In [2]:

```
%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/database.sql
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 50000
# 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 rat
def partition(x):
    if x < 3:
        return 0
    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	HelpfulnessDenom
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	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"}, keep='fi
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 left
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, 348689.13it/s]
```

In [20]:

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

```
100%|████████████████████████████████████████████████████████████████████████
███████████████████████████████████████████████████████| 364171/364171 [00:00
<00:00, 545983.46it/s]
```

In [21]:

```
from bs4 import BeautifulSoup
text_lst = []
for text in tqdm(removed_urls_list):
  soup = BeautifulSoup(text, 'lxml')
  text = soup.get_text()
  text_lst.append(text)
# print(text)
# print("="*50)
```

```
100%|████████████████████████████████████████████████████████████████████████
█████████████████████████████████████████████████████████| 364171/364171 [02:
07<00:00, 2863.46it/s]
```

In [22]:

```
print(len(final['cleanReview']))
```

```
364171
```

In [23]:

```
# 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 [24]:

```
decat_lst = []
for decat_text in tqdm(text_lst):
  text = decontracted(decat_text)
  decat_lst.append(text)
```

```
100%|███████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████| 364171/364171 [00:0
5<00:00, 67496.57it/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)
```

```
100%|███████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████| 364171/364171 [00:1
9<00:00, 18494.66it/s]
```

In [26]:

```
spatial_list = []
for to_spatial in tqdm(strip_list):
  text = re.sub('[^A-Za-z0-9]+', ' ', to_spatial)
  spatial_list.append(text)
```

```
100%|███████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████| 364171/364171 [00:0
9<00:00, 36464.50it/s]
```

In [27]:

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

In [28]:

```
# Combining all the above stundents
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(spatial_list):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').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 stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|████████████████████████████████████████████████████████████████████████
█████████████████████████████████████████████████████████| 364171/364171 [02:
01<00:00, 2999.66it/s]
```

In [29]:

```
print(len(preprocessed_reviews))
preprocessed_reviews[-1]
```

```
364171
```

Out[29]:

```
'great honey satisfied product advertised use cereal raw vinegar general swe
etner'
```

In [30]:

```
final['cleanReview'] = preprocessed_reviews
```

In [117]:

```
print(len(final))
final.tail(5)
```

```
364171
```

Out[117]:

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

In [118]:

```
final['cleanReview'][0]
```

Out[118]:

```
'good quality dog food bought several vitality canned dog food products foun
d good quality product looks like stew processed meat smells better labrador
finicky appreciates product better'
```

In [119]:

```
final['lengthOfReview'][0]
```

Out[119]:

```
27
```

In [3]:

```
dir_path = os.getcwd()
# conn = sqlite3.connect(os.path.join(dir_path, '/content/gdrive/My Drive/Colab Notebooks/A
conn = sqlite3.connect(os.path.join(dir_path, 'final.sqlite'))
# final.to_sql('Reviews', conn, if_exists='replace', index=False)
```

In [4]:

```
review_3 = pd.read_sql_query(""" SELECT count(*) FROM Reviews""", conn)
print(review_3)
```

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

In [5]:

```
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews""", conn)
```

In [6]:

```
filtered_data.shape
```

Out[6]:

```
(364171, 12)
```

In [7]:

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

In [8]:

```
filtered_data.head(5)
```

Out[8]:

ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score
006641040	ACITT7DI6IDDL	shari zychinski	0	0	1
006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	1
0004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0	1
0004RYGX	A344SMIA5JECGM	Vincent P. Ross	1	2	1
0004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	0	1

In [9]:

```
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 12 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
dtypes: datetime64[ns](1), int64(5), object(6)
memory usage: 36.1+ MB
100000
```

In [10]:

```
filtered_data['Score'].value_counts()
```

Out[10]:

```
1    87729
0    12271
Name: Score, dtype: int64
```

In [11]:

```
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 educational witty little book makes...
117901    whole series great way spend time child rememb...
298792    entertainingl funny beetlejuice well written m...
169281    modern day fairy tale twist rumplestiskin capt...
298791    fantastic beetlejuice excellent funny movie ke...
Name: cleanReview, dtype: object
None
shape of y: 117924    1
117901    1
298792    1
169281    1
298791    1
Name: Score, dtype: int64
```

In [12]:

```
len(filtered_data['lengthOfReview'])
```

Out[12]:

```
100000
```

In [13]:

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

In [14]:

```
print(len(X_train))
print(len(Y_train))
```

```
100000
100000
```

In [15]:

```
print(X_train.shape)
print(X_train.shape)
```

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

###

In [ ]:

```

```

In [ ]:

```

```

Bag of Words

In [17]:

```
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_
print(X_train_vect.shape)
# print(feature_names)
```

```
(100000, 61444)
```

In [18]:

```
X_train_vect.shape
```

Out[18]:

```
(100000, 61444)
```

In [19]:

```
len(filtered_data['lengthOfReview'])
```

Out[19]:

```
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])[:,Nor
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'][80000:10000
```

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_vect': conc
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 [20]:

```
tf_idf_vect = TfidfVectorizer(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_idf.get_s
```

```
the shape of out text TFIDF vectorizer  (100000, 12778)
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the number of unique words including both unigrams and bigrams  12778
```

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 [16]:

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

In [34]:

```
features = dict(zip(tf_idf_vect.get_feature_names(), tf_idf_vect.idf_))
top_features = sorted(list(features.items()), key=lambda x: x[1])
top_features
```

```
 ('sweetened', 6.039044768567954),
 ('kinds', 6.042135961137626),
 ('flavoring', 6.043685148124457),
 ('reviewers', 6.043685148124457),
 ('whether', 6.045236738815875),
 ('chance', 6.046790740682609),
 ('personally', 6.046790740682609),
 ('sized', 6.046790740682609),
 ('grew', 6.049906007999558),
 ('poor', 6.049906007999558),
 ('skin', 6.051467288566511),
 ('suggest', 6.051467288566511),
 ('offered', 6.053031010542694),
 ('salads', 6.057736901580107),
 ('berry', 6.059310466027537),
 ('feeding', 6.059310466027537),
 ('oats', 6.059310466027537),
 ('nature', 6.0608865104830025),
 ('sitting', 6.0624650427760525),
 ('celiac', 6.064046070773371),
```

In [62]:

```
top_features_list = [x[0] for x in top_features]
```

In [36]:

```
co_occurence_matrix = np.zeros((len(top_features), len(top_features)), np.int)
co_occ_df = pd.DataFrame(co_occurence_matrix,index=top_features_list,columns=top_features_l
co_occ_df.head()
```

Out[36]:

	not	great	good	like	taste	one	product	love	flavor	best	...	yearly	yell	yelling
not	0	0	0	0	0	0	0	0	0	0	...	0	0	0
great	0	0	0	0	0	0	0	0	0	0	...	0	0	0
good	0	0	0	0	0	0	0	0	0	0	...	0	0	0
like	0	0	0	0	0	0	0	0	0	0	...	0	0	0
taste	0	0	0	0	0	0	0	0	0	0	...	0	0	0

5 rows × 12778 columns

In [37]:

```
for sent in tqdm(X.values.tolist()):
    words = sent.split(" ")
    for i in range(len(words)):
        for j in range(i-5, i+6):
            if j>=0 and j<len(words):
                if words[i] in top_features_list and words[j] in top_features_list and worc
                    co_occ_df.loc[words[i], words[j]] += 1
```

```
100%|███████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████| 100000/100000 [3:0
8:41<00:00,  8.83it/s]
```

In [39]:

```
co_occ_df.head(10)
```

Out[39]:

	not	great	good	like	taste	one	product	love	flavor	best	...	yearly	yell
not	0	6707	10991	15505	10540	6848	6552	4387	7408	3020	...	1	3
great	6707	0	3225	2707	5336	1799	6729	3257	3476	1405	...	0	0
good	10991	3225	0	3894	4946	2342	3558	1744	3142	1274	...	2	0
like	15505	2707	3894	0	6620	2721	2033	1902	3418	1373	...	0	0
taste	10540	5336	4946	6620	0	1638	1581	1872	1945	1094	...	0	3
one	6848	1799	2342	2721	1638	0	1243	1353	1513	2286	...	0	1
product	6552	6729	3558	2033	1581	1243	0	1949	955	1118	...	0	0
love	4387	3257	1744	1902	1872	1353	1949	0	1724	1432	...	0	0
flavor	7408	3476	3142	3418	1945	1513	955	1724	0	1246	...	0	2
best	3020	1405	1274	1373	1094	2286	1118	1432	1246	0	...	0	1

10 rows × 12778 columns

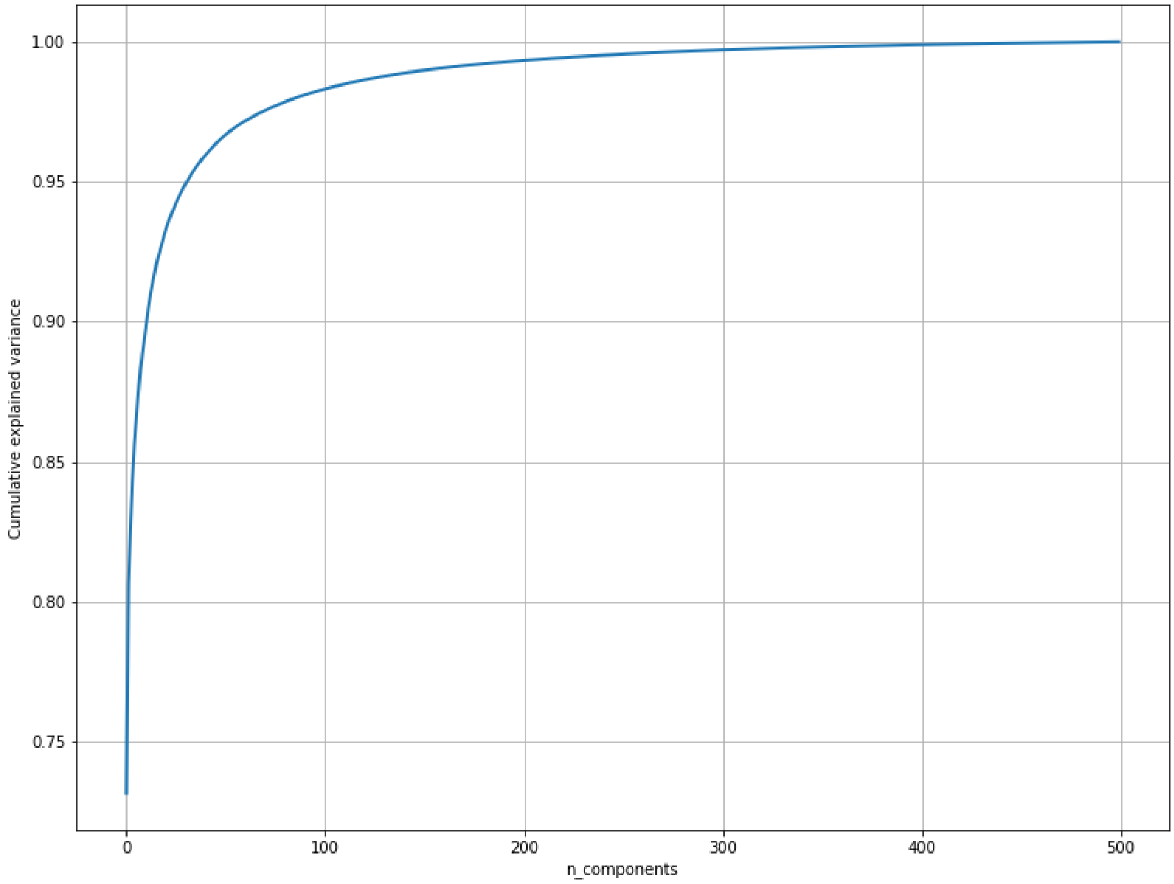
In [41]:

```
from sklearn.decomposition import TruncatedSVD

tsvd_dict = {}
for i in tqdm(range(1, 200, 10)):
    tsvd_dict[i] = TruncatedSVD(n_components=i).fit(co_occ_df.values)
```

```
100%|███████████████████████████████████████████████████████████████
██████████████████████████████████████████████████████████| 20/20 [0
5:19<00:00, 17.89s/it]
```

In [42]:

```
svd = TruncatedSVD(n_components=500).fit(co_occ_df.values)
var_perc = svd.explained_variance_ / np.sum(svd.explained_variance_);
cum_var = np.cumsum(var_perc)

plt.figure(figsize=(12.6, 9.8))
plt.plot(cum_var, linewidth=2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative explained variance')
plt.show()
```



95% variance can be explained by 50 n_components

K-Means

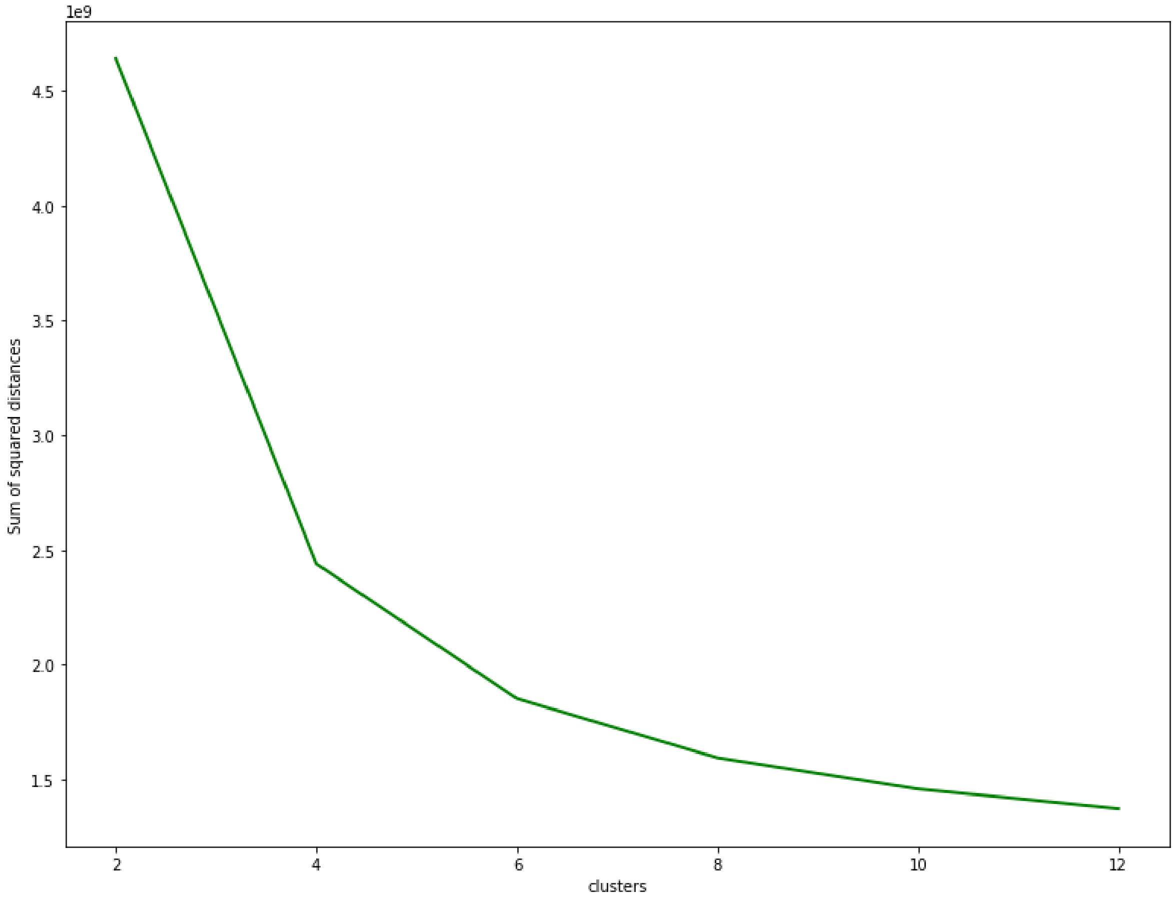
In [57]:

```
from sklearn.cluster import KMeans
from wordcloud import WordCloud

squared_distances_list = []
all_models = {}
for k in tqdm(range(2, 14, 2)):
    classifier = KMeans(n_clusters=k)
    classifier.fit(co_occ_df.values)
    squared_distances_list.append(classifier.inertia_)
    all_models[k] = classifier
```

```
100%|██████████████████████████████████
████████████████████████████████| 6/6 [1
0:19<00:00, 111.71s/it]
```

In [58]:

```
plt.figure(figsize=(13, 10))
plt.plot(range(2, 14, 2), squared_distances_list, color='green', lw=2)
plt.xlabel('clusters')
plt.ylabel('Sum of squared distances')
plt.show()
```



In [60]:

```
classifier = all_models[4]
c_desc = classifier.cluster_centers_.argsort()[:, ::-1]
```

In [63]:

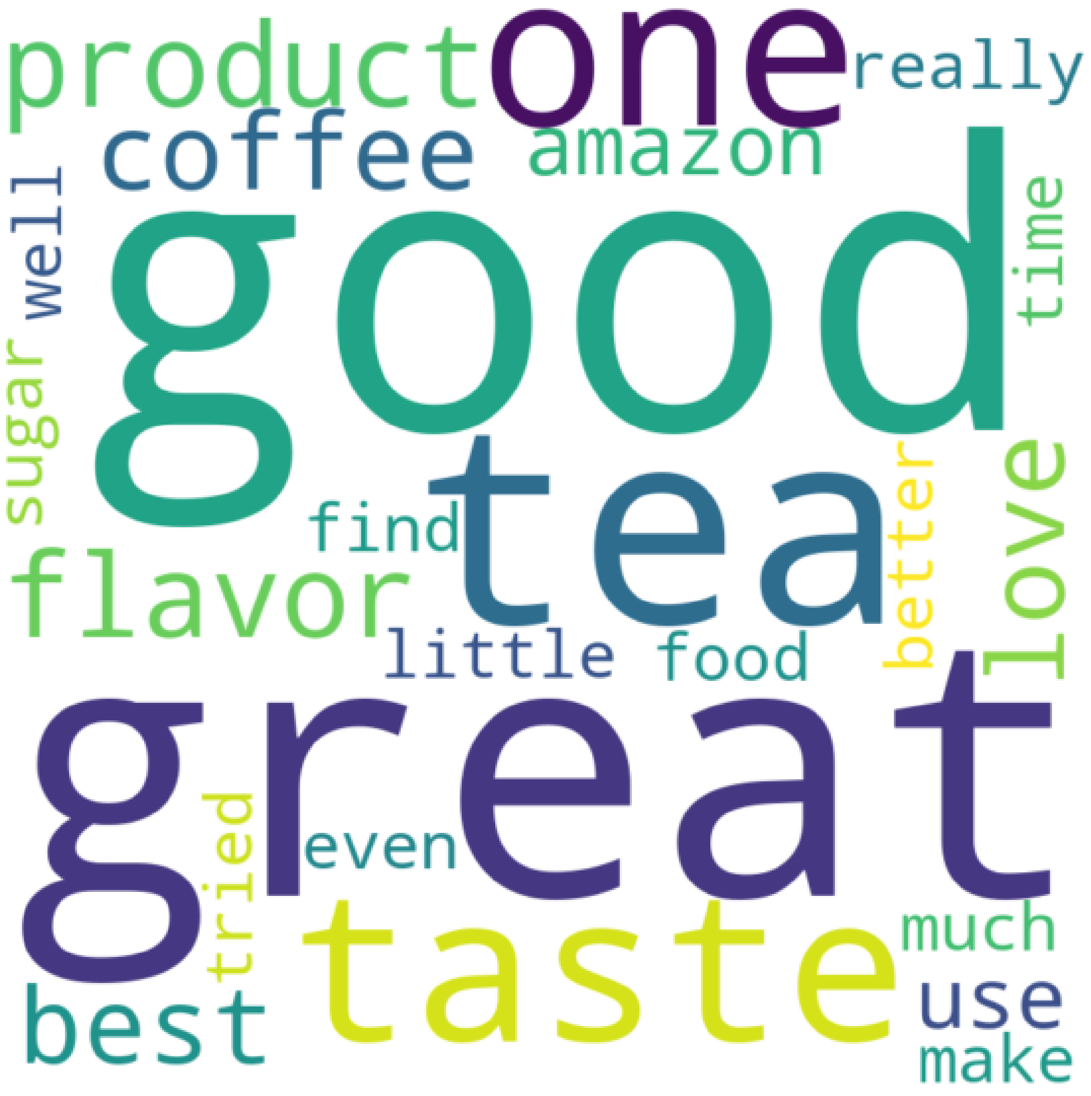
```
cluster_features = {}
for cluster in range(4):
    feature_list = []
    for index in c_desc[cluster, :30]:
        feature_list.append(top_features_list[index])
    cluster_features[cluster] = feature_list
```

In [64]:

```
wc1 = WordCloud(width=800, height=800, background_color='white', min_font_size=10).generate
wc2 = WordCloud(width=800, height=800, background_color='white', min_font_size=10).generate
wc3 = WordCloud(width=800, height=800, background_color='white', min_font_size=10).generate
wc4 = WordCloud(width=800, height=800, background_color='white', min_font_size=10).generate
```

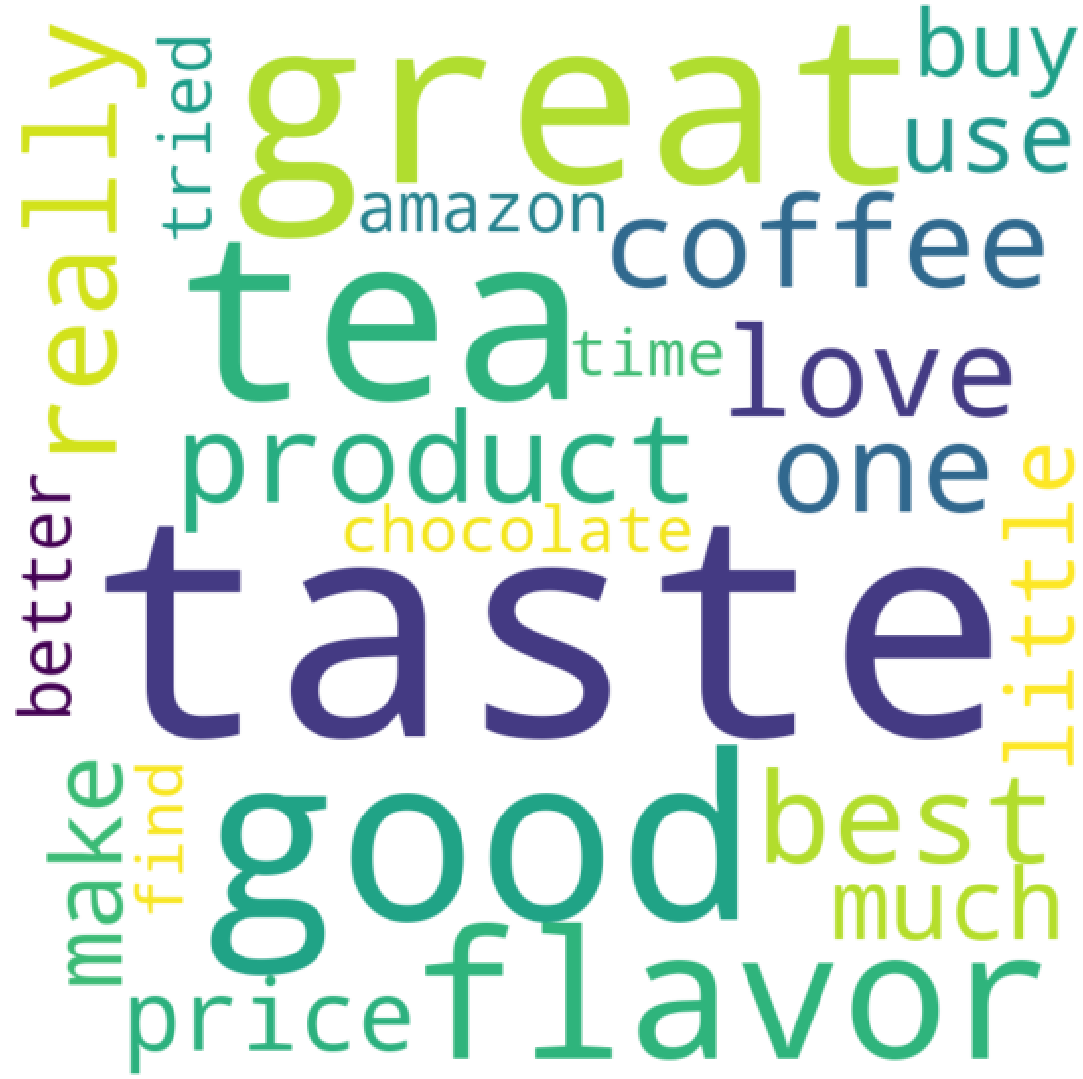
In [65]:

```
plt.figure(figsize=(8,8), facecolor=None)
plt.imshow(wc1, interpolation='bilinear')
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()
```



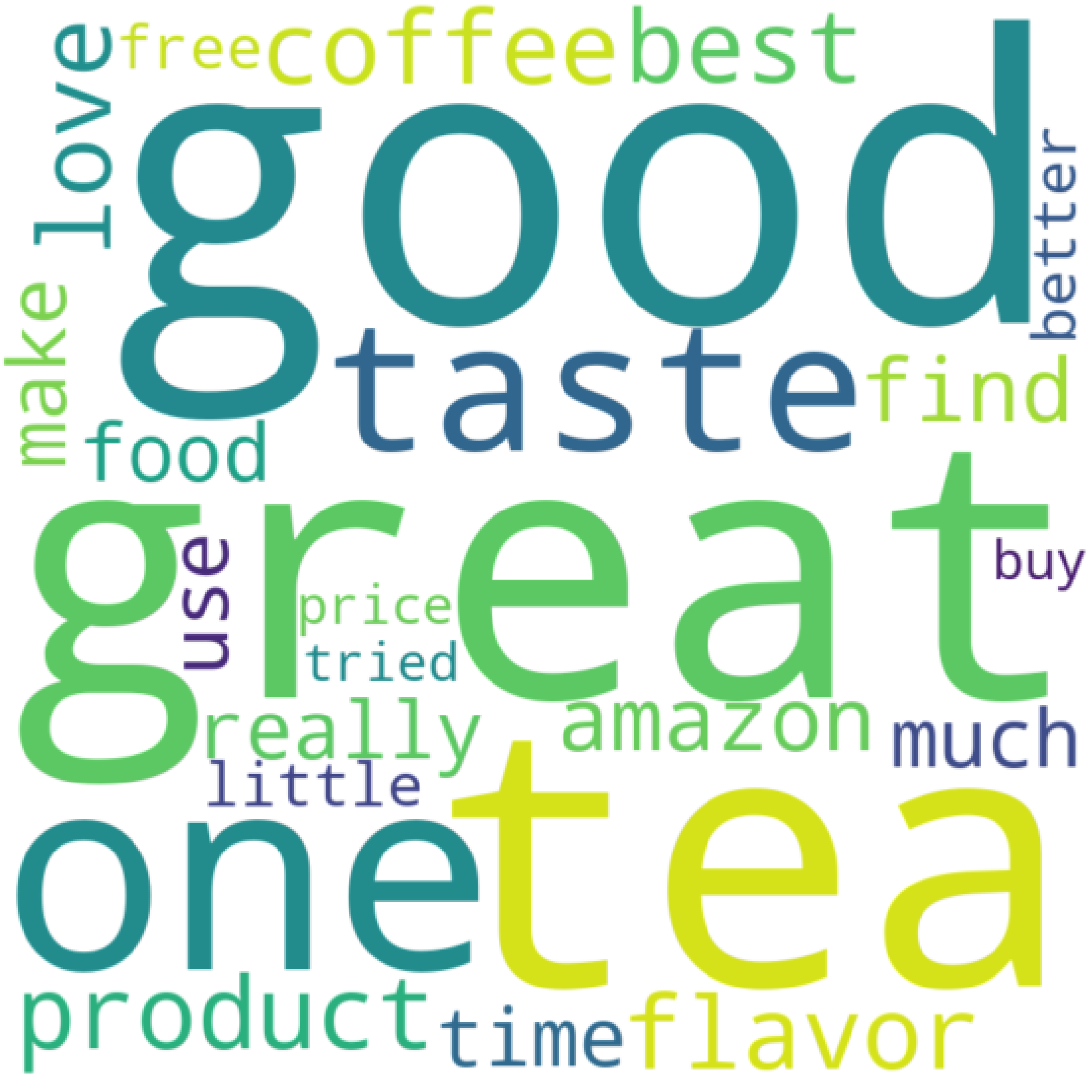
In [66]:

```
plt.figure(figsize=(8,8), facecolor=None)
plt.imshow(wc2, interpolation='bilinear')
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()
```



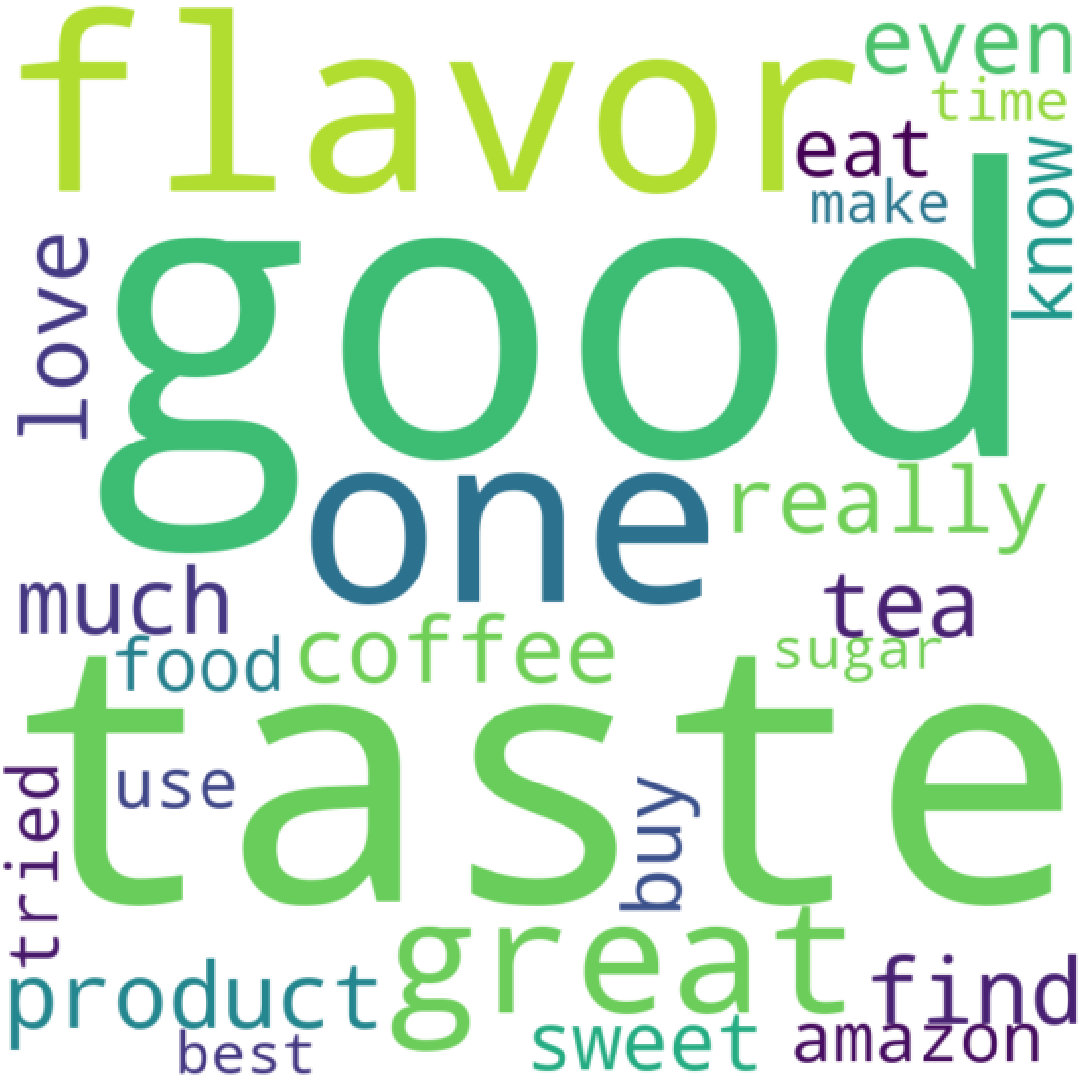
In [67]:

```
plt.figure(figsize=(8,8), facecolor=None)
plt.imshow(wc3, interpolation='bilinear')
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()
```



In [68]:

```
plt.figure(figsize=(8,8), facecolor=None)
plt.imshow(wc4, interpolation='bilinear')
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()
```



Similarity Function

In [54]:

```
from sklearn.metrics.pairwise import cosine_similarity

co_mat = co_occ_df.values

def similar_words(w):
    sim = cosine_similarity(co_mat)
    word_vect = sim[top_features.index(w)]
    i = word_vect.argsort()[::-1][0:10]
    top_sim_words = [top_features[i[x]] for x in range(len(i))]
    return top_sim_words
```

In [55]:

```
print(top_features[1000])
similar_words(top_features[1000])
```

```
('grown', 6.158565424122878)
```

Out[55]:

```
[('grown', 6.158565424122878),
 ('fact', 4.714002154879012),
 ('made', 3.583367757094892),
 ('fine', 4.775319182062324),
 ('good', 2.1974044889158137),
 ('lot', 4.002766888440956),
 ('course', 5.1579335438149725),
 ('enjoy', 3.9959422935066584),
 ('however', 4.119719453323702),
 ('kind', 4.694505192849572)]
```

Steps to complete assignment

1. Applied tfidf on 100k data points.
2. Took top 3000 features based on idf values.
3. Created co-occurence matrix of the top features.
4. Applied Truncated SVD on tfidf.
5. Applied Kmeans on co-occurence matrix.
6. Plotted word clouds for each cluster.
7. Created a similarity function.