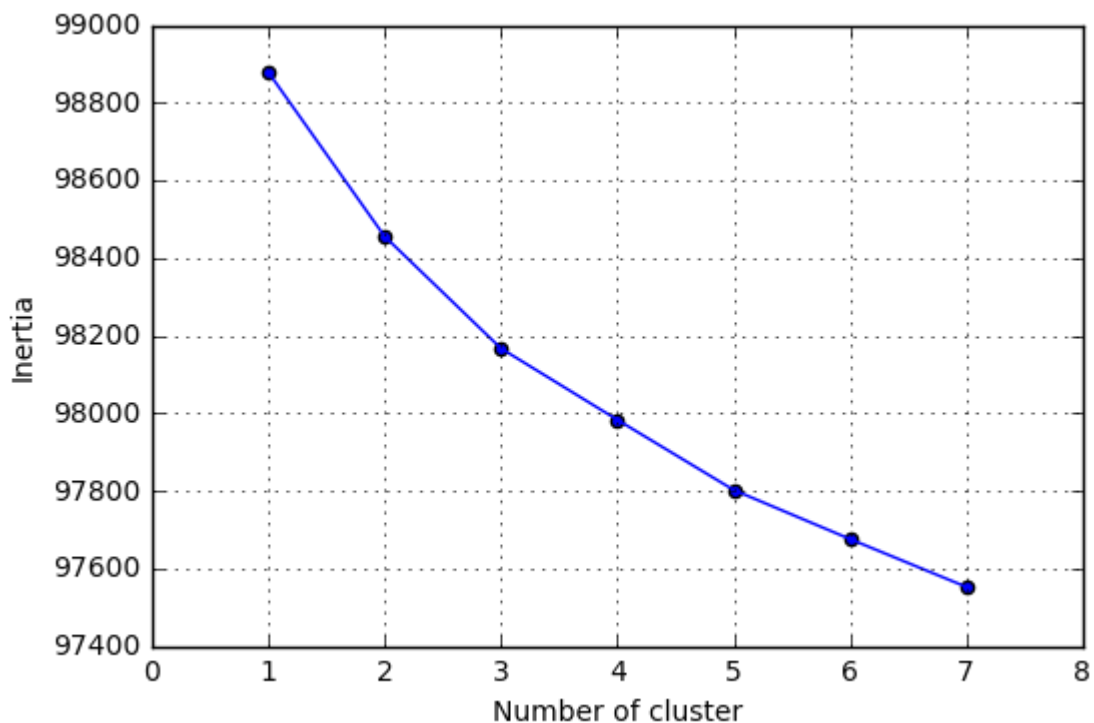
for k_val in range(1, 8):
    tfidf_km_clf = KMeans(n_clusters = k_val, n_jobs = -1)
    tfidf_km_clf.fit(train_tf_idf)
    tfidf_k_inertia_train[k_val] = (tfidf_km_clf.inertia_)
```

```
In [33]: tfidf_k_inertia_train
```

```
Out[33]: {1: 98878.82946187379,
2: 98455.39946212494,
3: 98166.69384298568,
4: 97982.62814136724,
5: 97801.75092993751,
6: 97675.24596462022,
7: 97551.88000632105}
```

```
In [34]: with open('cluster_dict.pkl', 'wb') as handle:
         pickle.dump(tfidf_k_inertia_train, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

```
In [35]: plt.figure()
         plt.plot(list(tfidf_k_inertia_train.keys()), list(tfidf_k_inertia_train.values()))
         plt.scatter(list(tfidf_k_inertia_train.keys()), list(tfidf_k_inertia_train.values))
         plt.xlabel("Number of cluster")
         plt.ylabel("Inertia")
         plt.grid()
         plt.show()
```



Training with 7 clusters

```
In [ ]: import datetime

         t1 = datetime.datetime.now()
         tfidf_final_clf = KMeans(n_clusters = 7, n_jobs = -1)
         tfidf_final_clf.fit(train_tf_idf)
         print("time required = ",datetime.datetime.now() - t1 )

         time required = 0:48:57.536801
```



```
In [44]: from wordcloud import WordCloud
imp_features_tfidf = []
print("Top terms per cluster:")
order_centroids = tfidf_final_clf.cluster_centers_.argsort()[:, :-1]
terms = tf_idf_vect.get_feature_names()
print(len(terms))
for i in range(7):
    for ind in order_centroids[i, :20]:
        imp_features_dict = {}
        imp_features_dict[i] = terms[ind-1]
        imp_features_tfidf.append(imp_features_dict)
```

Top terms per cluster:  
59901

```
In [45]: len(imp_features_tfidf)
```

Out[45]: 140

```
In [53]: cl1 = [d[0] for d in imp_features_tfidf if 0 in d]
cl2 = [d[1] for d in imp_features_tfidf if 1 in d]
cl3 = [d[2] for d in imp_features_tfidf if 2 in d]
cl4 = [d[3] for d in imp_features_tfidf if 3 in d]
cl5 = [d[4] for d in imp_features_tfidf if 4 in d]
cl6 = [d[5] for d in imp_features_tfidf if 5 in d]
cl7 = [d[6] for d in imp_features_tfidf if 6 in d]
print(cl1, cl2, cl3, cl4, cl5, cl6)
```

['doesnt taste', 'doggy', 'fong sriracha', 'treatments', 'casual', 'catnip', 'l  
overs try', 'nostalgic', 'treasures', 'lousy', 'dog lover', 'dogs little', 'omi  
tted', 'greasy salty', 'dog favorite', 'easy yummy', 'lightweight', 'worthy',  
'vessel', 'gesture'] ['glutamic acid', 'gluten flour', 'fred', 'brazilian', 'mi  
tigate', 'nostalgic', 'cookie would', 'greasy salty', 'besides great', 'past ye  
ars', 'whatsoever', 'goo', 'best glad', 'flossies', 'tast', 'lightweight', 'gev  
alia signature', 'mak', 'celestial seasons', 'easy yummy'] ['chocolat', 'nostal  
gic', 'darjeeling teas', 'dark brown', 'hot chili', 'hosting', 'banned', 'milit  
ary', 'lightweight', 'coco', 'goo', 'besides great', 'tast', 'barry tea', 'cook  
ie would', 'greasy salty', 'flavonoids', 'lousy', 'delicioso', 'sweep'] ['nosta  
lgic', 'goo', 'lightweight', 'tast', 'greasy salty', 'flavonoids', 'lousy', 'be  
sides great', 'omitted', 'worthy', 'realizing', 'delicioso', 'nj', 'sufficientl  
y', 'litters', 'usda organic', 'sweep', 'gesture', 'easy yummy', 'mak'] ['produ  
cing', 'greasy salty', 'amazingly tasty', 'prey', 'nostalgic', 'financial', 'sh  
ipper', 'goo', 'orchards', 'buttons', 'storage not', 'great problem', 'excellen  
ce', 'stored refrigerator', 'lousy', 'tim tams', 'grocers', 'lobster meat', 'wo  
rthy', 'besides great'] ['coffe', 'cumin', 'nostalgic', 'greasy salty', 'goo',  
'flavonoids', 'stroller', 'cup cocoa', 'lightweight', 'roaring', 'boils', 'besi  
des great', 'pod system', 'great coconut', 'tast', 'espressione', 'omitted', 's  
ense smell', 'darjeeling teas', 'cuppa tea']

```
In [54]: cl1_string = ' '.join(cl1)
cl2_string = ' '.join(cl2)
cl3_string = ' '.join(cl3)
cl4_string = ' '.join(cl4)
cl5_string = ' '.join(cl5)
cl6_string = ' '.join(cl6)
cl7_string = ' '.join(cl7)
```

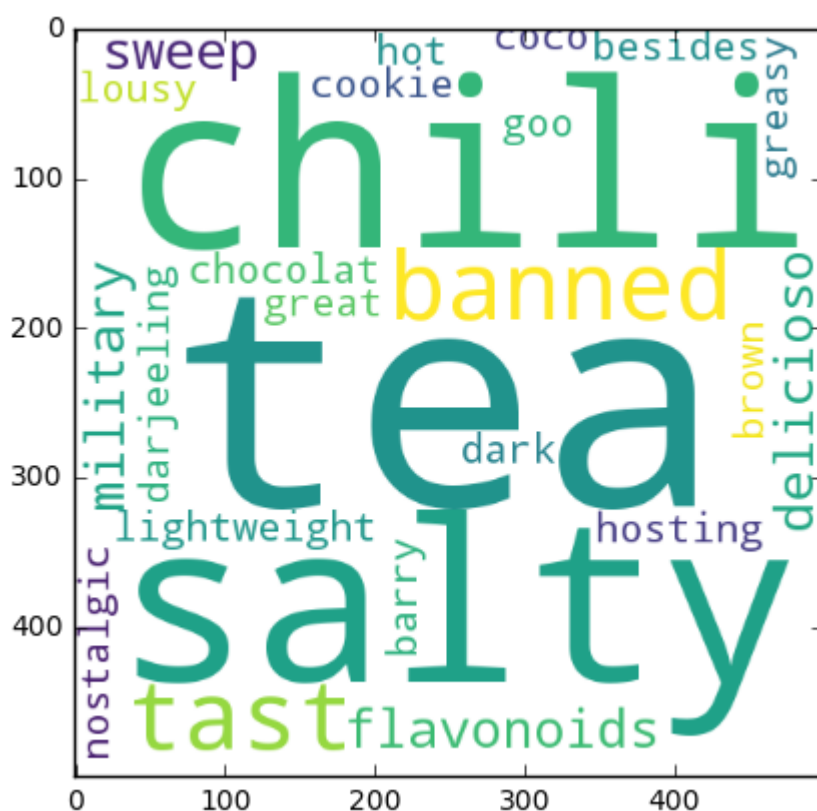
```
In [55]: from wordcloud import WordCloud
wordcloud_cluster1 = WordCloud(width = 500, height = 500, background_color = 'white')
wordcloud_cluster2 = WordCloud(width = 500, height = 500, background_color = 'white')
wordcloud_cluster3 = WordCloud(width = 500, height = 500, background_color = 'white')
wordcloud_cluster4 = WordCloud(width = 500, height = 500, background_color = 'white')
wordcloud_cluster5 = WordCloud(width = 500, height = 500, background_color = 'white')
wordcloud_cluster6 = WordCloud(width = 500, height = 500, background_color = 'white')
wordcloud_cluster7 = WordCloud(width = 500, height = 500, background_color = 'white')
# plot the WordCloud image
plt.imshow(wordcloud_cluster1)
plt.tight_layout(pad = 0)
plt.show()
```



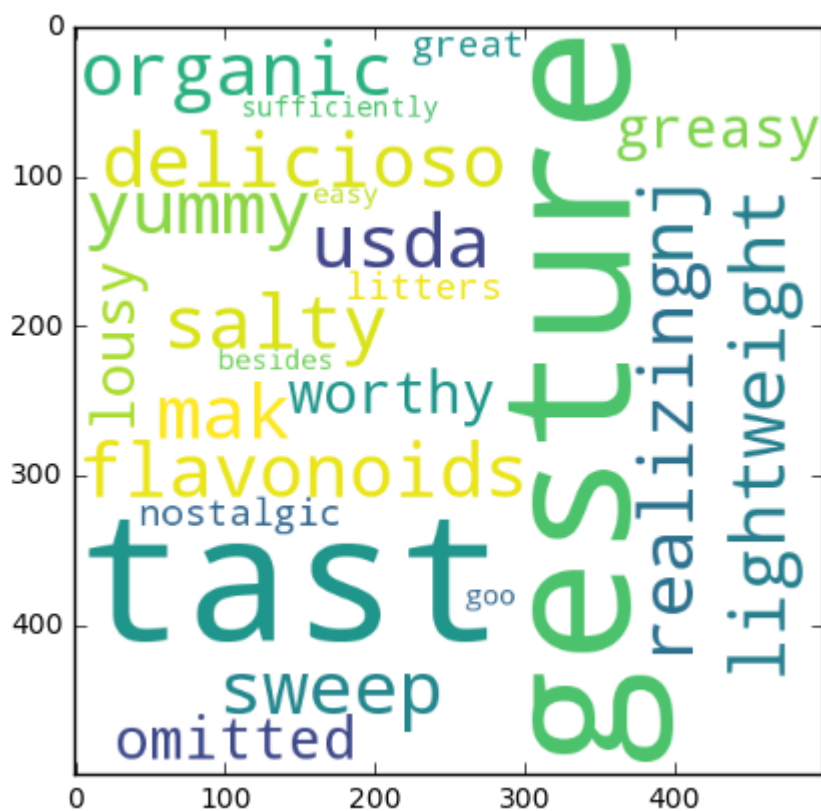
```
In [56]: plt.imshow(wordcloud_cluster2)  
plt.tight_layout(pad = 0)  
plt.show()
```



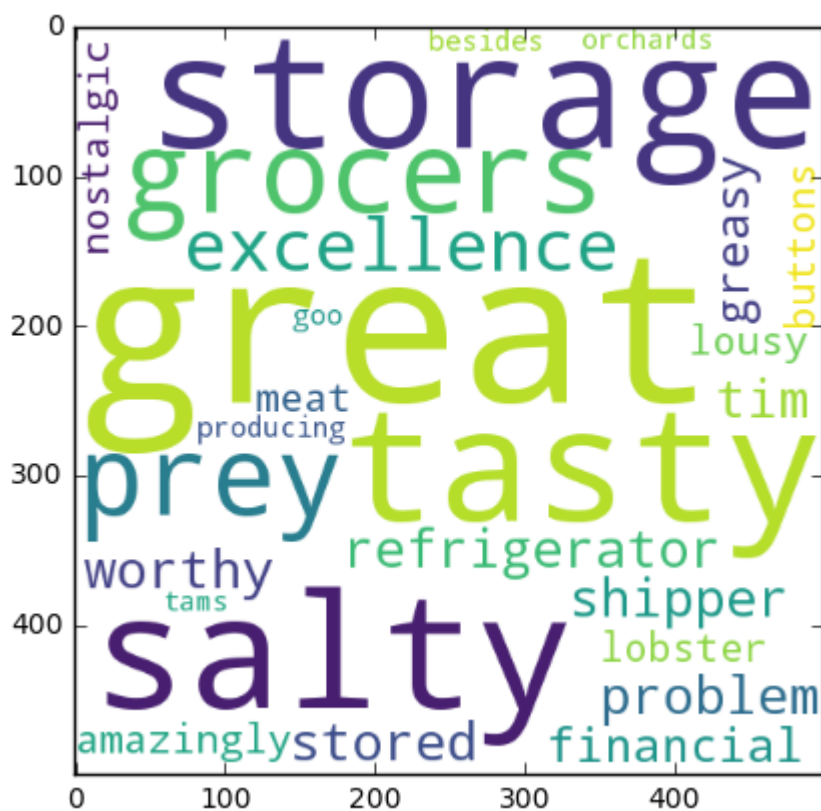
```
In [57]: plt.imshow(wordcloud_cluster3)
plt.tight_layout(pad = 0)
plt.show()
```



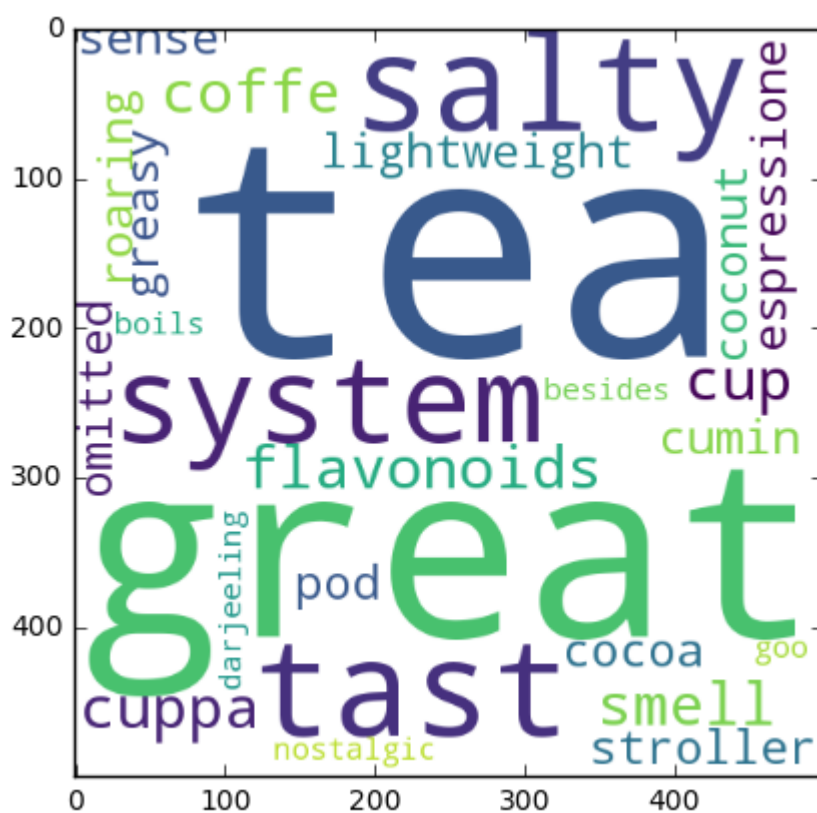
```
In [58]: plt.imshow(wordcloud_cluster4)
plt.tight_layout(pad = 0)
plt.show()
```



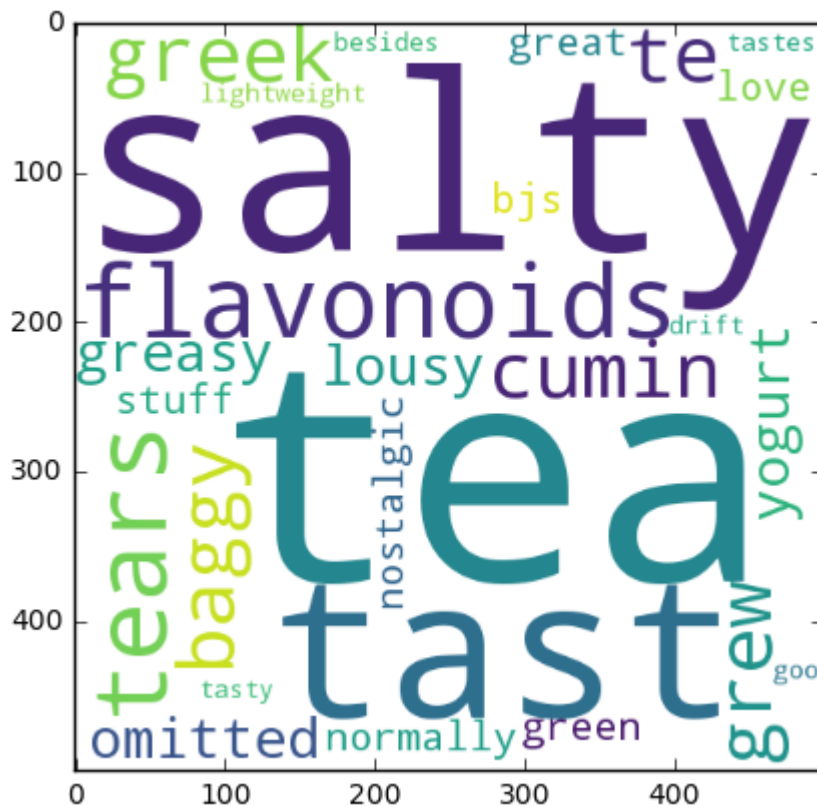
```
In [59]: plt.imshow(wordcloud_cluster5)  
plt.tight_layout(pad = 0)  
plt.show()
```



```
In [60]: plt.imshow(wordcloud_cluster6)
plt.tight_layout(pad = 0)
plt.show()
```



```
In [61]: plt.imshow(wordcloud_cluster7)
plt.tight_layout(pad = 0)
plt.show()
```



```
In [62]: #function that takes a word and returns the most similar words using cosine similarity
def cosine_similarity(a,b):
    return dot(a,b) / ( (dot(a,a) **.5) * (dot(b,b) **.5) )
```

### Observations

cluster 1 represents related to dog food eg. dog, doggy, yummy  
cluster 2 represents type of food eg. cookie, salty, brazilian  
cluster 3 represents taste of food eg. salty, chili, sweet, coco  
cluster 4 represents organic food eg. organic, great, worthy  
cluster 5 represents shipping related information eg. shipping, financial, refrigerator  
cluster 6 represents taste of coffee eg. coffee, cocoa, smell, stronger  
cluster 7 represents tea related information eg. tea, green, tasty