finalclusteringassignment

March 14, 2019

1 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.

2 [1]. Reading Data

2.1 [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 [0]: %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 [0]: # Code to read csv file into Colaboratory:
        !pip install -U -q PyDrive
        from pydrive.auth import GoogleAuth
        from pydrive.drive import GoogleDrive
        from google.colab import auth
        from oauth2client.client import GoogleCredentials
        # Authenticate and create the PyDrive client.
        auth.authenticate_user()
        gauth = GoogleAuth()
```

```
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
    100% || 993kB 19.3MB/s
  Building wheel for PyDrive (setup.py) ... done
```

In [0]:
```
link = 'https://drive.google.com/open?id=1cpwGHmONMCohLX-EQu9ubkB58ZoVc9pI' # The shar
```

In [0]:
```
fluff, id = link.split('=')
print (id) # Verify that you have everything after '='
```

```
1cpwGHmONMCohLX-EQu9ubkB58ZoVc9pI
```

In [0]:
```
import pandas as pd
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('opendata.csv')
df3 = pd.read_csv('opendata.csv')
```

In [0]:
```
filtered_data=df3
```

In [0]:
```
# using SQLite Table to read data.
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 point
# 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 5
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negativ
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 (5000, 10)
```

```
Out[0]:    Id   ProductId          UserId                      ProfileName  \
        0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian
        1   2  B00813GRG4  A1D87F6ZCVE5NK                           dll pa
        2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"

           HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
        0                     1                       1      1  1303862400
        1                     0                       0      0  1346976000
        2                     1                       1      1  1219017600

                         Summary                                              Text
        0  Good Quality Dog Food  I have bought several of the Vitality canned d...
        1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
        2  "Delight" says it all  This is a confection that has been around a fe...
```

```
In [0]: display = pd.read_sql_query("""
        SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
        FROM Reviews
        GROUP BY UserId
        HAVING COUNT(*)>1
        """, con)
```

```
In [0]: print(display.shape)
        display.head()
```

```
(80668, 7)
```

```
Out[0]:               UserId   ProductId             ProfileName        Time  Score  \
        0  #oc-R115TNMSPFT9I7  B007Y59HVM                 Breyton  1331510400      2
        1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
        2  #oc-R11DNU2NBKQ23Z  B007Y59HVM        Kim Cieszykowski  1348531200      1
        3  #oc-R11O5J5ZVQE25C  B005HG9ET0           Penguin Chick  1346889600      5
        4  #oc-R12KPBODL2B5ZD  B007OSBE1U   Christopher P. Presta  1348617600      1

                                                        Text  COUNT(*)
        0  Overall its just OK when considering the price...         2
        1  My wife has recurring extreme muscle spasms, u...         3
        2  This coffee is horrible and unfortunately not ...         2
        3  This will be the bottle that you grab from the...         3
        4  I didnt like this coffee. Instead of telling y...         2
```

```
In [0]: display[display['UserId']=='AZY10LLTJ71NX']
```

```
Out[0]:               UserId   ProductId                     ProfileName        Time  \
        80638  AZY10LLTJ71NX  B006P7E5ZI  undertheshrine "undertheshrine"  1334707200

               Score                                               Text  COUNT(*)
        80638      5  I was recommended to try green tea extract to ...         5
```

```
In [0]: display['COUNT(*)'].sum()

Out[0]: 393063
```

3 [2] Exploratory Data Analysis

3.1 [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 [0]: display= pd.read_sql_query("""
        SELECT *
        FROM Reviews
        WHERE Score != 3 AND UserId="AR5J8UI46CURR"
        ORDER BY ProductID
        """, con)
        display.head()

Out[0]:        Id   ProductId         UserId      ProfileName  HelpfulnessNumerator  \
        0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2
        1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2
        2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2
        3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                     2
        4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                     2

           HelpfulnessDenominator  Score        Time  \
        0                       2      5  1199577600
        1                       2      5  1199577600
        2                       2      5  1199577600
        3                       2      5  1199577600
        4                       2      5  1199577600

                                     Summary  \
        0  LOACKER QUADRATINI VANILLA WAFERS
        1  LOACKER QUADRATINI VANILLA WAFERS
        2  LOACKER QUADRATINI VANILLA WAFERS
        3  LOACKER QUADRATINI VANILLA WAFERS
        4  LOACKER QUADRATINI VANILLA WAFERS

                                                        Text
        0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8) ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

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

```
In [0]: #Deduplication of entries
        final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
        final.shape
```

```
Out[0]: (46072, 11)
```

```
In [0]: #Checking to see how much % of data still remains
        (final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
Out[0]: 92.144
```

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

        display.head()
```

```
In [0]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
In [0]: #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()
```

```
(46072, 11)
```

```
Out[0]: 5    31266
        4     7214
        1     4774
        2     2818
        Name: Score, dtype: int64
```

4 [3] Preprocessing

4.1 [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 [0]: # printing some random reviews
        sent_0 = final['Text'].values[0]
        print(sent_0)
        print("="*50)

        sent_1000 = final['Text'].values[1000]
        print(sent_1000)
        print("="*50)

        sent_1500 = final['Text'].values[1500]
        print(sent_1500)
        print("="*50)

        sent_4900 = final['Text'].values[4900]
        print(sent_4900)
        print("="*50)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cobbler. home made is
==================================================
```

```
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are
==================================================
For those of you wanting a high-quality, yet affordable green tea, you should definitely give t
==================================================
```

```
In [0]: # remove urls from text python: https://stackoverflow.com/a/40823105/4084039
        sent_0 = re.sub(r"http\S+", "", sent_0)
        sent_1000 = re.sub(r"http\S+", "", sent_1000)
        sent_150 = re.sub(r"http\S+", "", sent_1500)
        sent_4900 = re.sub(r"http\S+", "", sent_4900)

        print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
```

```
In [0]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-
        from bs4 import BeautifulSoup

        soup = BeautifulSoup(sent_0, 'lxml')
        text = soup.get_text()
        print(text)
        print("="*50)

        soup = BeautifulSoup(sent_1000, 'lxml')
        text = soup.get_text()
        print(text)
        print("="*50)

        soup = BeautifulSoup(sent_1500, 'lxml')
        text = soup.get_text()
        print(text)
        print("="*50)

        soup = BeautifulSoup(sent_4900, 'lxml')
        text = soup.get_text()
        print(text)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cobbler. home made is
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are
==================================================
For those of you wanting a high-quality, yet affordable green tea, you should definitely give t
```

```
In [0]: # 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 [0]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are
==================================================
```

In [0]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
```

In [0]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Great flavor low in calories high in nutrients high in protein Usually protein powders are hig
```

In [0]:
```
# 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 step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselve
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', '
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "t
```

```
                'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
                'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
                'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
                'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'o
                'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
                'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
                's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'n
                've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't"
                "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mig
                "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", '
                'won', "won't", 'wouldn', "wouldn't"])
```

In [0]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    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 stopwo
    preprocessed_reviews.append(sentance.strip())
```

```
100%|| 46072/46072 [00:18<00:00, 2533.96it/s]
```

In [0]:
```
preprocessed_reviews[1500]
```

Out[0]:
```
'great flavor low calories high nutrients high protein usually protein powders high pr
```

[3.2] Preprocessing Review Summary

In [0]:
```
def func(x):
  if x>3:
    return 1
  else:
    return 0
```

In [0]:
```
x=preprocessed_reviews
y=final['Score'].apply(func)
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

In [0]:
```
from sklearn.feature_extraction.text import CountVectorizer
count_vect=CountVectorizer()
xtrainonehotencoding=count_vect.fit_transform(x)
```

```
In [0]: print(xtrainonehotencoding.shape)
```

```
(46072, 39365)
```

5.2 [4.2] Bi-Grams and n-Grams.

```
In [0]: #bi-gram, tri-gram and n-gram

        #removing stop words like "not" should be avoided before building n-grams
        # count_vect = CountVectorizer(ngram_range=(1,2))
        # please do read the CountVectorizer documentation http://scikit-learn.org/stable/modu

        # you can choose these numebrs min_df=10, max_features=5000, of your choice
        count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
        final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
        print("the type of count vectorizer ",type(final_bigram_counts))
        print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
        print("the number of unique words including both unigrams and bigrams ", final_bigram_
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (46072, 5000)
the number of unique words including both unigrams and bigrams  5000
```

5.3 [4.3] TF-IDF

```
In [0]: from sklearn.feature_extraction.text import TfidfVectorizer
        tfidf= TfidfVectorizer()
        xtraintfidfencoding=tfidf.fit_transform(x)

        print(xtraintfidfencoding.shape)
```

```
(46072, 39365)
```

5.4 [4.4] Word2Vec

```
In [0]: # Train your own Word2Vec model using your own text corpus
        i=0
        list_of_sentance=[]
        for sentance in preprocessed_reviews:
            list_of_sentance.append(sentance.split())
```

```
In [0]: # Using Google News Word2Vectors

        # in this project we are using a pretrained model by google
        # its 3.3G file, once you load this into your memory
        # it occupies ~9Gb, so please do this step only if you have >12G of ram
```

```
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=True
want_to_use_google_w2v =True
want_to_train_w2v = False

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bi
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to
```

```
you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w2v
```

In [0]:
```
w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)

print(w2v_model.wv.most_similar('great'))
print('='*50)
print(w2v_model.wv.most_similar('worst'))
```

```
WARNING:gensim.models.base_any2vec:consider setting layer size to a multiple of 4 for greater p


[('awesome', 0.8385358452796936), ('fantastic', 0.810681164264679), ('terrific', 0.807738482952
==================================================
[('greatest', 0.7555773258209229), ('nastiest', 0.7460689544677734), ('best', 0.71960687637329
```

In [0]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  12798
sample words  ['dogs', 'loves', 'chicken', 'product', 'china', 'wont', 'buying', 'anymore', 'ha
```

5.5 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

In [0]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|| 46071/46071 [01:31<00:00, 502.32it/s]

46071
50
```

In [0]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectorsspecial = []; # the avg-w2v for each sentence/review is stored in this lis
listofsentance=pd.DataFrame(list_of_sentance)
list_of_sentance1 = listofsentance.sample(frac=0.2)
list_of_sentance12=list_of_sentance1.values
for sent in tqdm(list_of_sentance12): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
```

```
            sent_vec /= cnt_words
        sent_vectorsspecial.append(sent_vec)
    print(len(sent_vectorsspecial))
    print(len(sent_vectorsspecial[0]))
```

```
100%|| 9214/9214 [2:16:01<00:00,  1.08it/s]

9214
50
```

[4.4.1.2] TFIDF weighted W2v

```
In [0]: model = TfidfVectorizer()
        xtraintfidfw2v = model.fit_transform(preprocessed_reviews)
        #xtesttfidfw2v=model.transform(xtest)
        #xcvtfidfw2v=model.transform(xcv)
        tfidf_feat = model.get_feature_names()
        dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [0]: # TF-IDF weighted Word2Vec
        tfidf_feat = model.get_feature_names() # tfidf words/col-names
        # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

        tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this li
        row=0;
        for sent in tqdm(list_of_sentance): # for each review/sentence
            sent_vec = np.zeros(50) # as word vectors are of zero length
            weight_sum =0; # num of words with a valid vector in the sentence/review
            for word in sent: # for each word in a review/sentence
                if word in w2v_words and word in tfidf_feat:
                    vec = w2v_model.wv[word]
        #             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                    # to reduce the computation we are
                    # dictionary[word] = idf value of word in whole courpus
                    # sent.count(word) = tf valeus of word in this review
                    tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                    sent_vec += (vec * tf_idf)
                    weight_sum += tf_idf
            if weight_sum != 0:
                sent_vec /= weight_sum
            tfidf_sent_vectors.append(sent_vec)
            row += 1
```

```
100%|| 46071/46071 [43:25<00:00, 17.68it/s]
```

```
In [ ]: # TF-IDF weighted Word2Vec
        tfidf_feat = model.get_feature_names() # tfidf words/col-names
        # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

        tfidf_sent_vectorsspecial= []; # the tfidf-w2v for each sentence/review is stored in th
        listofsentance=pd.DataFrame(list_of_sentance)
        list_of_sentance1a = listofsentance.sample(frac=0.2)
        list_of_sentance12a=list_of_sentance1a.values
        row=0;
        for sent in tqdm(list_of_sentance12a): # for each review/sentence
            sent_vec1 = np.zeros(50) # as word vectors are of zero length
            weight_sum =0; # num of words with a valid vector in the sentence/review
            for word in sent: # for each word in a review/sentence
                if word in w2v_words and word in tfidf_feat:
                    vec = w2v_model.wv[word]
        #             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                    # to reduce the computation we are
                    # dictionary[word] = idf value of word in whole courpus
                    # sent.count(word) = tf valeus of word in this review
                    tf_idf = dictionary[word]
                    sent_vec1 += (vec * tf_idf)
                    weight_sum += tf_idf
            if weight_sum != 0:
                sent_vec1 /= weight_sum
            tfidf_sent_vectorsspecial.append(sent_vec1)
            row += 1
```

6 [5] Assignment 10: K-Means, Agglomerative & DBSCAN Clustering

```
<li><strong>Apply K-means Clustering on these feature sets:</strong>
    <ul>
<li><font color='red'>SET 1:</font>Review text, preprocessed one converted into vectors using
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vectors
<li>Find the best k using the elbow-knee method (plot k vs inertia_)</li>
<li>Once after you find the k clusters, plot the word cloud per each cluster so that at a singl
```

go we can analyze the words in a cluster.

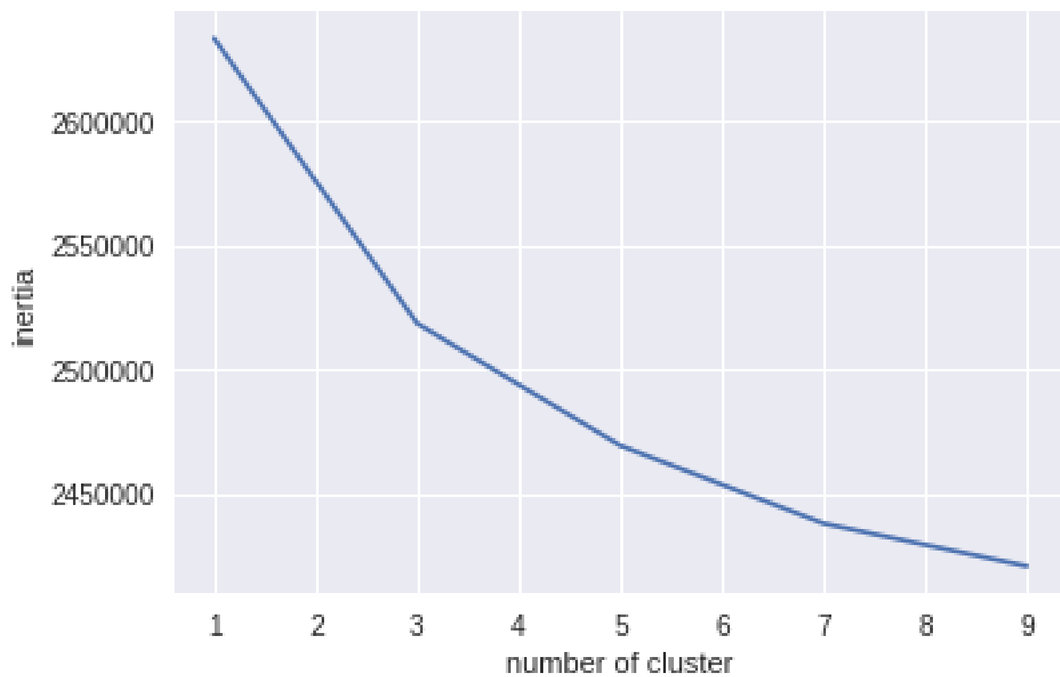
```
    </ul>
</li>
<br>
<li><strong>Apply Agglomerative Clustering on these feature sets:</strong>
    <ul>
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vectors
<li>Apply agglomerative algorithm and try a different number of clusters like 2,5 etc.</li>
```

```
<li>Same as that of K-means, plot word clouds for each cluster and summarize in your own words
        <li>You can take around 5000 reviews or so(as this is very computationally expensive or
    </ul>
</li>
<br>
<br>
<li><strong>Apply DBSCAN Clustering on these feature sets:</strong>
    <ul>
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vectors
<li>Find the best Eps using the <a href='https://stackoverflow.com/questions/12893492/choosing-
<li>Same as before, plot word clouds for each cluster and summarize in your own words what that
        <li>You can take around 5000 reviews for this as well.</li>
    </ul>
</li>
```

6.1 [5.1] K-Means Clustering

6.1.1 [5.1.1] Applying K-Means Clustering on BOW, SET 1

In [0]:
```python
from sklearn.cluster import KMeans
dic={}
for i in range(1,10,2):

  kmeans=KMeans(n_clusters=i)
  kmeans.fit(xtrainonehotencoding)
  dic[i]=kmeans.inertia_
fig,ax=plt.subplots()
ax.plot(list(dic.keys()),list(dic.values()))
plt.xlabel('number of cluster')
plt.ylabel('inertia')
plt.show()
```



6.1.2 [5.1.2] Wordclouds of clusters obtained after applying k-means on BOW SET 1

```
In [0]: from sklearn.cluster import KMeans
        kmeans=KMeans(n_clusters=5)
        kmeans.fit(xtrainonehotencoding)
        labels=kmeans.predict(xtrainonehotencoding)
```

```
In [0]: wordsofcluster1=[]
        wordsofcluster2=[]
        wordsofcluster3=[]
        wordsofcluster4=[]
        wordsofcluster5=[]
```

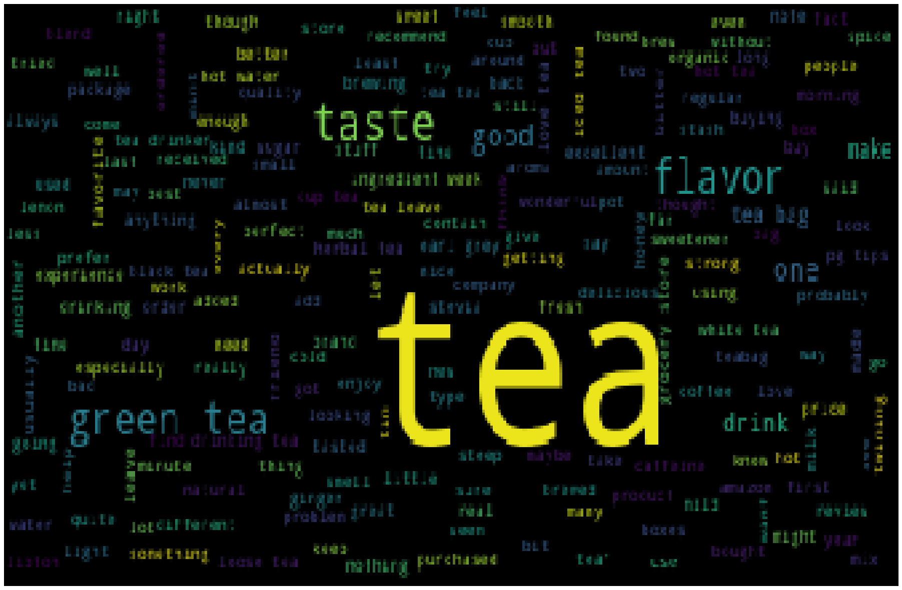
```
In [0]: for i in range(len(labels)):
          if labels[i]==0:
            wordsofcluster1.append(x[i])
          elif labels[i]==1:
            wordsofcluster2.append(x[i])
          elif labels[i]==2:
            wordsofcluster3.append(x[i])
          elif labels[i]==3:
            wordsofcluster4.append(x[i])
          else:
            wordsofcluster5.append(x[i])
```

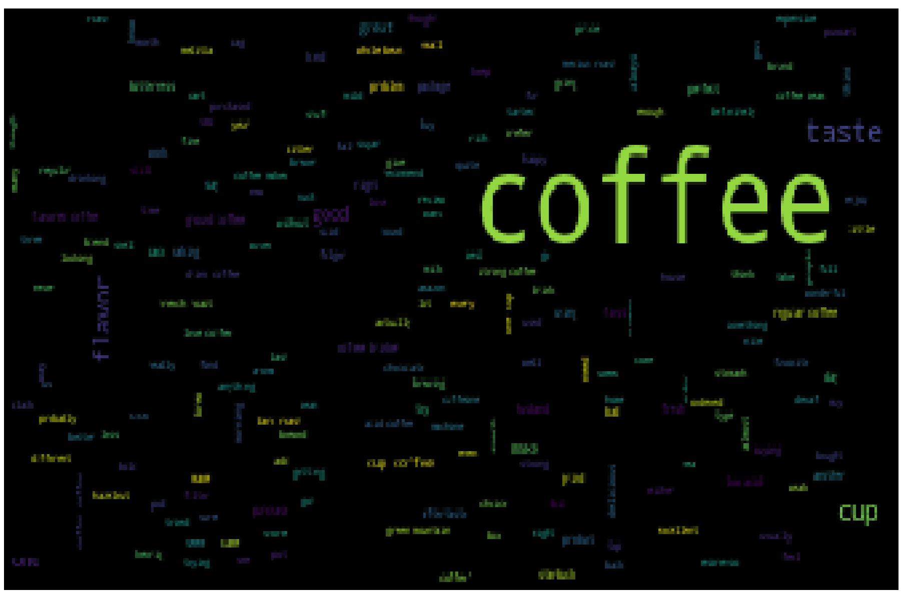
In [0]:

```
from wordcloud import WordCloud
from matplotlib.pyplot import figure
import matplotlib.pyplot as plt1
word_cloud = WordCloud(relative_scaling = 1.0).generate(str(wordsofcluster1))
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
word_cloud = WordCloud(relative_scaling = 1.0).generate(str(wordsofcluster2))
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
word_cloud = WordCloud(relative_scaling = 1.0).generate(str(wordsofcluster3))
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()

word_cloud = WordCloud(relative_scaling = 1.0).generate(str(wordsofcluster4))
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()

word_cloud = WordCloud(relative_scaling = 1.0).generate(str(wordsofcluster5))
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```


love
one
taste
flavor
good
product
great
use
make
really
much
treat
tried
used
delicious
tasty
still
well
need
mix
got
first
brand
price
drink
little
found
best
try
made
dog
tea
food
buy
box
perfect
chocolate
amazon
think
bought
nice
better
cup
snack
coffee
lot
favorite
time
never
keep
order
using
give
work
eat
year
wonderful
ordered
even
cookie
fresh
enjoy
bag
day
alway
cat
say
come
want
thing
know
right
without
bit
crap
excellent
eating
item
package
pack


one
food
product
taste
flavor
treat
really
tried
first
even
know
dog
good
great
buy
bag
think
cat
love
drink
time
chocolate
well
much
make
use
used
eat
brand
ingredient
little
review
water
made
amazon
still
thing
price
cookie
need
actually
though
take
natural
without
stuff
mix
sweet
many
eating
another
work
seem
say
bought
got
lot
people
found
day
give
see
put
texture
best
find
right
want
better
nice
way
bad
always
sugar
organic
going
box
come
sure
less
look
keep
bit
try
probably
something
never
small
enough
two
healthy
cup
add
diet
chicken
go
kind
may
anything

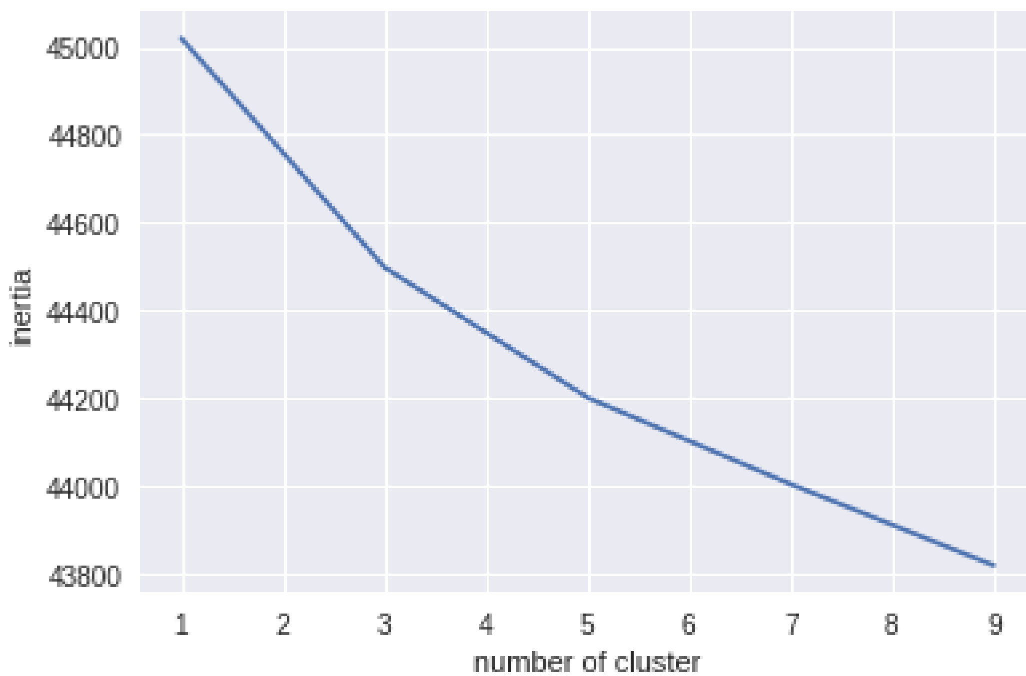




different clusters are representing different aspects of the reviews .firsrt cluster represents tate .and also clusters represents flavors of different things even some similar words of each cluster present in another .third cluster represent animals like dog and cat andthe things related to them .foutrh cluster reperesents tea fgreeen tea flavor and taste ..fourth cluster reperesent scoffee and stuff

6.1.3 [5.1.3] Applying K-Means Clustering on TFIDF, SET 2

In [0]:
```
from sklearn.cluster import KMeans
dic={}
for i in range(1,10,2):
  kmeans=KMeans(n_clusters=i)
  kmeans.fit(xtraintfidfencoding)
  dic[i]=kmeans.inertia_
fig,ax=plt.subplots()
ax.plot(list(dic.keys()),list(dic.values()))
plt.xlabel('number of cluster')
plt.ylabel('inertia')
plt.show()
```



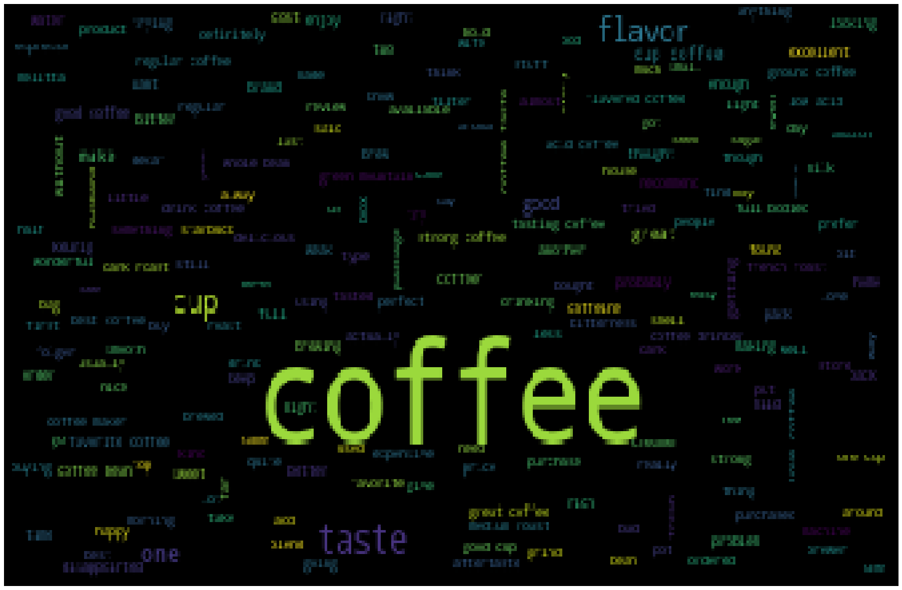
6.1.4 [5.1.4] Wordclouds of clusters obtained after applying k-means on TFIDF SET 2

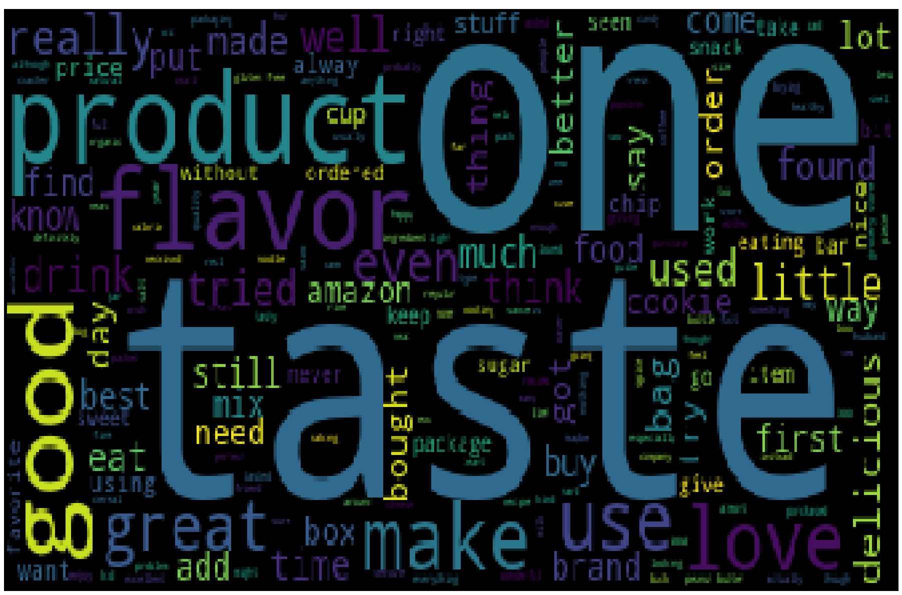
In [0]:
```
kmeans=KMeans(n_clusters=5)
kmeans.fit(xtraintfidfencoding)
labels=kmeans.predict(xtraintfidfencoding)
```

In [0]:
```
wordsofcluster1=[]
wordsofcluster2=[]
wordsofcluster3=[]
wordsofcluster4=[]
wordsofcluster5=[]
```

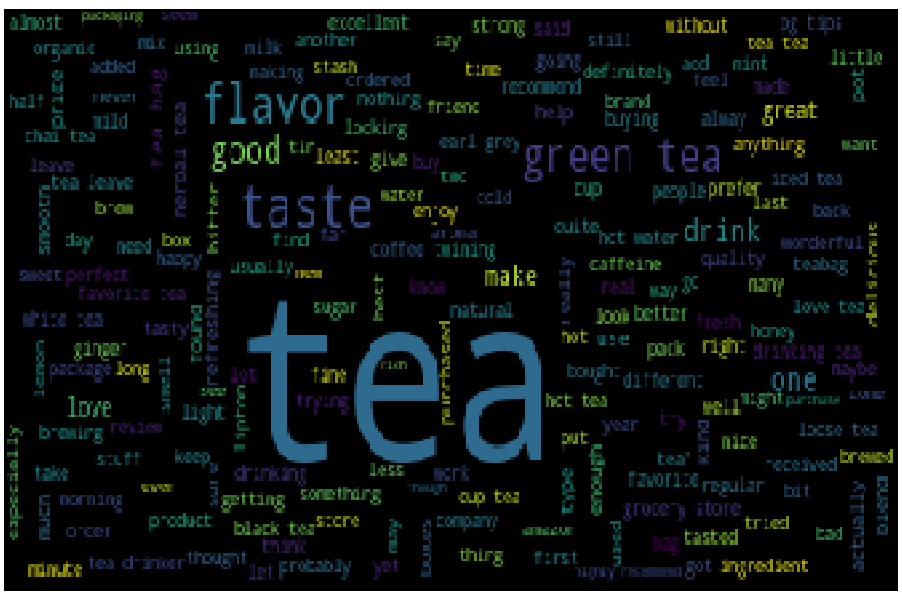
In [0]:
```
for i in range(len(labels)):
    if labels[i]==0:
      wordsofcluster1.append(x[i])
    elif labels[i]==1:
      wordsofcluster2.append(x[i])
    elif labels[i]==2:
      wordsofcluster3.append(x[i])
    elif labels[i]==3:
      wordsofcluster4.append(x[i])
    else:
      wordsofcluster5.append(x[i])
```

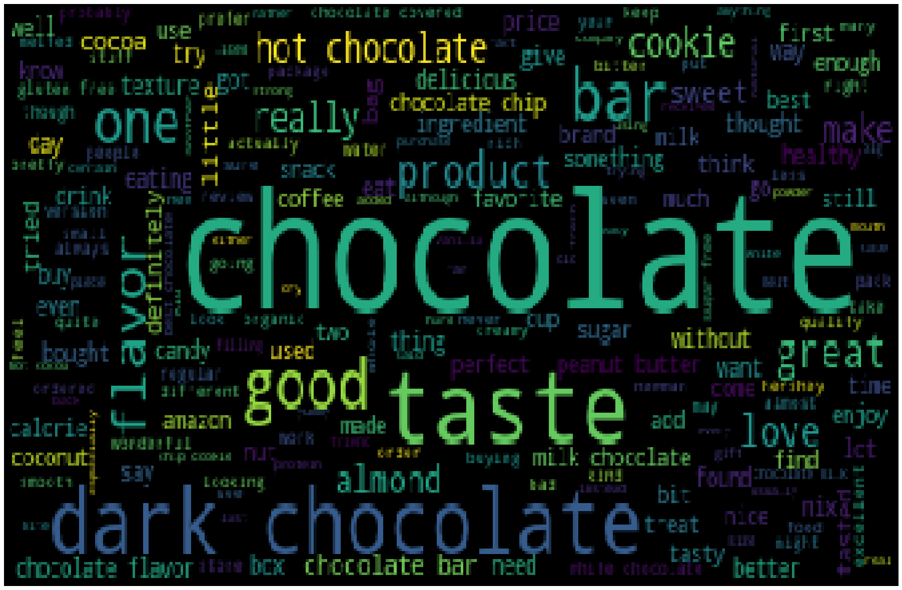
In [0]:
```
from wordcloud import WordCloud
from matplotlib.pyplot import figure
import matplotlib.pyplot as plt1
word_cloud = WordCloud(relative_scaling = 1.0).generate(str(wordsofcluster1))
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
word_cloud = WordCloud(relative_scaling = 1.0).generate(str(wordsofcluster2))
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
word_cloud = WordCloud(relative_scaling = 1.0).generate(str(wordsofcluster3))
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()

word_cloud = WordCloud(relative_scaling = 1.0).generate(str(wordsofcluster4))
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()

word_cloud = WordCloud(relative_scaling = 1.0).generate(str(wordsofcluster5))
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```


coffee
flavor
taste
cup
one


one
taste
product
flavor
good
really
made
well
love
make
use
great
delicious
found
little
better
even
much
food
used
tried
amazon
think
drink
find
know
cookie
way
lot
come
stuff
best
still
mix
need
eat
box
time
add
brand
bought
buy
bag
first
order
thing
say
chip


food
dog
treat
cat
one
product
dog food
love
eat
good
little

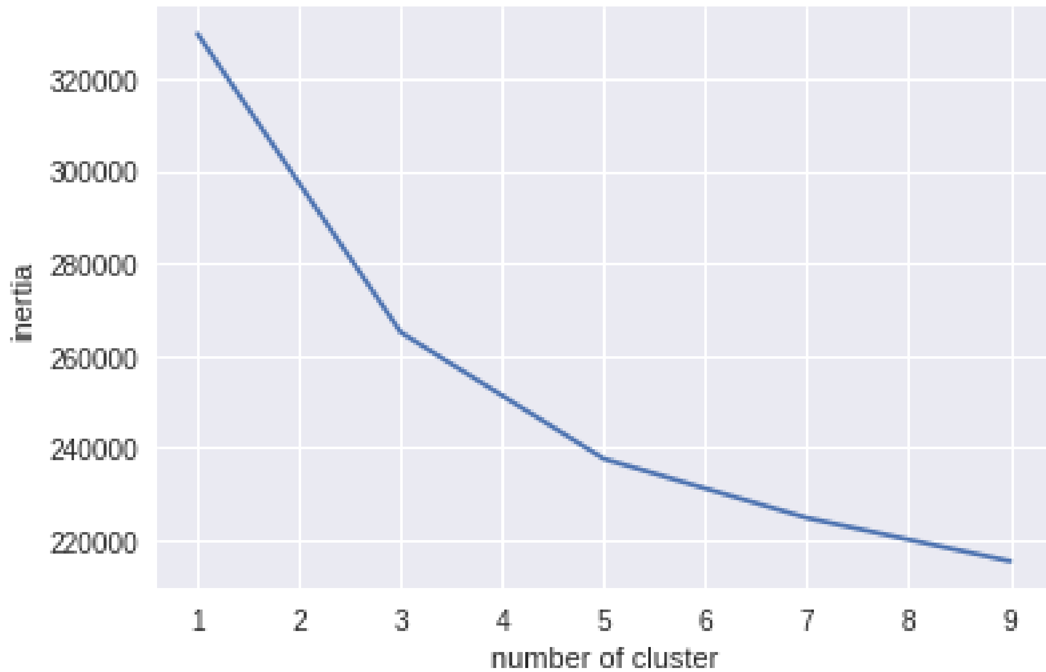

tea
taste
flavor
green tea
good
drink
one



- .cluster5 clearly represents choclate dark choclate and cookie and bar
- cluster 4 clearly represents about tea green tea flavor and taste
- .cluster2 clearly represents dog and dog feed dog food nad cats
- .cluster 3 also represent s about cat and dog trat and food
- .cluster1 also represenrts flavar and taste and good

6.1.5 [5.1.5] Applying K-Means Clustering on AVG W2V, SET 3

```
In [0]: from sklearn.cluster import KMeans
        dic={}
        for i in range(1,10,2):
          kmeans=KMeans(n_clusters=i)
          kmeans.fit(sent_vectors)
          dic[i]=kmeans.inertia_
        fig,ax=plt.subplots()
        ax.plot(list(dic.keys()),list(dic.values()))
        plt.xlabel('number of cluster')
        plt.ylabel('inertia')
        plt.show()
```



6.1.6 [5.1.6] Wordclouds of clusters obtained after applying k-means on AVG W2V SET 3

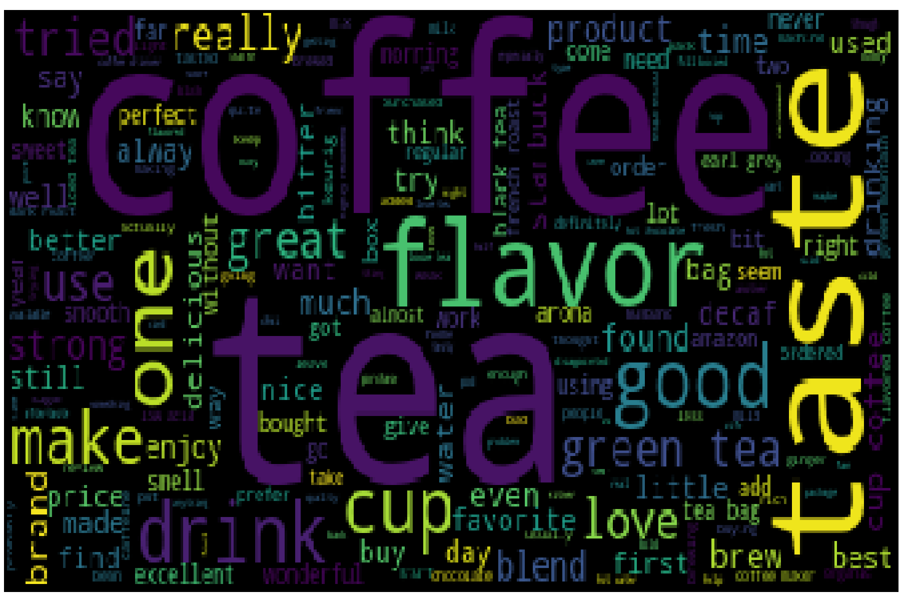
In [0]:
```
kmeans=KMeans(n_clusters=5)
kmeans.fit(sent_vectors)
labels=kmeans.predict(sent_vectors)
```

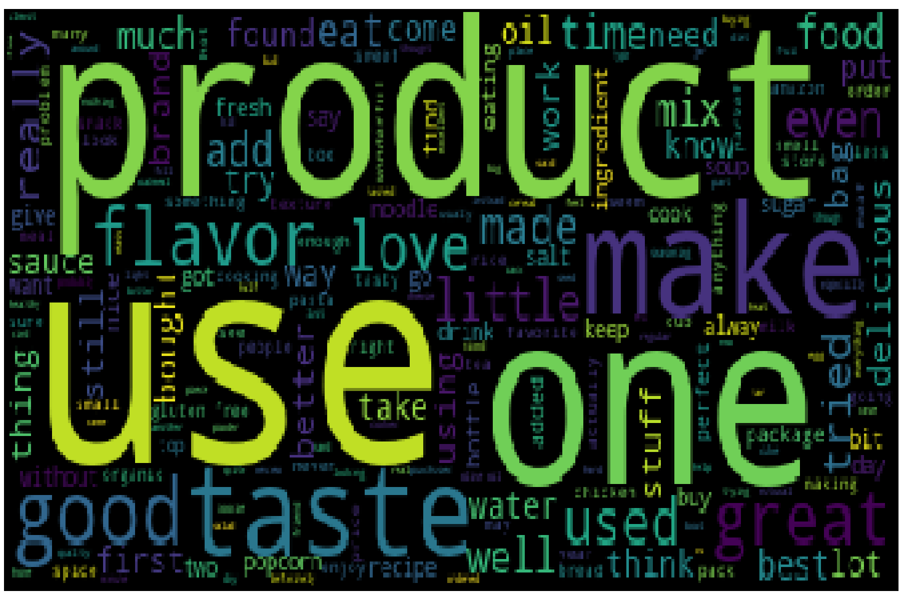
In [0]:
```
wordsofcluster1=[]
wordsofcluster2=[]
wordsofcluster3=[]
wordsofcluster4=[]
wordsofcluster5=[]
```

In [0]:
```
for i in range(len(labels)):
  if labels[i]==0:
    wordsofcluster1.append(x[i])
  elif labels[i]==1:
    wordsofcluster2.append(x[i])
  elif labels[i]==2:
    wordsofcluster3.append(x[i])
  elif labels[i]==3:
    wordsofcluster4.append(x[i])
  else:
    wordsofcluster5.append(x[i])
```

In [0]:
```
from wordcloud import WordCloud
from matplotlib.pyplot import figure
```

```
import matplotlib.pyplot as plt1
word_cloud = WordCloud(relative_scaling = 1.0).generate(str(wordsofcluster1))
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
word_cloud = WordCloud(relative_scaling = 1.0).generate(str(wordsofcluster2))
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
word_cloud = WordCloud(relative_scaling = 1.0).generate(str(wordsofcluster3))
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()

word_cloud = WordCloud(relative_scaling = 1.0).generate(str(wordsofcluster4))
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()

word_cloud = WordCloud(relative_scaling = 1.0).generate(str(wordsofcluster5))
plt.imshow(word_cloud,aspect='auto')
plt.axis('off')
plt.show()
```



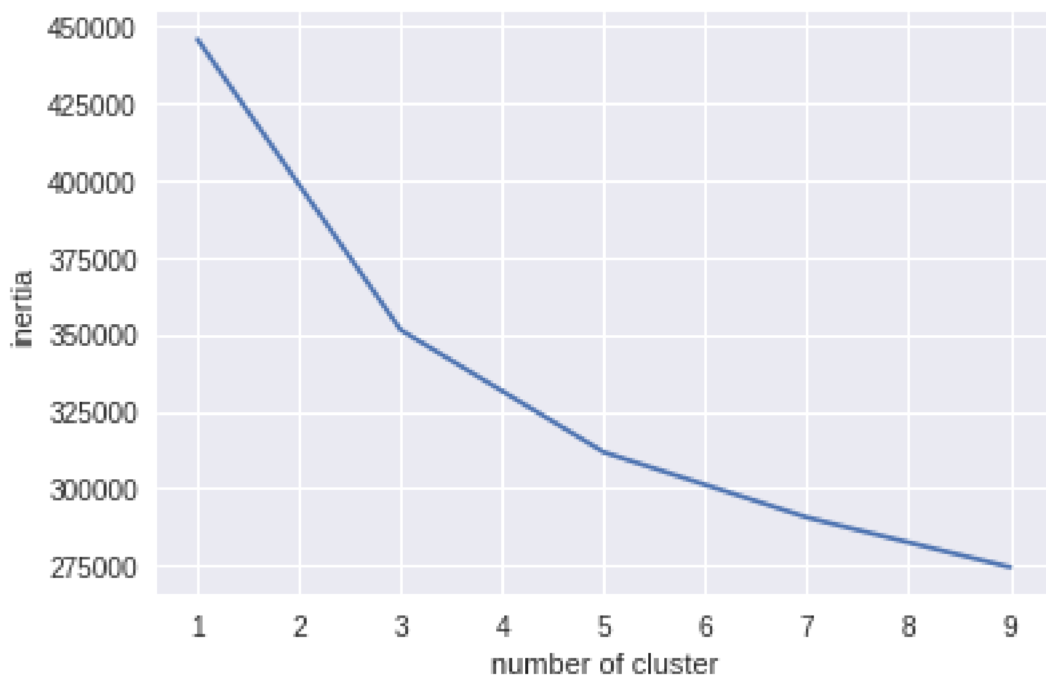

dog
food
treat
one
cat
product
love
great
bought
think
eat
little
time
really
healthy
found
well
kibble
happy
toy
bag
chew
still
look
use
better
need
feed
diet
stuff
ingredient
give
chicken
eating
brand
seem
used
got
dog food
cat food
dog love
formula
keep
greenie
make
first
pet
problem
smell
year old
work
put
try
flavor
even
two
see
take
good
something
day
dogs love
getting
buy
price
much
actually
come
vet
natural
puppy
find
week
teeth
hard
month
amazon
expensive
anything
never
best
know
gave
small
feeding
thing
organic
way
around
help
alway
bit
less
sure
another
done
giving

product
use
one
make
taste
good
flavor
love
great
little
made
mix
even
know
food
time
need
oil
come
eat
found
much
really
brand
add
try
fresh
say
work
ingredient
tried
bag
delicious
still
thought
better
take
using
bottle
stuff
perfect
package
bit
day
water
used
well
think
best
lot
first
two
popcorn
recipe
sauce
give
want
got
way
thing
without
drink
keep
alway
salt
noodle
soup
sugar
cook
anything
actually
added
buy
making
gluten free
organic
spice
store
people
right

- cluster1 represents coffee,tea and flavor of them
- cluster2 clearly represents dog cat and cat food
- cluster3 clearly represents product flavor andgood taste

- cluster4 represents amazon bags and related to buying things
- cluster5 represents taste flavor sweet cookie etcc...

6.1.7 [5.1.7] Applying K-Means Clustering on TFIDF W2V, SET 4

```
In [0]: from sklearn.cluster import KMeans
        dic={}
        for i in range(1,10,2):
          kmeans=KMeans(n_clusters=i)
          kmeans.fit(tfidf_sent_vectors)
          dic[i]=kmeans.inertia_
        fig,ax=plt.subplots()
        ax.plot(list(dic.keys()),list(dic.values()))
        plt.xlabel('number of cluster')
        plt.ylabel('inertia')
        plt.show()
```



6.1.8 [5.1.8] Wordclouds of clusters obtained after applying k-means on TFIDF W2V SET 4

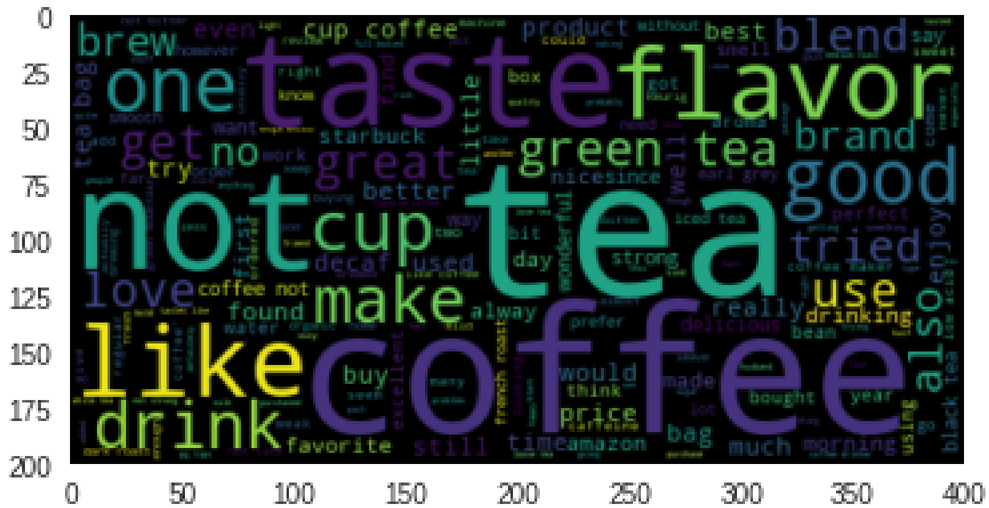
```
In [0]: kmeans=KMeans(n_clusters=5)
        kmeans.fit(tfidf_sent_vectors)
        labels=kmeans.predict(tfidf_sent_vectors)
```

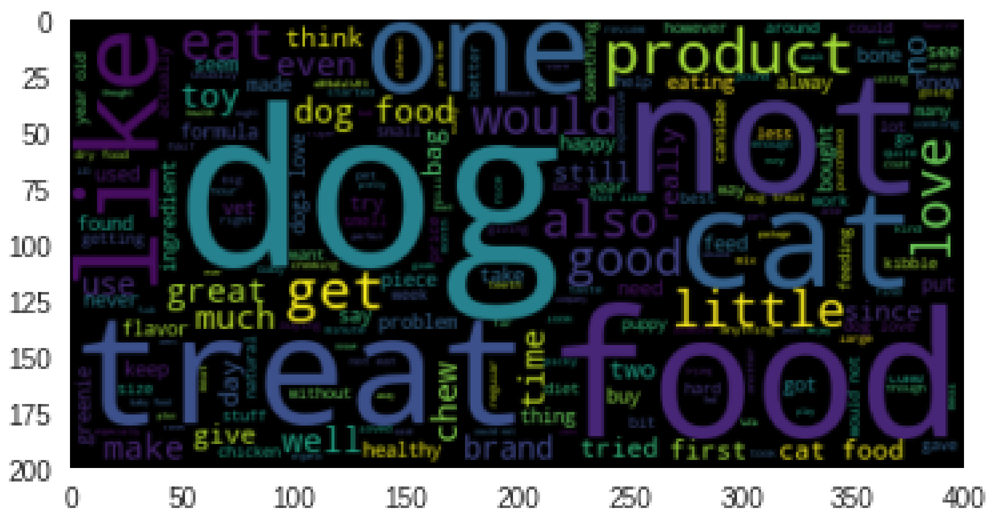
```
In [0]: wordsofcluster1=[]
        wordsofcluster2=[]
```

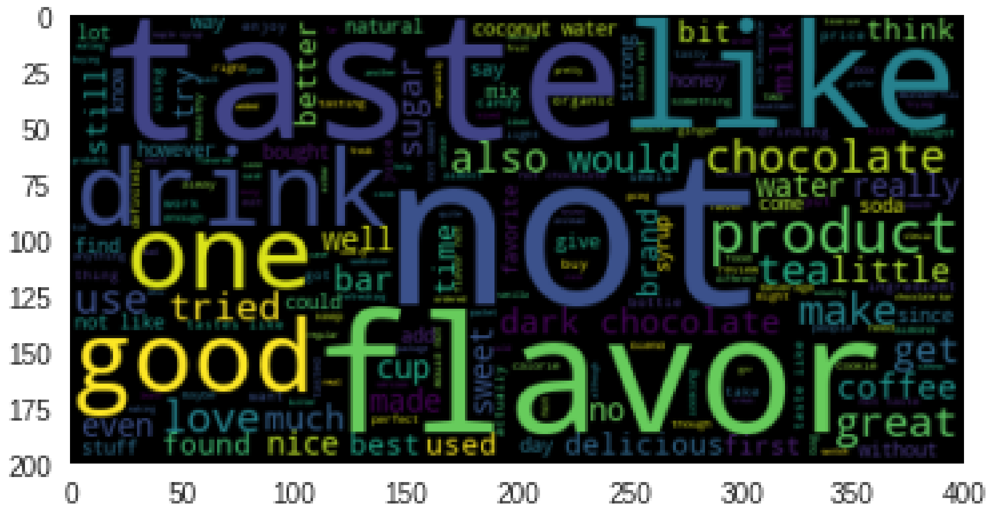
```
wordsofcluster3=[]
wordsofcluster4=[]
wordsofcluster5=[]
```

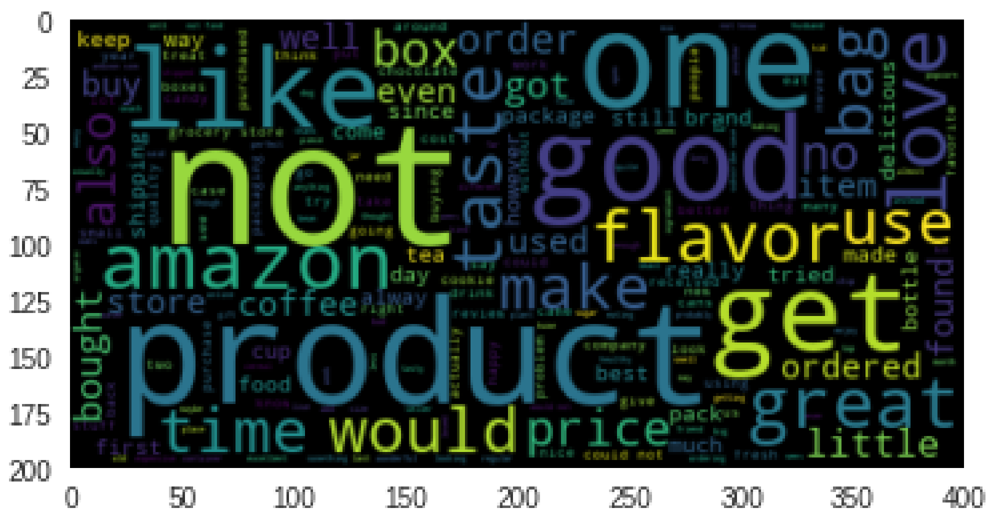
In [0]:
```
for i in range(len(labels)):
  if labels[i]==0:
    wordsofcluster1.append(x[i])
  elif labels[i]==1:
    wordsofcluster2.append(x[i])
  elif labels[i]==2:
    wordsofcluster3.append(x[i])
  elif labels[i]==3:
    wordsofcluster4.append(x[i])
  else:
    wordsofcluster5.append(x[i])
```

In [0]:
```
from wordcloud import WordCloud
from matplotlib.pyplot import figure
import matplotlib.pyplot as plt1

wc=WordCloud(stopwords=stopwords)
wc.generate(str(wordsofcluster1))
plt.grid(False)
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
wc=WordCloud(stopwords=stopwords)
wc.generate(str(wordsofcluster2))
plt.grid(False)
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
wc=WordCloud(stopwords=stopwords)
wc.generate(str(wordsofcluster3))
plt.grid(False)
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
wc=WordCloud(stopwords=stopwords)
wc.generate(str(wordsofcluster4))
plt.grid(False)
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
wc=WordCloud(stopwords=stopwords)
wc.generate(str(wordsofcluster5))
plt.grid(False)
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
```


0
25
50
75
100
125
150
175
200
0
50
100
150
200
250
300
350
400
taste
flavor
one
brew
cup coffee
blend
product
best
get
no
great
green tea
brand
not
tea
good
cup
tried
love
make
use
like
coffee
also
drink
buy
price
bag
favorite
still
time
amazon
much
morning
starbuck
better
decaf
used
found
coffee not
really
drinking
delicious
would
made
bought
year
try
little
first
strong
day
alway
prefer
bean
water
excellent
french roast
black tea
enjoy
perfect
iced tea
earl grey
nice
since
aroma
smell
box
got
even
say
wonderful
bit
think
caffeine
lot
using
go
tea bag


0
25
50
75
100
125
150
175
200
0
50
100
150
200
250
300
350
400
like
eat
think
one
product
even
seen
toy
dog food
would
not
formula
dog
also
cat
love
good
get
great
much
little
treat
food
flavor
problem
piece
chew
time
two
give
well
healthy
brand
tried
first
cat food
make
keep
without
thing
buy
got
day
ingredient
found
use
never
feed
feeding
kibble
put
since
bone
eating
alway
happy
still
really
bought
best
try
vet
bag
take
need
puppy
hard
diet
size
stuff
chicken
greenie
made
see
know
lot
less
used
dry food
year old


taste
like
not
drink
one
flavor
good
product
chocolate
also would
tealittle
make
dark chocolate
coffee
great
get
love
much
found
nice
best
used
delicious
first
tried
use
well
bar
cup
water
really
0
25
50
75
100
125
150
175
200
0
50
100
150
200
250
300
350
400


like
one
not
good
product
get
amazon
flavor
use
taste
love
bag
make
time
would
price
great
little
ordered
box
order
store
coffee
also
bought
used
got
well
buy
even
0
25
50
75
100
125
150
175
200
0
50
100
150
200
250
300
350
400

- cluster1 represents coffee,tea and flavor of them
- cluster2 clearly represents dog cat and cat food
- cluster3 clearly represents product flavor andgood taste
- cluster4 represents amazon bags and related to buying things
- cluster5 represents like love delicious etcc...

6.2 [5.2] Agglomerative Clustering

6.2.1 [5.2.1] Applying Agglomerative Clustering on AVG W2V, SET 3

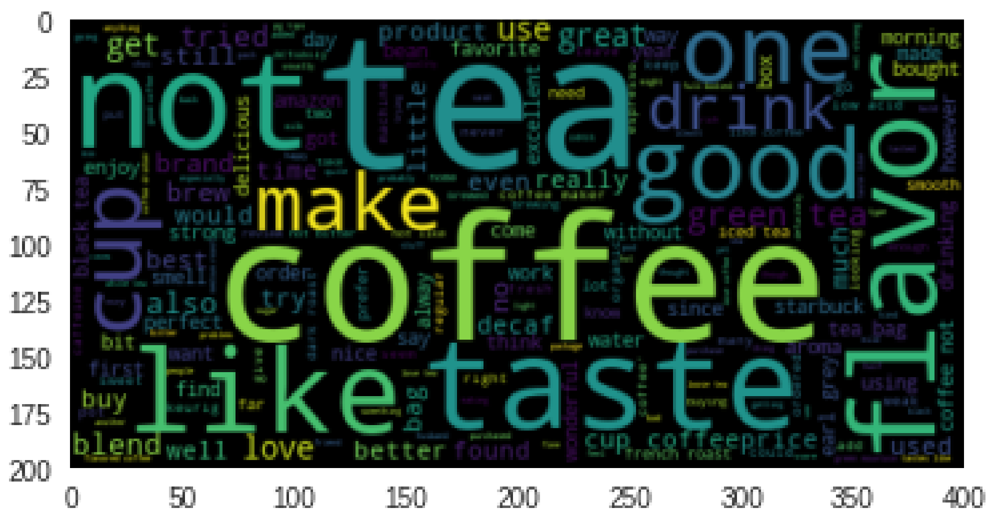
```
In [0]: from sklearn.cluster import AgglomerativeClustering
        sent_vectorsspeciall=np.array(sent_vectorsspecial)
        dic={}
        agg = AgglomerativeClustering(n_clusters = 2)
        agg.fit(sent_vectorsspeciall)
```

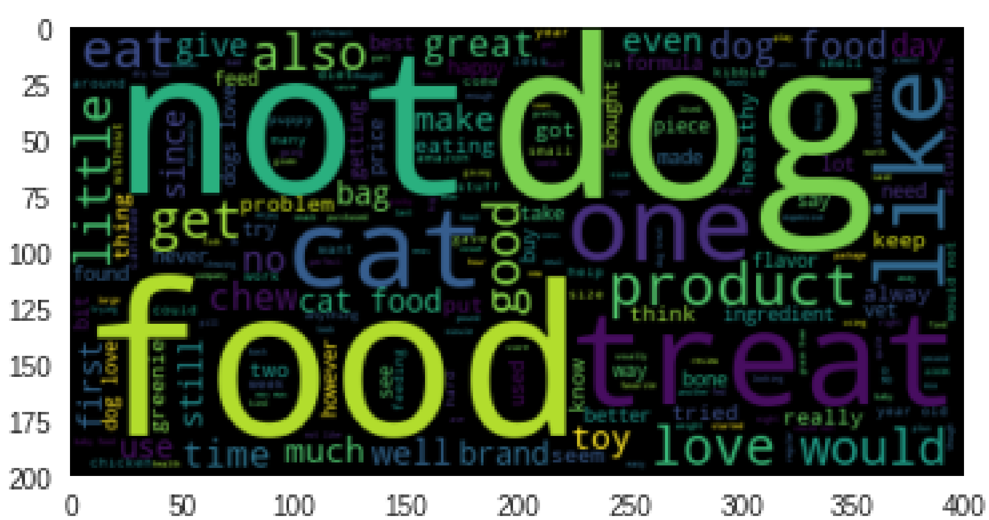
```
Out[0]: AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                    connectivity=None, linkage='ward', memory=None, n_clusters=2,
                    pooling_func='deprecated')
```

```
In [0]: wordsofcluster1=[]
        wordsofcluster2=[]
```

```
In [0]: for i in range(len(labels)):
          if labels[i]==0:
            wordsofcluster1.append(x[i])
          elif labels[i]==1:
            wordsofcluster2.append(x[i])
```

```
In [0]: from wordcloud import WordCloud
        from matplotlib.pyplot import figure
        import matplotlib.pyplot as plt1

        wc=WordCloud(stopwords=stopwords)
        wc.generate(str(wordsofcluster1))
        plt.grid(False)
        plt1.imshow(wc,interpolation='bilinear')
        plt1.show()
        wc=WordCloud(stopwords=stopwords)
        wc.generate(str(wordsofcluster2))
        plt.grid(False)
        plt1.imshow(wc,interpolation='bilinear')
        plt1.show()
```





- cluster1 represents coffee,tea and flavor of them
- cluster2 clearly represents dog cat and cat food,treat

6.2.2 [5.2.2] Wordclouds of clusters obtained after applying Agglomerative Clustering on AVG W2V SET 3

In [0]:
```
from sklearn.cluster import AgglomerativeClustering
sent_vectorsspeciall=np.array(sent_vectorsspecial)
dic={}
agg = AgglomerativeClustering(n_clusters = 5)
agg.fit(sent_vectorsspeciall)
```

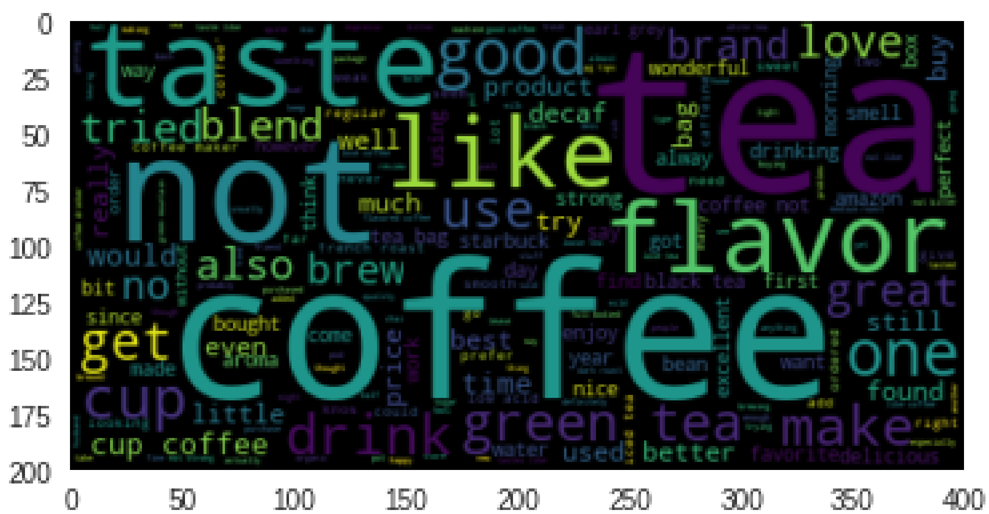
Out[0]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
            connectivity=None, linkage='ward', memory=None, n_clusters=5,
            pooling_func='deprecated')
```

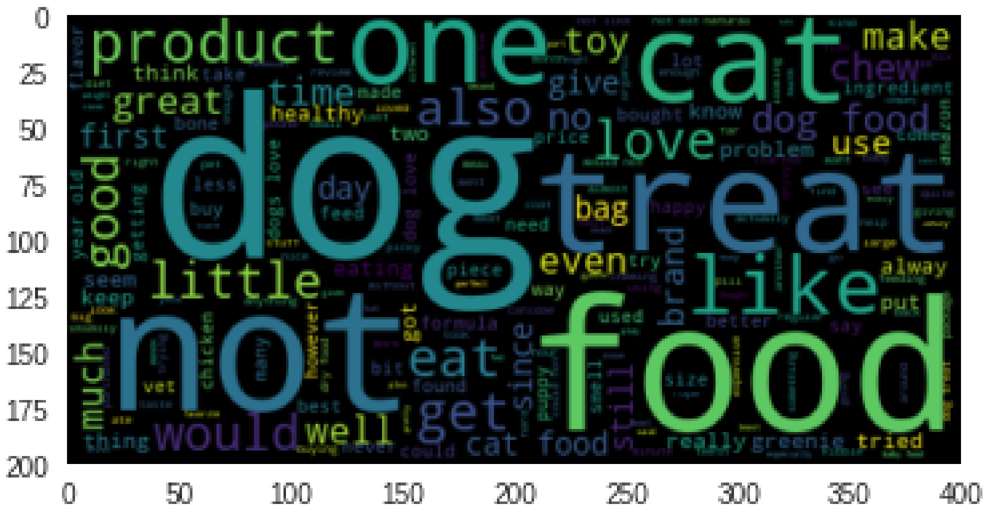
In [0]:
```
wordsofcluster1=[]
wordsofcluster2=[]
wordsofcluster3=[]
wordsofcluster4=[]
wordsofcluster5=[]
```

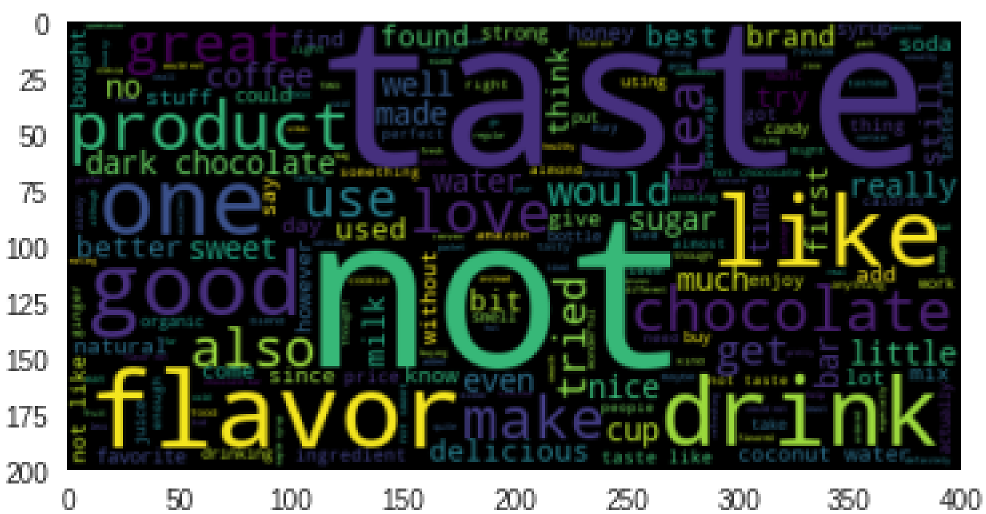
In [0]:
```
for i in range(len(labels)):
  if labels[i]==0:
    wordsofcluster1.append(x[i])
  elif labels[i]==1:
    wordsofcluster2.append(x[i])
  elif labels[i]==2:
    wordsofcluster3.append(x[i])
  elif labels[i]==3:
    wordsofcluster4.append(x[i])
  else:
    wordsofcluster5.append(x[i])
```

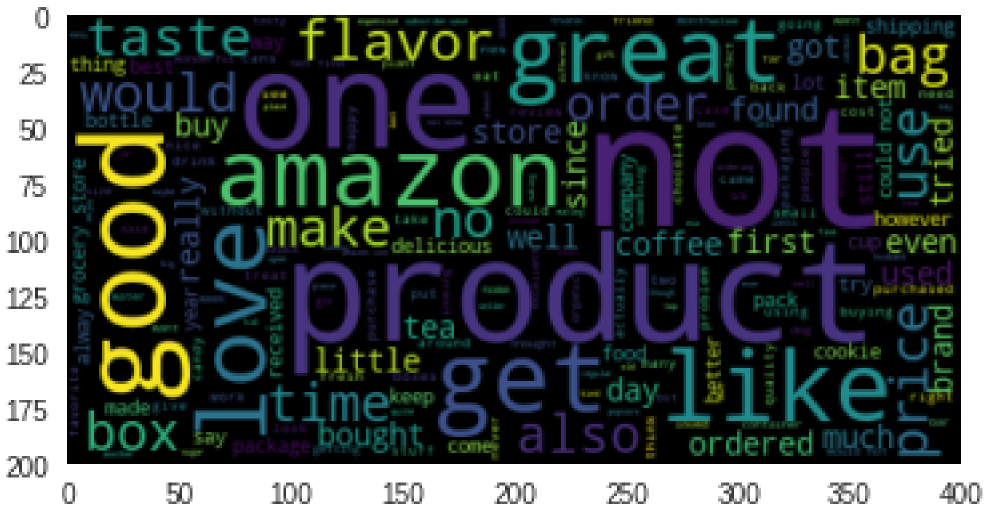
In [0]:
```
from wordcloud import WordCloud
from matplotlib.pyplot import figure
import matplotlib.pyplot as plt1

wc=WordCloud(stopwords=stopwords)
wc.generate(str(wordsofcluster1))
plt.grid(False)
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
wc=WordCloud(stopwords=stopwords)
wc.generate(str(wordsofcluster2))
plt.grid(False)
```

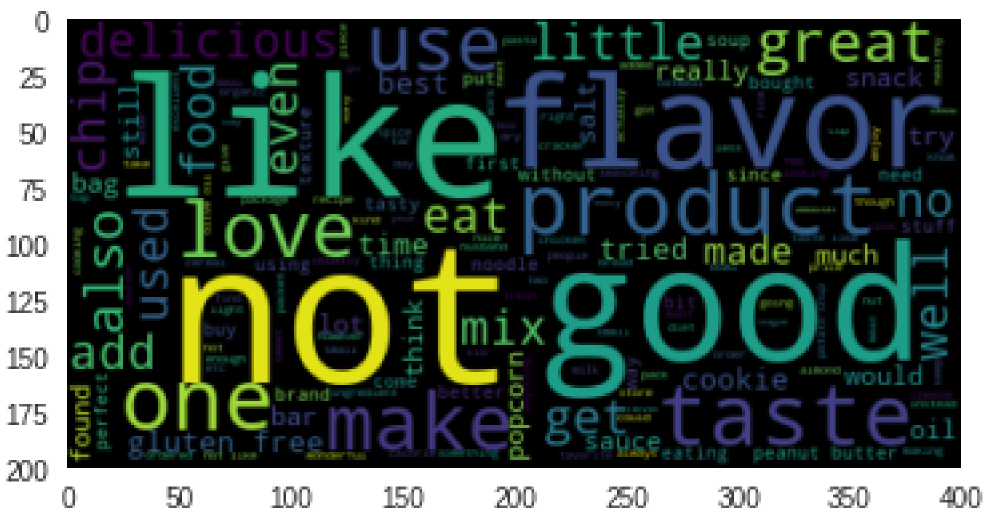
```
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
wc=WordCloud(stopwords=stopwords)
wc.generate(str(wordsofcluster3))
plt.grid(False)
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
wc=WordCloud(stopwords=stopwords)
wc.generate(str(wordsofcluster4))
plt.grid(False)
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
wc=WordCloud(stopwords=stopwords)
wc.generate(str(wordsofcluster5))
plt.grid(False)
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
```




dog
treat
not
food
one
cat
product
like
little
good
love
also
great
time
toy
make
chew
dog food
eat
get
would
well
cat food
first
much
bag
even
brand
still
since
give
no
use
day
healthy
problem
really
greenie
tried
think
0
25
50
75
100
125
150
175
200
0
50
100
150
200
250
300
350
400


taste
not
flavor
drink
like
chocolate
good
one
product
great
love
use
tea
also
make
get
little
dark chocolate
coffee
would
found
best
brand
really
sugar
sweet
better
much
tried
first
time
milk
delicious
nice
even
cup
bar
coconut water
taste like
0
25
50
75
100
125
150
175
200
0
50
100
150
200
250
300
350
400





- cluster1 represents coffee,tea and flavor of them
- cluster2 clearly represents dog cat and cat food
- cluster3 clearly represents product flavor andgood taste
- cluster4 represents amazon bags and related to buying things
- cluster5 represents like love delicious etcc...

6.2.3 [5.2.3] Applying Agglomerative Clustering on TFIDF W2V, SET 4

In [0]:
```
from sklearn.cluster import AgglomerativeClustering
tfidf_sent_vectorsspeciall=np.array(tfidf_sent_vectorsspecial)
dic={}
```

```
agg = AgglomerativeClustering(n_clusters = 2)
agg.fit(tfidf_sent_vectorsspeciall)
```

Out[0]:

```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
            connectivity=None, linkage='ward', memory=None, n_clusters=2,
            pooling_func='deprecated')
```

In [0]:

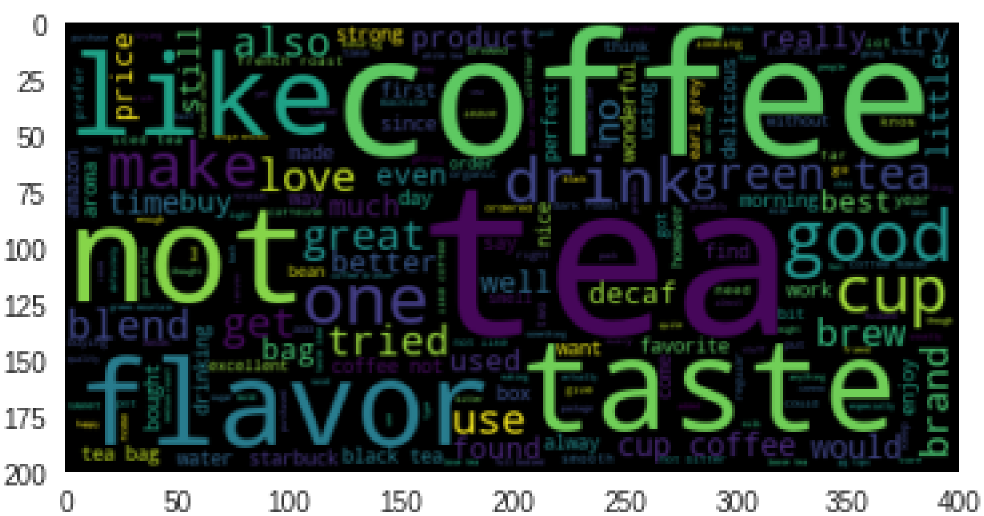
```
wordsofcluster1=[]
wordsofcluster2=[]
```

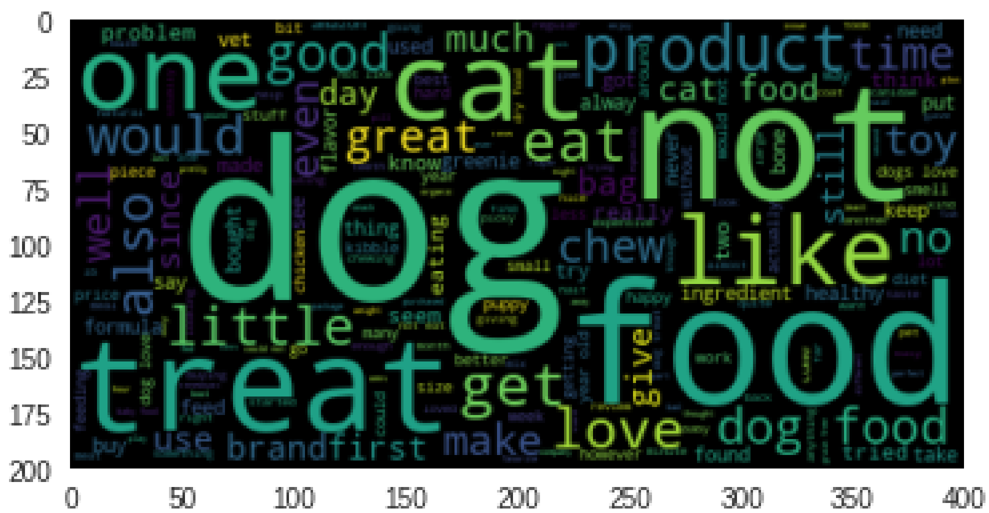
In [0]:

```
for i in range(len(labels)):
  if labels[i]==0:
    wordsofcluster1.append(x[i])
  elif labels[i]==1:
    wordsofcluster2.append(x[i])
```

In [0]:

```
from wordcloud import WordCloud
from matplotlib.pyplot import figure
import matplotlib.pyplot as plt1

wc=WordCloud(stopwords=stopwords)
wc.generate(str(wordsofcluster1))
plt.grid(False)
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
wc=WordCloud(stopwords=stopwords)
wc.generate(str(wordsofcluster2))
plt.grid(False)
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
```





- cluster1 represents coffee,tea and flavor of them
- cluster2 clearly represents dog cat and cat food,treat

6.2.4 [5.2.4] Wordclouds of clusters obtained after applying Agglomerative Clustering on TFIDF W2V SET 4

In [0]:
```
from sklearn.cluster import AgglomerativeClustering
tfidf_sent_vectorsspeciall=np.array(tfidf_sent_vectorsspecial)
dic={}
agg = AgglomerativeClustering(n_clusters = 5)
agg.fit(tfidf_sent_vectorsspeciall)
```

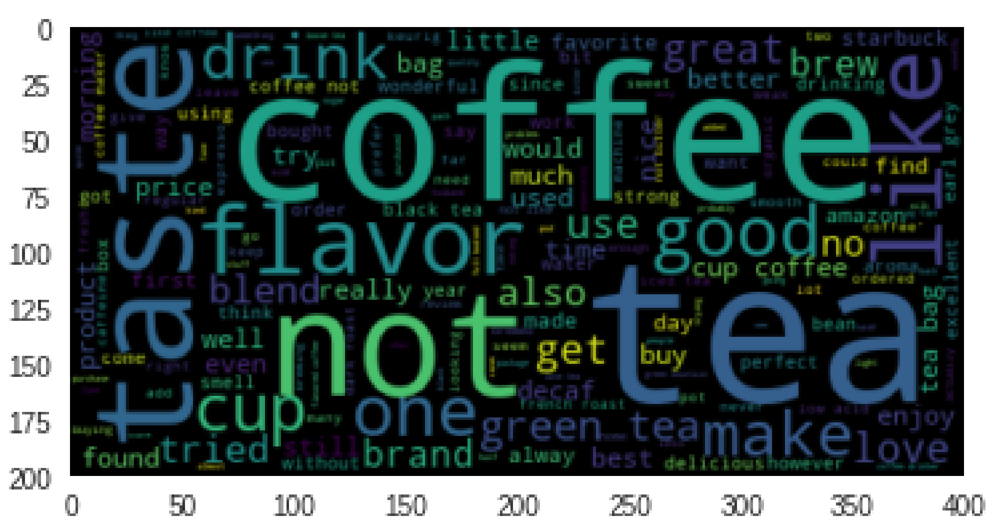
Out[0]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
            connectivity=None, linkage='ward', memory=None, n_clusters=5,
            pooling_func='deprecated')
```

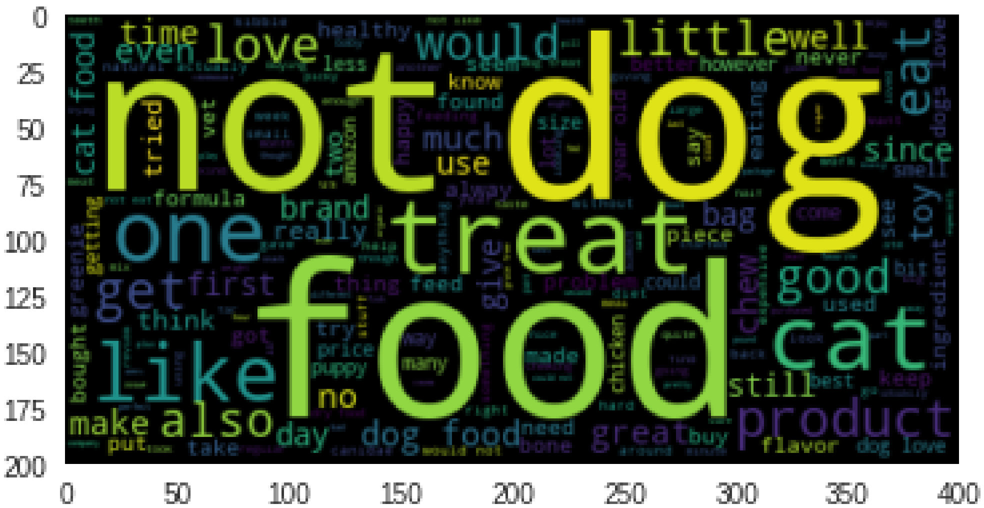
In [0]:
```
wordsofcluster1=[]
wordsofcluster2=[]
wordsofcluster3=[]
wordsofcluster4=[]
wordsofcluster5=[]
```

In [0]:
```
for i in range(len(labels)):
  if labels[i]==0:
    wordsofcluster1.append(x[i])
  elif labels[i]==1:
    wordsofcluster2.append(x[i])
  elif labels[i]==2:
    wordsofcluster3.append(x[i])
  elif labels[i]==3:
```

```
        wordsofcluster4.append(x[i])
      else:
        wordsofcluster5.append(x[i])
```

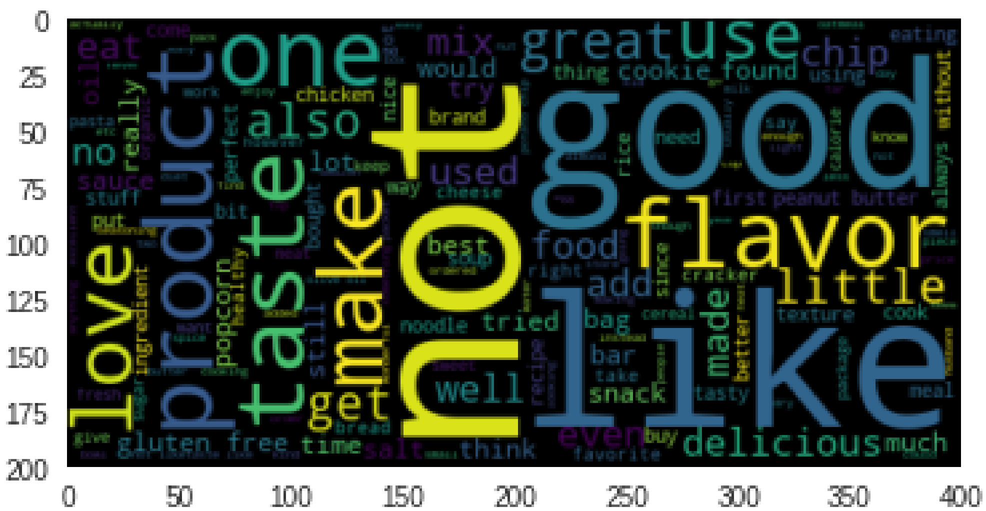
In [0]:
```
from wordcloud import WordCloud
from matplotlib.pyplot import figure
import matplotlib.pyplot as plt1

wc=WordCloud(stopwords=stopwords)
wc.generate(str(wordsofcluster1))
plt.grid(False)
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
wc=WordCloud(stopwords=stopwords)
wc.generate(str(wordsofcluster2))
plt.grid(False)
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
wc=WordCloud(stopwords=stopwords)
wc.generate(str(wordsofcluster3))
plt.grid(False)
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
wc=WordCloud(stopwords=stopwords)
wc.generate(str(wordsofcluster4))
plt.grid(False)
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
wc=WordCloud(stopwords=stopwords)
wc.generate(str(wordsofcluster5))
plt.grid(False)
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
```





not
dog
food
treat
one
like
cat
product
good
would
little
well
love
time
even
get
first
also
make
eat
since
toy
bag
brand
really
chew
still
give
much
use
treat
great
day
dog food
flavor
dog love
0
25
50
75
100
125
150
175
200
0
50
100
150
200
250
300
350
400

taste
not
good
one
product
flavor
like
drink
chocolate
great
love
would
make
also
tea
use
tried
really
delicious
much
sweet
dark chocolate
even
get
little
coffee
water
coconut water
nice
0
25
50
75
100
125
150
175
200
0
50
100
150
200
250
300
350
400



- cluster1 represents coffee,tea and flavor of them
- cluster2 clearly represents dog cat and cat food
- cluster3 clearly represents product flavor andgood taste
- cluster4 represents amazon bags and related to buying things
- cluster5 represents like love delicious etcc...

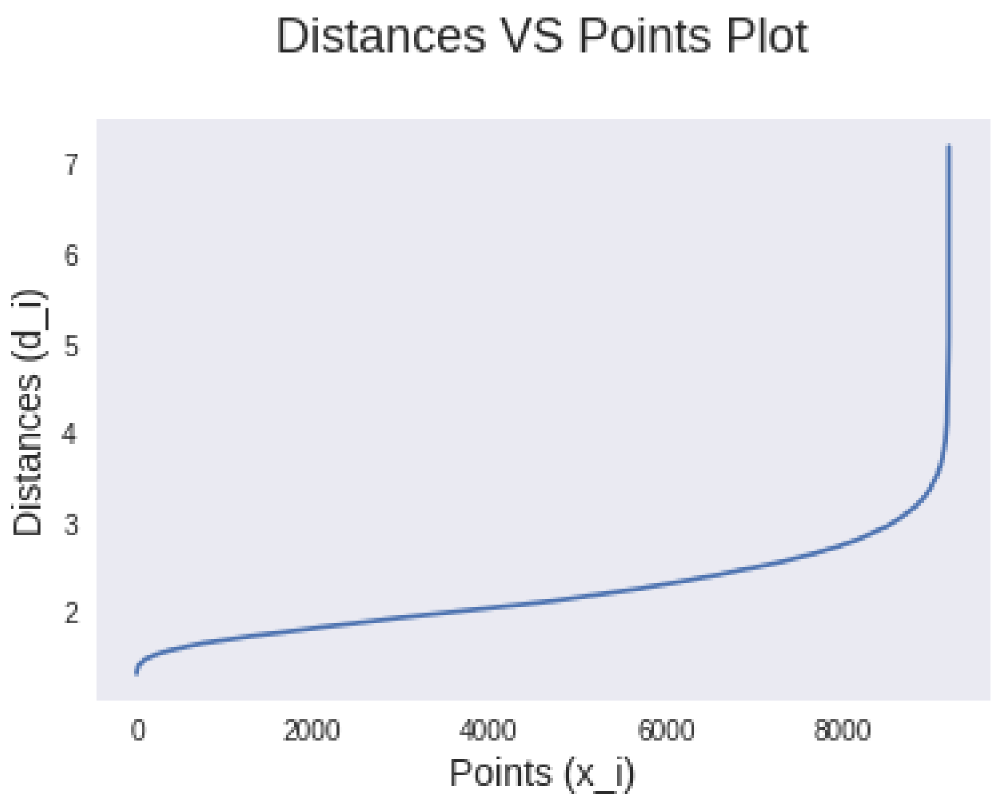
6.3 [5.3] DBSCAN Clustering

6.3.1 [5.3.1] Applying DBSCAN on AVG W2V, SET 3

```
In [0]: def neighbour(vectors , n):
            distance = []
```

```
    for point in vectors:
        temp = np.sort(np.sum((vectors-point)**2,axis=1),axis=None)
        distance.append(temp[n])
    return np.sqrt(np.array(distance))
```

In [0]:
```
data=np.array(sent_vectorsspecial)
min_points = 2*data.shape[1]

distances = neighbour(data,min_points)
sorted_distance = np.sort(distances)
points = [i for i in range(data.shape[0])]
plt.plot(points, sorted_distance)
plt.xlabel('Points (x_i)',size=14)
plt.ylabel('Distances (d_i)',size=14)
plt.title('Distances VS Points Plot\n',size=18)
plt.grid()
plt.show()
```



In [0]:
```
print(min_points)
```

```
100
```

6.3.2 [5.3.2] Wordclouds of clusters obtained after applying DBSCAN on AVG W2V SET 3

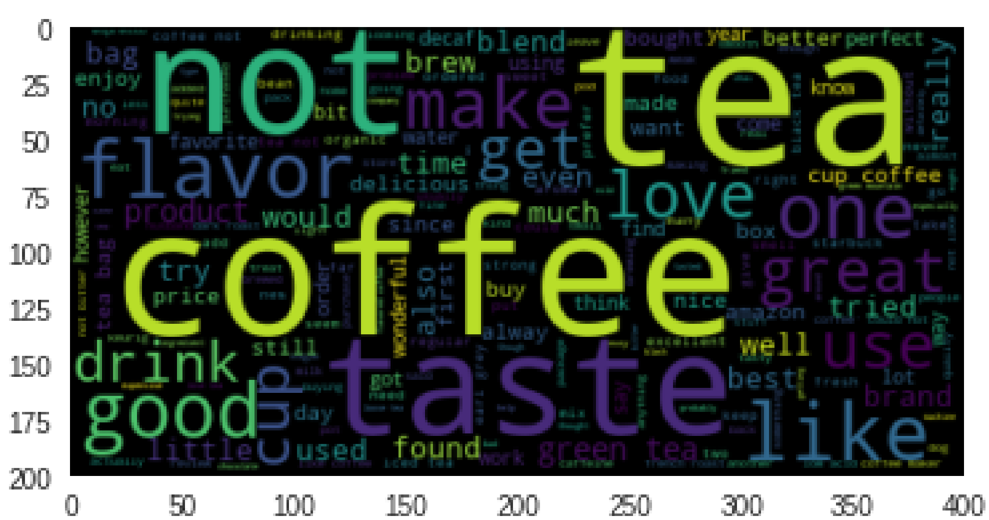
```
In [0]: from sklearn.cluster import DBSCAN
        sent_vectorsspeciall=np.array(sent_vectorsspecial)
        dic={}
        agg = DBSCAN(eps =2,min_samples=100)
        agg.fit(sent_vectorsspeciall)
```

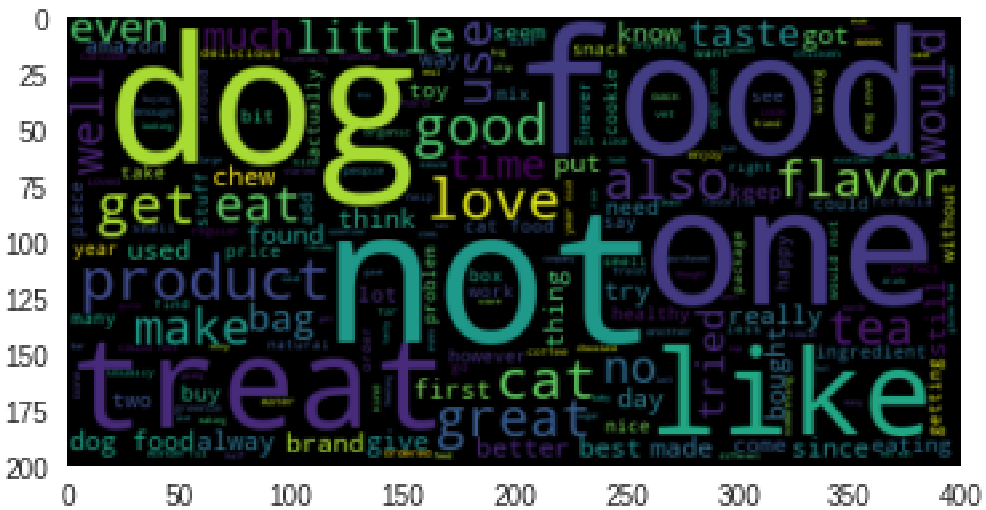
```
Out[0]: DBSCAN(algorithm='auto', eps=2, leaf_size=30, metric='euclidean',
            metric_params=None, min_samples=100, n_jobs=None, p=None)
```

```
In [0]: wordsofcluster1=[]
        wordsofcluster2=[]
```

```
In [0]: for i in range(len(labels)):
          if labels[i]==-1:
            wordsofcluster1.append(x[i])
          elif labels[i]==0:
            wordsofcluster2.append(x[i])
```

```
In [0]: from wordcloud import WordCloud
        from matplotlib.pyplot import figure
        import matplotlib.pyplot as plt1

        wc=WordCloud(stopwords=stopwords)
        wc.generate(str(wordsofcluster1))
        plt.grid(False)
        plt1.imshow(wc,interpolation='bilinear')
        plt1.show()
        wc=WordCloud(stopwords=stopwords)
        wc.generate(str(wordsofcluster2))
        plt.grid(False)
        plt1.imshow(wc,interpolation='bilinear')
        plt1.show()
```

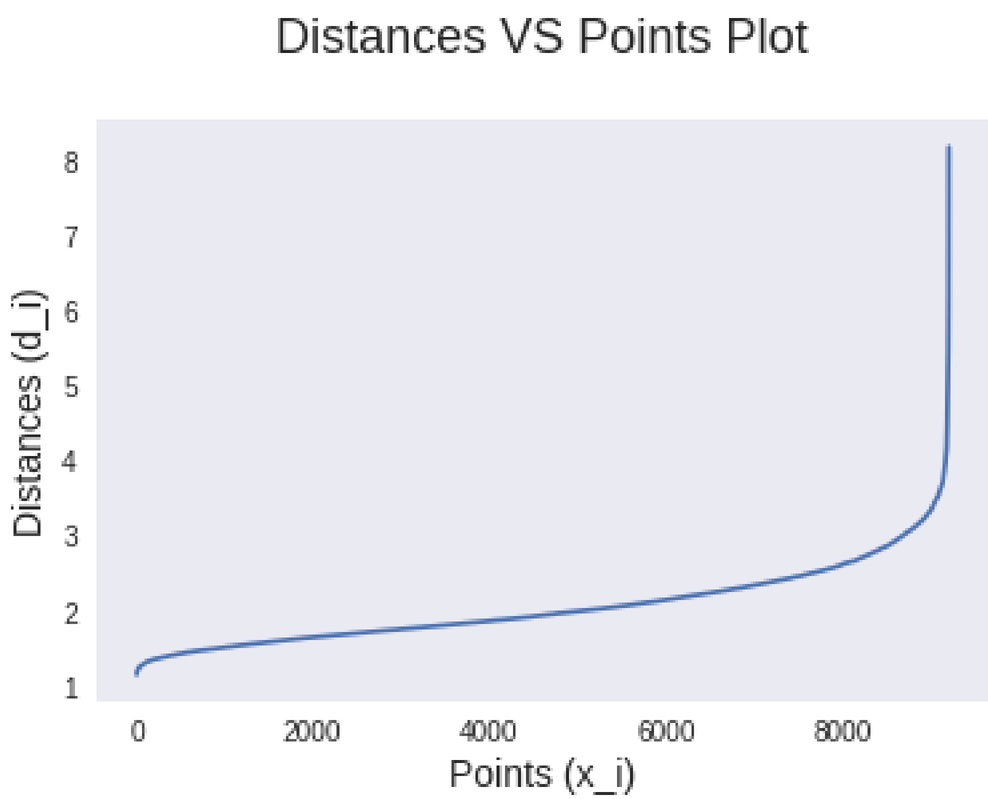




- cluster1 represents coffee,tea and flavor of them
- cluster2 clearly represents dog cat and cat food

6.3.3 [5.3.3] Applying DBSCAN on TFIDF W2V, SET 4

In [0]:
```
data=np.array(tfidf_sent_vectorsspecial)
min_points = 2*data.shape[1]
distances = n_neighbour(data,min_points)
sorted_distance = np.sort(distances)
points = [i for i in range(data.shape[0])]
plt.plot(points, sorted_distance)
plt.xlabel('Points (x_i)',size=14)
plt.ylabel('Distances (d_i)',size=14)
plt.title('Distances VS Points Plot\n',size=18)
plt.grid()
plt.show()
```



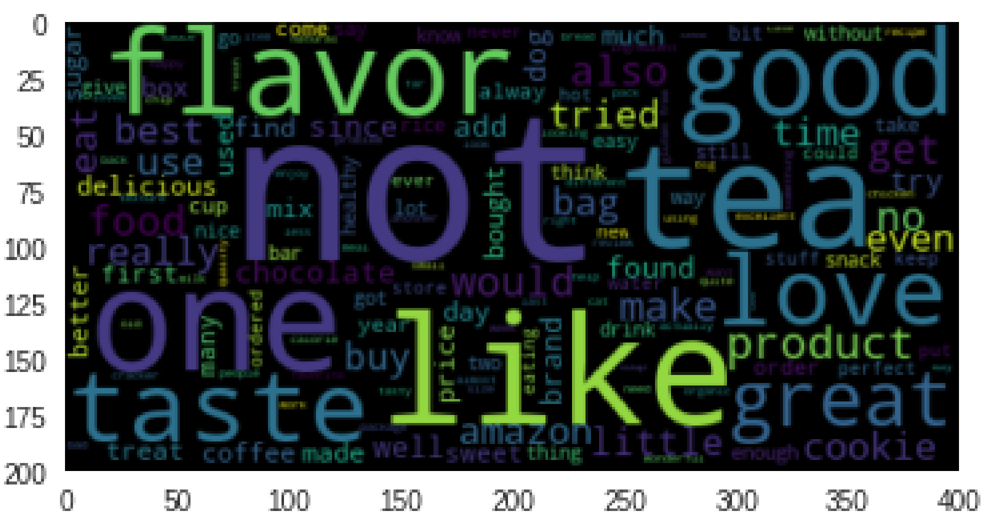
6.3.4 [5.3.4] Wordclouds of clusters obtained after applying DBSCAN on TFIDF W2V SET 4

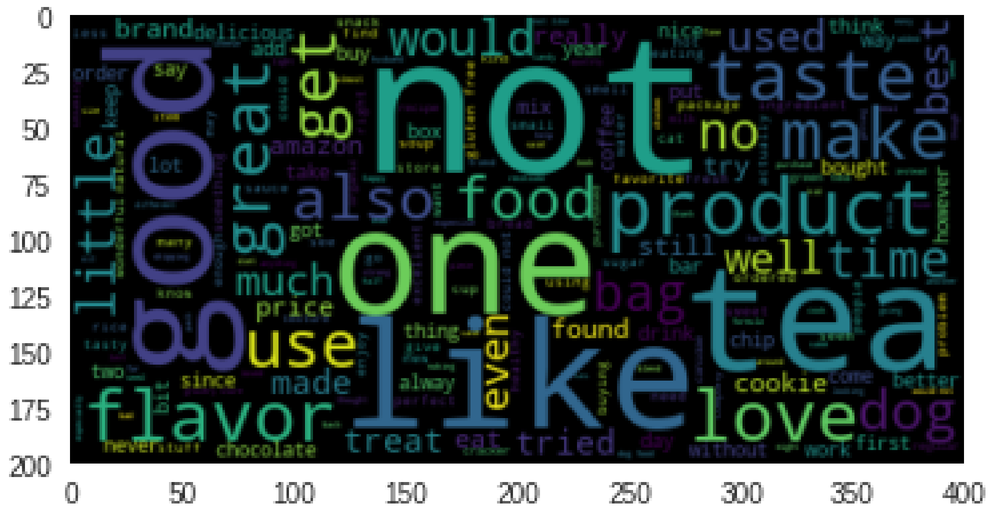
```
In [0]: from sklearn.cluster import DBSCAN
        tfidf_sent_vectorsspeciall=np.array(tfidf_sent_vectorsspecial)
        dic={}
        agg = DBSCAN(eps =2,min_samples=100)
        agg.fit(tfidf_sent_vectorsspeciall)

Out[0]: DBSCAN(algorithm='auto', eps=2, leaf_size=30, metric='euclidean',
            metric_params=None, min_samples=100, n_jobs=None, p=None)

In [0]: print(*agg.labels_)

-1 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 -1 -1 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

In [0]: wordsofcluster1=[]
        wordsofcluster2=[]

In [0]: labels=agg.labels_
```

labels which are -1 indicatees the words that words that are present in outliers

```
In [0]: for i in range(len(labels)):
          if labels[i]==-1:
            wordsofcluster1.append(x[i])
          elif labels[i]==0:
            wordsofcluster2.append(x[i])
```

```
In [0]: from wordcloud import WordCloud
        from matplotlib.pyplot import figure
        import matplotlib.pyplot as plt1

        wc=WordCloud(stopwords=stopwords)
        wc.generate(str(wordsofcluster1))
        plt.grid(False)
        plt1.imshow(wc,interpolation='bilinear')
        plt1.show()
        wc=WordCloud(stopwords=stopwords)
        wc.generate(str(wordsofcluster2))
        plt.grid(False)
        plt1.imshow(wc,interpolation='bilinear')
        plt1.show()
```

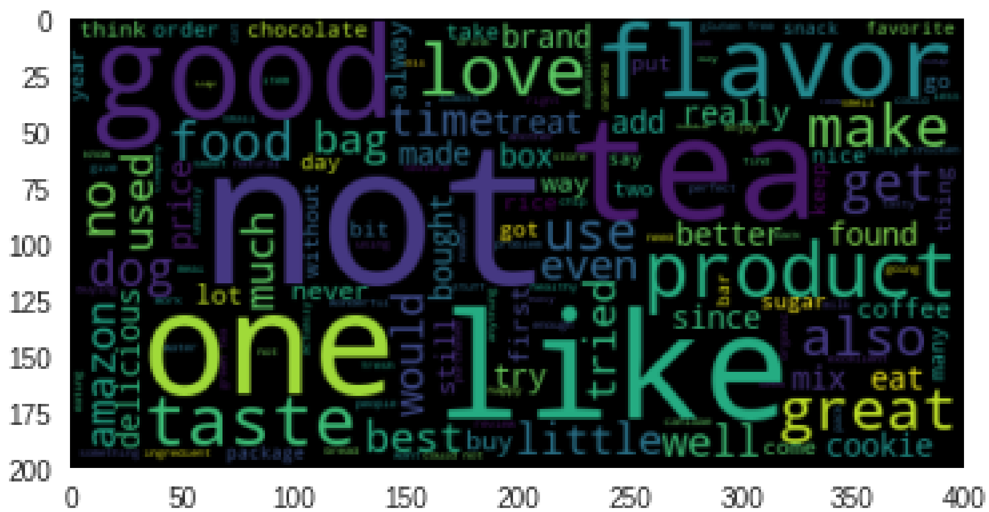


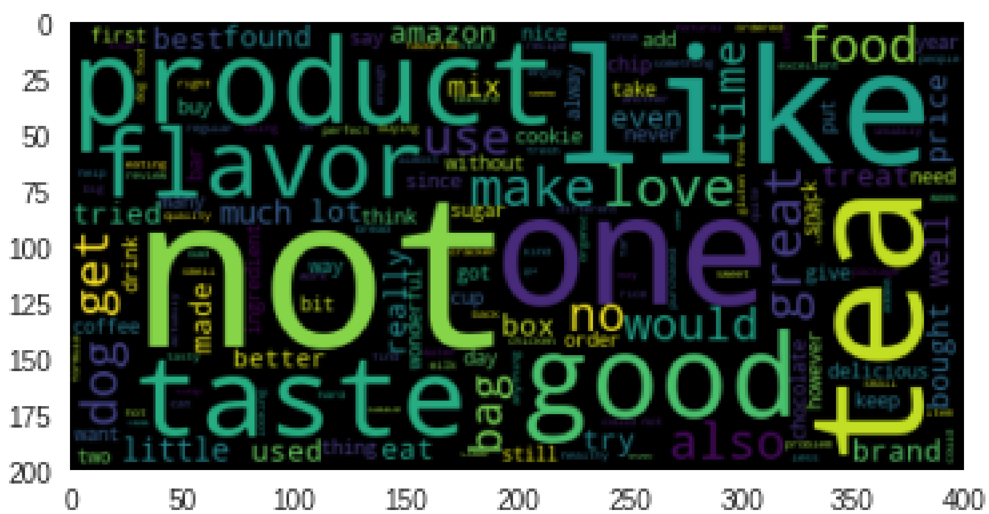


- cluster1 represents coffee,tea and flavor of them
- cluster2 clearly represents good food and products

```
In [0]: from sklearn.cluster import DBSCAN
        tfidf_sent_vectorsspeciall=np.array(tfidf_sent_vectorsspecial)
        dic={}
        agg = DBSCAN(eps =1.5,min_samples=100)
        agg.fit(tfidf_sent_vectorsspeciall)

Out[0]: DBSCAN(algorithm='auto', eps=1.5, leaf_size=30, metric='euclidean',
            metric_params=None, min_samples=100, n_jobs=None, p=None)

In [0]: wordsofcluster1=[]
        wordsofcluster2=[]

In [0]: labels=agg.labels_

In [0]: for i in range(len(labels)):
          if labels[i]==-1:
            wordsofcluster1.append(x[i])
          elif labels[i]==0:
            wordsofcluster2.append(x[i])


In [0]: from wordcloud import WordCloud
        from matplotlib.pyplot import figure
        import matplotlib.pyplot as plt1

        wc=WordCloud(stopwords=stopwords)
        wc.generate(str(wordsofcluster1))
```

```
plt.grid(False)
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
wc=WordCloud(stopwords=stopwords)
wc.generate(str(wordsofcluster2))
plt.grid(False)
plt1.imshow(wc,interpolation='bilinear')
plt1.show()
```





- cluster1 clearly represents tea foodproduct
- cluster2 represents taste goood prioduct

7 [6] Conclusions

In [0]:
```
import pandas as pd
data = [['kmeans','bow',5],['kmeans','tfidf',5],['kmeans','avg.w2v',5],['kmeans','tfid
pd.DataFrame(data, columns=['clustering','type','number of clusters'],index=['1','2','3
```

Out[0]:

	clustering	type	number of clusters
1	kmeans	bow	5
2	kmeans	tfidf	5
3	kmeans	avg.w2v	5
4	kmeans	tfidf w2v	5

In [0]:
```
import pandas as pd
data = [['agglomerative','avg w2v',2],['agglomerative','tfidf w2v',5],['agglomerative'
pd.DataFrame(data, columns=['clustering','type','number of clusters'],index=['1','2','3
```

Out[0]:

	clustering	type	number of clusters
1	agglomerative	avg w2v	2
2	agglomerative	tfidf w2v	5
3	agglomerative	avg w2v	2
4	agglomerative	tfidf w2v	5

In [0]:
```
import pandas as pd
data = [['dbscan','avg w2v',2],['dbscan','tfidf w2v',5],['dbscan','avg w2v',2],['dbscan
pd.DataFrame(data, columns=['clustering','type','number of clusters'],index=['1','2','3
```

Out[0]:

	clustering	type	number of clusters
1	dbscan	avg w2v	2
2	dbscan	tfidf w2v	5
3	dbscan	avg w2v	2
4	dbscan	tfidf w2v	5