Amazon Fine Food Reviews Analysis

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

EDA: 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 [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.co
m/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
import itertools
from nltk import ngrams
from wordcloud import WordCloud
import pickle

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

In [2]:

```
#loading train and test and validation dataset

file=open("x_train.pkl","rb")
x_train=pickle.load(file) # loading 'train' dataset

file=open("x_cv.pkl",'rb')
x_cv=pickle.load(file) # loading 'validation' dataset

file=open("x_test.pkl",'rb')
x_test=pickle.load(file) # loading 'test' dataset
```

```python
file=open("y_train.pkl","rb")
y_train=pickle.load(file) # loading 'train' dataset

file=open("y_cv.pkl",'rb')
y_cv=pickle.load(file) # loading 'validation' dataset

file=open("y_test.pkl",'rb')
y_test=pickle.load(file) # loading 'test' dataset

#loading train_bow and test _bow
file=open('x_train_bow.pkl','rb')
x_train_bow=pickle.load(file)

file=open('x_test_bow.pkl','rb')
x_test_bow=pickle.load(file)

file=open('x_cv_bow.pkl','rb')
x_cv_bow=pickle.load(file)

#loading train_tf_idf and test_tf_idf
file=open('train_tf_idf.pkl','rb')
train_tf_idf=pickle.load(file)

file=open('cv_tf_idf.pkl','rb')
cv_tf_idf=pickle.load(file)

file=open('test_tf_idf.pkl','rb')
test_tf_idf=pickle.load(file)

#loading train_w2v and test_w2v
file=open('train_w2v.pkl','rb')
train_w2v=pickle.load(file)

file=open('cv_w2v.pkl','rb')
cv_w2v=pickle.load(file)

file=open('test_w2v.pkl','rb')
test_w2v=pickle.load(file)

#loading train_tf_idf_w2v and test_tf_idf_w2v
file=open('train_tf_idf_w2v.pkl','rb')
train_tf_idf_w2v=pickle.load(file)

file=open('cv_tf_idf_w2v.pkl','rb')
cv_tf_idf_w2v=pickle.load(file)

file=open('test_tf_idf_w2v.pkl','rb')
test_tf_idf_w2v=pickle.load(file)
```

[5] Assignment 11: Truncated SVD

1. **Apply Truncated-SVD on only this feature set:**
    - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)

    - **Procedure:**

- Take top 2000 or 3000 features from tf-idf vectorizers using idf_ score.
- You need to calculate the co-occurrence matrix with the selected features (Note: X.X^T doesn't give the co-occurrence matrix, it returns the covariance matrix, check these bolgs blog-1, blog-2 for more information)
- You should choose the n_components in truncated svd, with maximum explained variance. Please search on how to choose that and implement them. (hint: plot of cumulative explained variance ratio)
- After you are done with the truncated svd, you can apply K-Means clustering and choose the best number of clusters based on elbow method.
- Print out wordclouds for each cluster, similar to that in previous assignment.
- You need to write a function that takes a word and returns the most similar words using cosine similarity between the vectors(vector: a row in the matrix after truncatedSVD)

Truncated-SVD

[5.1] Taking top features from TFIDF, SET 2

In [14]:
```
# Using tfidf for choosing top 20000 feature to make  co-occura
nce matrix

tf=TfidfVectorizer(ngram_range=(1,1),max_features=2000)
temp=tf.fit(x_train)
vocab = tf.get_feature_names()
```

[5.2] Calulation of Co-occurrence matrix

steps to find co-occurance matrix

1. find vocabulary using tfidf or bow (Can limit max_features for computational strain eg, max_feature=20000)

1. make a dict of vocabluary named vocab

1. make psudo/zero matrix of shape(len(vocab),len(vocab)) named co_occurrence_matrix

1. for each reviews/sentence in traing data, make a contetx window of size k (I am using nltk.ngram for making context window of default k=2)

1. Update the co_occurance_matrix using context window of each reviews

In [88]:
```
# co_occurance matrix
def create_co_occurance_matrix(data, name="without_freq", windo
w_size=2, vocab_size=2000):
```

```python
    tf=TfidfVectorizer()
    tf.fit(data)

    idx = np.argsort(tf.idf_)[:vocab_size]  #limiting vocab_size using IDF_ value
    vocab = np.array(tf.get_feature_names())[idx] # vocabualary
    vocab_index = {word: i for i, word in enumerate(sorted(vocab))}

    co_occurance_matrix= np.zeros((len(vocab),len(vocab)),dtype=int)

    for sentence in data:
        sent=sentence.split()

        for i,word_i in enumerate(sent):
            start= max(0,i-window_size)
            end= min(i+window_size+1,len(sent))

            for j in range(start,end):
                word_j=sent[j]

                if (word_i != word_j) and (word_i in vocab) and (word_j in vocab):
                    w_i_index= vocab_index[word_i]
                    w_j_index= vocab_index[word_j]
                    co_occurance_matrix[w_i_index][w_j_index] += 1

                elif name=='with_freq' and (word_i == word_j) and (word_i in vocab):
                    w_i_index= vocab_index[word_i]
                    w_j_index= vocab_index[word_j]
                    co_occurance_matrix[w_i_index][w_j_index] += 1

    return co_occurance_matrix, vocab_index
```

In [86]:

```python
# toy dataset
data=["abc def ijk pqr","pqr klm opq","lmn pqr xyz abc def pqr abc"]

matrix, vocab_index = create_co_occurance_matrix(data, name='with_freq', window_size=2,vocab_size=3)
data_matrix = pd.DataFrame(matrix, index=vocab_index,
                           columns=vocab_index)

data_matrix
```

	abc	def	pqr
abc	3	3	3
def	3	2	2
pqr	3	2	4

In [90]:

```python
# toy dataset
data=["abc def ijk pqr","pqr klm opq","lmn pqr xyz abc def pqr abc"]
```

```
matrix, vocab_index = create_co_occurance_matrix(data, name='wi
thout_freq', window_size=2,vocab_size=3)
data_matrix = pd.DataFrame(matrix, index=vocab_index,
                             columns=vocab_index)

data_matrix
```

	abc	def	pqr
abc	0	3	3
def	3	0	2
pqr	3	2	0

In [91]:
```
# toy dataset
data=["abc def ijk pqr","pqr klm opq","lmn pqr xyz abc def pqr
 abc"]

matrix, vocab_index = create_co_occurance_matrix(data, name='wi
th_freq', window_size=2,vocab_size=200)
data_matrix = pd.DataFrame(matrix, index=vocab_index,
                             columns=vocab_index)

data_matrix
```

	abc	def	ijk	klm	lmn	opq	pqr	xyz
abc	3	3	1	0	0	0	3	1
def	3	2	1	0	0	0	2	1
ijk	1	1	1	0	0	0	1	0
klm	0	0	0	1	0	1	1	0
lmn	0	0	0	0	1	0	1	1
opq	0	0	0	1	0	1	1	0
pqr	3	2	1	1	1	1	4	1
xyz	1	1	0	0	1	0	1	1

a. coccurance matrix where diagnal elements are filled with zeroes

In [ ]:
```
# reviews dataset
data = x_train #reviews

matrix, vocab = create_co_occurance_matrix(data, name='without_
freq', window_size=2, vocab_size=2000)
data_matrix = pd.DataFrame(matrix, index = vocab_index,
                             columns = vocab_index)
data_matrix
```

In [94]:
```
# coccurance matrix  where diagnal elements are filled with zer
oes
# saving file using pickle
"""
file= open("datamatrix_with_freq.pkl","wb")
pickle.dump(data_matrix,file)
file.close()
```

```
file= open("vocab_index_with_freq.pkl","wb")
pickle.dump(vocab_index,file)
file.close()"""

file= open("vocab_index_with_freq.pkl","rb")
vocab=pickle.load(file)
file.close()

file= open("datamatrix_with_freq.pkl","rb")
data_matrix=pickle.load(file)
file.close()
```

In [76]:
```
data_matrix.shape
```

```
(2000, 2000)
```

In [77]:
```
data_matrix.head()
```

	able	absolute	absolutely	according	acid	acidic	acro
able	0	0	7	1	1	0	2
absolute	0	0	0	0	0	0	0
absolutely	7	0	0	2	2	0	2
according	1	0	2	0	2	0	1
acid	1	0	2	2	0	3	0

5 rows × 2000 columns

b. coccurance matrix where diagnal elements are filled with frequecy of that word

In [ ]:
```
# reviews dataset
data = x_train #reviews

matrix, vocab = create_co_occurance_matrix(data, name='with_fre
q', window_size=2, vocab_size=2000)
data_matrix = pd.DataFrame(matrix, index = vocab_index,
                                 columns = vocab_index)
data_matrix
```

In [93]:
```
# coccurance matrix  where diagnal elements are filled with fre
quecy of that word
# saving file using pickle

"""file= open("datamatrix.pkl","wb")
pickle.dump(data_matrix,file)
file.close()

file= open("vocab_index.pkl","wb")
pickle.dump(vocab_index,file)
file.close()"""

file= open("vocab_index.pkl","rb")
vocab_index=pickle.load(file)
file.close()

file= open("datamatrix.pkl","rb")
```

```
data_matrix=pickle.load(file)
file.close()
```

In [61]:
```
data_matrix.shape
```

```
(2000, 2000)
```

In [74]:
```
data_matrix.head()
```

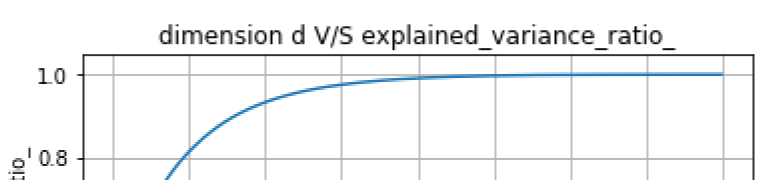
	able	absolute	absolutely	according	acid	acidic	acros
able	6817	0	7	1	1	0	2
absolute	0	798	0	0	0	0	0
absolutely	7	0	5959	2	2	0	2
according	1	0	2	1089	2	0	1
acid	1	0	2	2	2245	3	0

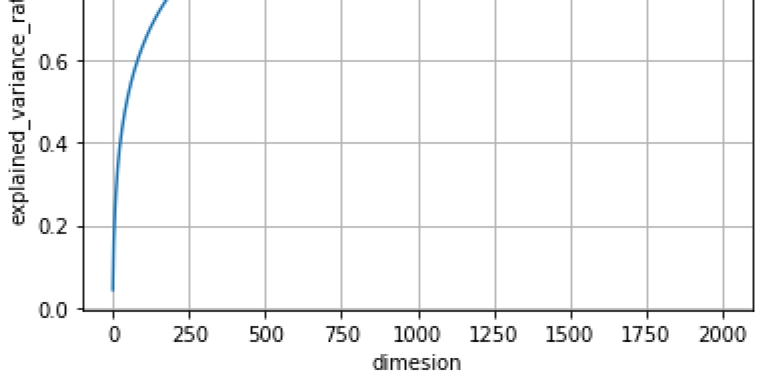
5 rows × 2000 columns

[5.3] Finding optimal value for number of components (n) to be retained.

a. coccurance matrix where diagnal elements are filled with zeroes

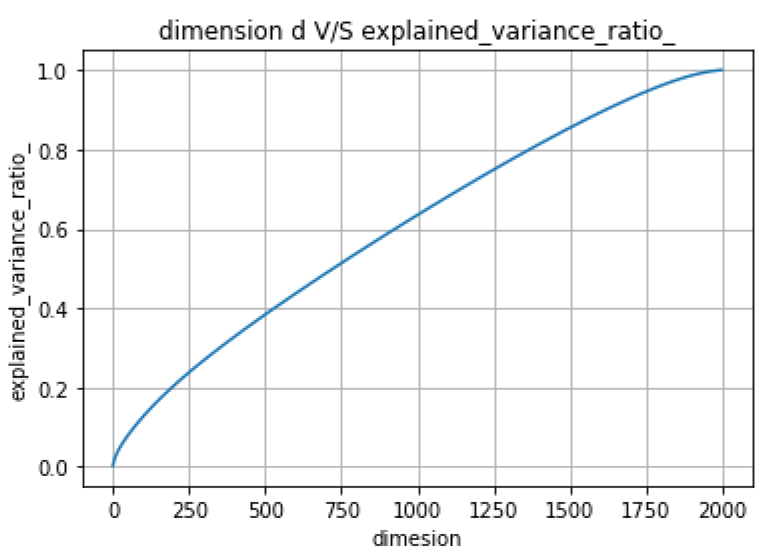
In [95]:
```
# normalise
from sklearn.preprocessing import normalize
data_matrix=normalize(data_matrix)

# applying SVD to find the optimum number of (n)
from sklearn.decomposition import TruncatedSVD
svd=TruncatedSVD(n_components=data_matrix.shape[1]-1)
svd.fit(data_matrix)

# filling diagnal matrix as zeros
plt.plot(range(data_matrix.shape[1]-1),np.cumsum(svd.explained_
variance_ratio_))
plt.title("dimension d V/S explained_variance_ratio_")
plt.xlabel("dimesion")
plt.ylabel("explained_variance_ratio_")
plt.grid()
plt.show()
```





b. coccurance matrix where diagnal elements are filled with frequecy of that word

In [497]:
```
# normalise
from sklearn.preprocessing import normalize
data_matrix=normalize(data_matrix)

# applying SVD to find the optimum number of (n)
from sklearn.decomposition import TruncatedSVD
svd=TruncatedSVD(n_components=data_matrix.shape[1]-1)
svd.fit(data_matrix)

# filling diagnal matrix as frequency of that word occur
plt.plot(range(data_matrix.shape[1]-1),np.cumsum(svd.explained_
variance_ratio_))
plt.title("dimension d V/S explained_variance_ratio_")
plt.xlabel("dimesion")
plt.ylabel("explained_variance_ratio_")
plt.grid()
plt.show()
```



Observation:

1. co_occurance matrix where diagnal matrix filled with zeroes is working better

1. Optimum n = 500

1. Able to explain 90 % variance

In [96]:
```
# applying SVD tusing optimum d=500
from sklearn.decomposition import TruncatedSVD
svd=TruncatedSVD(n_components=500)
svd_w2v=svd.fit_transform(data_matrix)
```

In [97]:
```
# creating dataframe of SVD_W2V
df={}
for idx,word in enumerate(list(vocab_index.keys())):
                           df.update({word:svd_w2v[idx]})

w2v_df=pd.DataFrame(df)
w2v_df=w2v_df.T
w2v_df.head()
```

	0	1	2	3	4	5
able	0.744141	-0.298105	0.022145	0.093912	-0.052519	0.024516
absolute	0.299363	0.136898	-0.061632	0.062936	-0.005146	0.117938
absolutely	0.506525	0.151063	-0.036939	0.000125	-0.110371	0.162425
according	0.518709	0.052886	-0.095109	0.039503	0.017190	-0.123320
acid	0.517402	0.130101	0.078904	0.146178	0.114756	-0.086926

5 rows × 500 columns

In [98]:
```
# sample
w2v_df.iloc[0]
```

```
0      0.744141
1     -0.298105
2      0.022145
3      0.093912
4     -0.052519
5      0.024516
6     -0.154012
7      0.062500
8     -0.154156
9     -0.132359
10    -0.175485
11     0.186537
12     0.060593
13     0.057165
14     0.087214
15    -0.035552
16     0.024535
17     0.185509
18     0.041823
19    -0.114937
20    -0.063319
21     0.084575
22     0.039600
23     0.061579
24     0.033749
25    -0.062283
```

```
26    -0.081115
27    -0.098348
28    -0.021580
29    -0.017758
         ...
470   -0.000582
471   -0.008447
472   -0.002808
473   -0.002483
474   -0.010753
475   -0.000087
476    0.004993
477   -0.006005
478   -0.009440
479   -0.006636
480    0.000764
481   -0.000130
482   -0.005599
483    0.003330
484    0.006670
485    0.003984
486   -0.008456
487    0.006128
488    0.004211
489    0.005353
490   -0.004872
491    0.001260
492    0.004483
493    0.005530
494   -0.008062
495    0.006279
496    0.005016
497   -0.006768
498    0.004612
499   -0.000281
Name: able, Length: 500, dtype: float64
```

[5.4] Applying k-means clustering

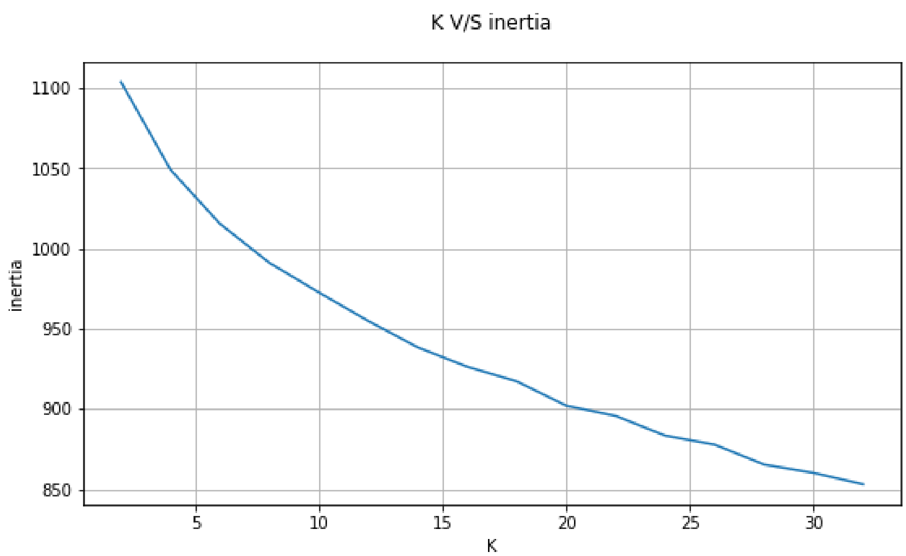
In [99]:

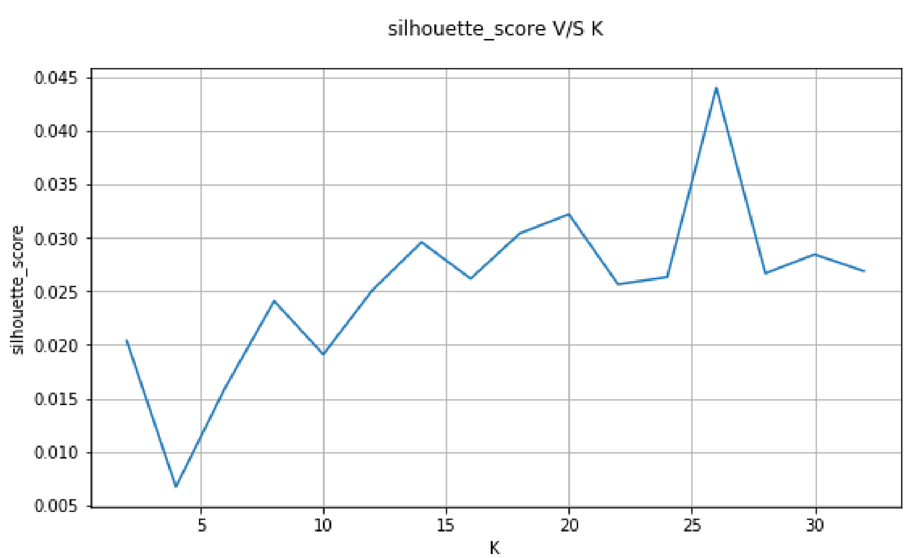
```
from sklearn.cluster import k_means
from sklearn.metrics import silhouette_score
k=[2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32]
inertia=[]
ss=[]

for i in k:
    km=KMeans(n_clusters=i)
    km.fit(w2v_df)
    inertia.append(km.inertia_)
    ss.append(silhouette_score(w2v_df,km.predict(w2v_df)))
```

In [100]:

```
# ploting K(n_cluster) V/S score
plt.figure(figsize=(9,5))
plt.plot(k,inertia)
```

```
plt.title("K V/S inertia\n")
plt.xlabel("K")
plt.ylabel("inertia")
plt.grid()
plt.show()

plt.figure(figsize=(9,5))
plt.plot(k,ss)
plt.title("silhouette_score V/S K\n")
plt.xlabel("K")
plt.ylabel("silhouette_score")
plt.grid()
plt.show()
```





Optimal number of cluser K = 4

```
In [101]: optimal_k=4
km=KMeans(n_clusters=optimal_k)
km.fit(w2v_df)
print("inertia: ",km.inertia_)
print("silhouette_score: ",silhouette_score(w2v_df,km.predict(w
2v_df)))
```

```
inertia:  1049.1951338386477
```

```
silhouette_score:  0.006492483143397871
```

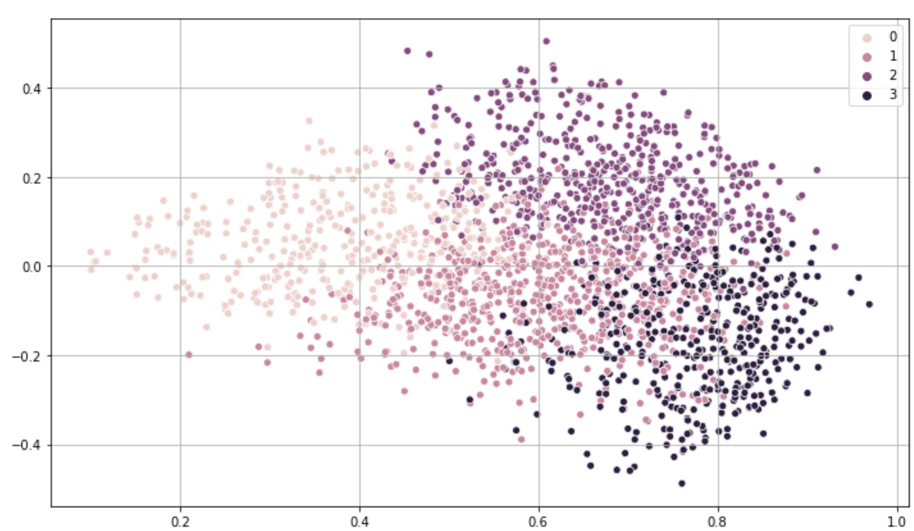
Observation: Not able to clearly seperate clusters

[5.5] Wordclouds of clusters obtained in the above section

In [102]:
```python
from sklearn.metrics import pairwise_distances_argmin_min
closest , _ = pairwise_distances_argmin_min(km.cluster_centers_
,w2v_df)
```

In [119]:
```python
# visualtion of w2v vectors using SVD
from sklearn.decomposition import TruncatedSVD
svd=TruncatedSVD(n_components=2)
viz_w2v=svd.fit_transform(data_matrix)

plt.figure(figsize=(12,7))
sns.scatterplot(viz_w2v[:,0],viz_w2v[:,1],hue=km.predict(w2v_df
),)
plt.title("Visualisation of svd w2v\n")
plt.grid()
plt.show()
```

Visualisation of svd w2v

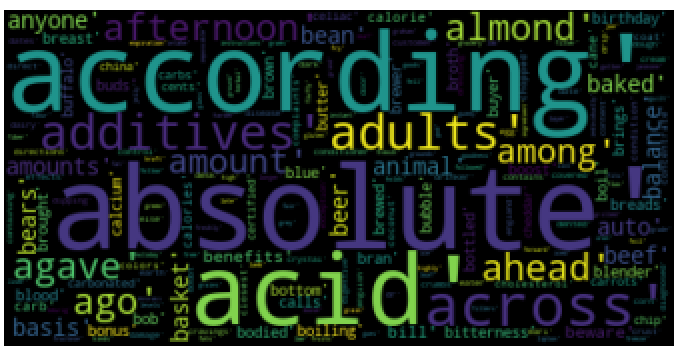


In [120]:
```python
# wordcloud of centroids
closest_words=(pd.DataFrame(vocab.keys()).iloc[closest])
text=[]
for i in range(closest_words.shape[0]):
    text.append(closest_words.iloc[i,0])

# Create and generate a word cloud image:
wordcloud = WordCloud().generate(str(text))

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title("wordcloud of k_mean centroids\n")
plt.show()
```

wordcloud of k_mean centroids

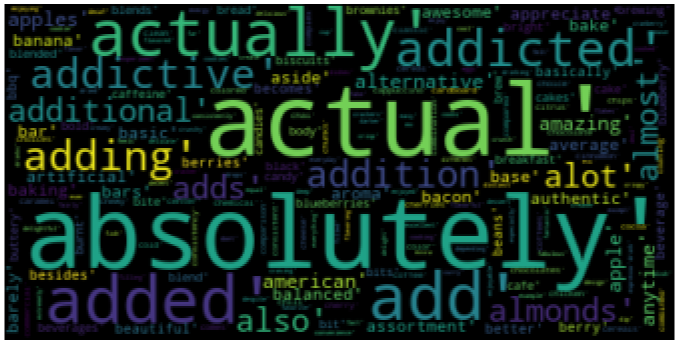

```
In [112]: # wordcloud of each cluster

for i in range(optimal_k):
    words_of_cluster=(pd.DataFrame(vocab.keys()).iloc[np.argwhere(km.predict(w2v_df)==i).ravel()])

    text=[]
    for j in range(words_of_cluster.shape[0]):
        text.append(words_of_cluster.iloc[j,0])

    # Create and generate a word cloud image:
    wordcloud = WordCloud().generate(str(text))

    # Display the generated image:
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(f"words from cluster: {i}\n")
    plt.axis("off")
    plt.show()
```

words from cluster: 0

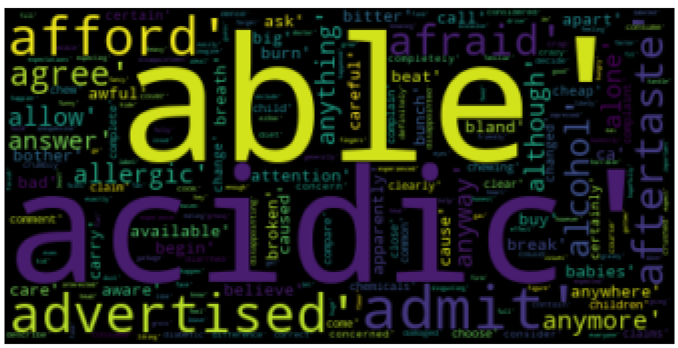


words from cluster: 1

words from cluster: 2



words from cluster: 3



[5.6] Function that returns most similar words for a given word.

In [113]:
```
from sklearn.metrics import pairwise_distances
def similarity(word,n=5):

    word=str(word)

    if word in vocab:
        p=pairwise_distances([w2v_df.loc[word]],w2v_df,metric=
'cosine') # calculating the distance of word vector to rest of
 the w2v vectors
        n_similar=p.argsort()[0][1:n+1] # finding n_closest wrd
        return w2v_df.iloc[n_similar] # n_similar words

    else:
        print("word is not present in vocabulary")
```

In [114]:
```
# example 1
```

```python
similarity("almost",10)
```

	0	1	2	3	4	5	
basically	0.853950	0.159282	-0.006441	-0.092219	-0.065754	-0.022528	0
kind	0.894267	0.158996	0.147922	0.008307	-0.049771	0.159083	-
actually	0.891602	0.154813	0.065859	0.008054	-0.098400	0.102978	-
literally	0.781333	0.069574	-0.062180	-0.144341	0.040860	0.158244	0
seems	0.756662	0.270937	-0.067953	-0.081032	-0.159911	0.146147	0
sort	0.768540	0.244507	0.218992	-0.018605	-0.100423	0.149002	0
kinda	0.713585	0.233898	0.202838	-0.021000	-0.133419	0.160861	-
seemed	0.715950	0.248628	-0.033905	-0.128841	-0.144524	0.183550	0
also	0.911041	0.215778	-0.059293	-0.060050	-0.163177	0.006870	-
not	0.718105	0.341021	-0.222394	0.053670	-0.127113	0.114929	0

10 rows × 500 columns

```python
In [115]: # example 2
similarity("tasty",10)
```

	0	1	2	3	4	5
yummy	0.869070	0.189862	0.065370	-0.006997	-0.121762	-0.00491
delicious	0.860767	0.224124	0.055305	0.114304	-0.000756	-0.01205
filling	0.793443	0.135866	0.141058	-0.104959	-0.154932	-0.07975
inexpensive	0.797804	-0.046553	-0.024696	0.070289	-0.082445	-0.03428
however	0.969420	-0.085649	0.091409	0.092301	-0.049794	0.006271
although	0.925173	-0.139675	0.259805	0.053356	-0.021345	-0.00149
still	0.931067	0.043804	0.042502	0.021561	-0.085798	0.014625
anyway	0.949028	-0.059309	0.023779	-0.018522	-0.041435	0.096704
course	0.957573	-0.025825	0.058782	0.031425	-0.089506	-0.03333
satisfying	0.752773	0.240287	0.151531	0.080567	-0.088734	-0.05639

10 rows × 500 columns

```python
In [116]: # example 3
print(list(similarity("good",10).index))
```

```
['like', 'ok', 'amazing', 'terrible', 'impressed', 'howe
ver', 'okay', 'awesome', 'disappointing', 'fantastic']
```

```python
In [117]: # example 5
print(list(similarity("bad",10).index))
```

```
['know', 'realize', 'mention', 'understand', 'disappoint
ed', 'obviously', 'care', 'bother', 'sorry', 'whether']
```

```python
In [118]: # example 5
similarity("ugly",10)
```

```
word is not present in vocabulary
```

[6] Conclusions

1. co_occurance matrix where diagnal matrix filled with zeroes is working better

1. We have used maximum dimension(d) = 500 as it able to explain 90% of variance of dataset.

1. Word2Vec obtained by SVD is not as powerfull as gensim w2v or other state of art w2v representaion.

1. optimal number of cluster obtained is k = 4

1. able to seperate clusters (refer: 5.4)