Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

Objective:

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

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

[Ans] I 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 my 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, I 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 sklearn
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
```

In [2]:
```
# 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 points
# you can change the number to any other number based on your computing power

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

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

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rating(0).
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)
```

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

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all	This is a confection that has been around a fe...

In [3]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
```

In [4]:
```
print(display.shape)
```

```
(80668, 7)
```

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B005ZBZLT4	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ESG	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B005ZBZLT4	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ESG	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBEV0	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [5]:

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B001ATMQK2	undertheshrine "undertheshrine"	1296691200	5	I bought this 6 pack because for the price tha...	5

In [6]:

Out[6]: 393063

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

In [7]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	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 [8]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='l
```

In [9]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
```

Out[9]: (364173, 10)

In [10]:
```
#Checking to see how much % of data still remains
```

Out[10]: 69.25890143662969

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 [11]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son at College	My son loves spaghetti so I didn't hesitate or...
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside	It was almost a 'love at first bite' - the per...

In [12]:

In [13]:
```
#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?
```

```
(364171, 10)
```

Out[13]:
```
1    307061
0     57110
Name: Score, dtype: int64
```

From the above result, we can see that data is imbalanced (Positive Reviews > Negative Reviews)

[3] Preprocessing

[3.1]. Preprocessing Review Text

Now that deduplication is finished for our data and 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 [14]:
```
# 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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along and he always can sing
the refrain. he's learned about whales, India, drooping roses:  i love all the new words this book  introduces and the
silliness of it all.  this is a classic book i am  willing to bet my son will STILL be able to recite from memory when
he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I prefer bolder taste.... imag
ine my surprise when I ordered 2 boxes - both were expired! One expired back in 2005 for gosh sakes.  I admit that Amaz
on agreed to credit me for cost plus part of shipping, but geez, 2 years expired!!!  I'm hoping to find local San Diego
area shoppe that carries pods so that I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing I do not think belon
gs in it is Canola oil. Canola or rapeseed is not someting a dog would ever find in nature and if it did find rapeseed
in nature and eat it, it would poison them. Today's Food industries have convinced the masses that Canola oil is a safe
and even better oil than olive or virgin coconut, facts though say otherwise. Until the late 70's it was poisonous unti
l they figured out a way to fix that. I still like it but it could be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of this product.<br /><br />Thi
ck, delicious.  Perfect.  3 ingredients: Water, Maltitol, Natural Maple Flavor.  PERIOD.  No chemicals.  No garbage.<br
/><br />Have numerous friends & family members hooked on this stuff.  My husband & son, who do NOT like "sugar free" pr
efer this over major label regular syrup.<br /><br />I use this as my SWEETENER in baking: cheesecakes, white brownies,
muffins, pumpkin pies, etc... Unbelievably delicious...<br /><br />Can you tell I like it? :)
==================================================
```

In [15]:
```
# 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)
```

```
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of this product.<br /><br />Thi
ck, delicious.  Perfect.  3 ingredients: Water, Maltitol, Natural Maple Flavor.  PERIOD.  No chemicals.  No garbage.<br
/><br />Have numerous friends & family members hooked on this stuff.  My husband & son, who do NOT like "sugar free" pr
efer this over major label regular syrup.<br /><br />I use this as my SWEETENER in baking: cheesecakes, white brownies,
muffins, pumpkin pies, etc... Unbelievably delicious...<br /><br />Can you tell I like it? :)
```

In [16]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-element
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()
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along and he always can sing
the refrain. he's learned about whales, India, drooping roses:  i love all the new words this book  introduces and the
silliness of it all.  this is a classic book i am  willing to bet my son will STILL be able to recite from memory when
he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I prefer bolder taste.... imag
ine my surprise when I ordered 2 boxes - both were expired! One expired back in 2005 for gosh sakes.  I admit that Amaz
on agreed to credit me for cost plus part of shipping, but geez, 2 years expired!!!  I'm hoping to find local San Diego
area shoppe that carries pods so that I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing I do not think belon
gs in it is Canola oil. Canola or rapeseed is not someting a dog would ever find in nature and if it did find rapeseed
in nature and eat it, it would poison them. Today's Food industries have convinced the masses that Canola oil is a safe
and even better oil than olive or virgin coconut, facts though say otherwise. Until the late 70's it was poisonous unti
l they figured out a way to fix that. I still like it but it could be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of this product.Thick, deliciou
s.  Perfect.  3 ingredients: Water, Maltitol, Natural Maple Flavor.  PERIOD.  No chemicals.  No garbage.Have numerous f
riends & family members hooked on this stuff.  My husband & son, who do NOT like "sugar free" prefer this over major la
bel regular syrup.I use this as my SWEETENER in baking: cheesecakes, white brownies, muffins, pumpkin pies, etc... Unbe
lievably delicious...Can you tell I like it? :)
```

In [17]:
```
# 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)
```

In [18]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
```

```
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing I do not think belon
gs in it is Canola oil. Canola or rapeseed is not someting a dog would ever find in nature and if it did find rapeseed
in nature and eat it, it would poison them. Today is Food industries have convinced the masses that Canola oil is a saf
e and even better oil than olive or virgin coconut, facts though say otherwise. Until the late 70 is it was poisonous u
ntil they figured out a way to fix that. I still like it but it could be better.
==================================================
```

In [19]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_1500 = re.sub("\S*\d\S*", "", sent_1500).strip()
```

```
Great ingredients although, chicken should have been  rather than chicken broth, the only thing I do not think belongs
in it is Canola oil. Canola or rapeseed is not someting a dog would ever find in nature and if it did find rapeseed in
nature and eat it, it would poison them. Today is Food industries have convinced the masses that Canola oil is a safe a
nd even better oil than olive or virgin coconut, facts though say otherwise. Until the late  is it was poisonous until
they figured out a way to fix that. I still like it but it could be better.
```

In [20]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_4900 = re.sub('[^A-Za-z0-9]+', ' ', sent_4900)
```

```
Can t do sugar Have tried scores of SF Syrups NONE of them can touch the excellence of this product br br Thick delicio
us Perfect 3 ingredients Water Maltitol Natural Maple Flavor PERIOD No chemicals No garbage br br Have numerous friends
family members hooked on this stuff My husband son who do NOT like sugar free prefer this over major label regular syru
p br br I use this as my SWEETENER in baking cheesecakes white brownies muffins pumpkin pies etc Unbelievably delicious
br br Can you tell I like it
```

In [21]:
```
# https://gist.github.com/sebleier/554280
# removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, "br" is present in reviews
# 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', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't",
            'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:
```
# Combining all the above preprocessing steps
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 stopwords)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 364171/364171 [02:25<00:00, 2498.04it/s]
```

In [23]:

Out[23]:
```
'ca not sugar tried scores sf syrups none touch excellence product thick delicious perfect ingredients water maltitol n
atural maple flavor period no chemicals no garbage numerous friends family members hooked stuff husband son not like su
gar free prefer major label regular syrup use sweetener baking cheesecakes white brownies muffins pumpkin pies etc unbe
lievably delicious tell like'
```

[4] Featurization

[4.1] BAG OF WORDS

Reference :

1. https://en.wikipedia.org/wiki/Bag-of-words_model#Example_implementation (https://en.wikipedia.org/wiki/Bag-of-words_model#Example_implementation)
2. http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html (http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html)

[4.2] Bi-Grams and n-Grams.

Reference : https://en.wikipedia.org/wiki/Bag-of-words_model#n-gram_model (https://en.wikipedia.org/wiki/Bag-of-words_model#n-gram_model)

In [24]:
```
#bi-gram
#removing stop words like "not" should be avoided before building n-grams
#CountVectorizer documentation http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVecto
from sklearn.preprocessing import StandardScaler
count_vect = CountVectorizer(ngram_range=(1,2), min_df=20)
standardizer = StandardScaler(with_mean=False)
bigrams = standardizer.fit_transform(count_vect.fit_transform(preprocessed_reviews))
print("some feature names ", count_vect.get_feature_names()[489:499])
print('='*50)

print("the type of count vectorizer ",type(bigrams))
print("the shape of out text BOW vectorizer for Train set ",bigrams.get_shape())
print("the number of unique words including both unigrams and bigrams in Train set ", bigrams.get_shape()[1])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtyp
e int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtyp
e int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)

some feature names  ['adaptable', 'adapted', 'adapter', 'adapts', 'add', 'add additional', 'add agave', 'add almond', '
add almost', 'add also']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer for Train set  (364171, 99325)
the number of unique words including both unigrams and bigrams in Train set  99325
```

[4.3] TF-IDF

Reference :

1. https://en.wikipedia.org/wiki/Tf%E2%80%93idf#Definition (https://en.wikipedia.org/wiki/Tf%E2%80%93idf#Definition)
2. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)

In [25]:
```
tfidf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=20)
standardizer = StandardScaler(with_mean=False)
tfidf_bigrams = standardizer.fit_transform(tfidf_vect.fit_transform(preprocessed_reviews))
print("some feature names ", tfidf_vect.get_feature_names()[5000:5010])
print('='*50)

print("the type of count vectorizer ",type(tfidf_bigrams))
print("the shape of out text Tfidf vectorizer for Train set ",tfidf_bigrams.get_shape())
```

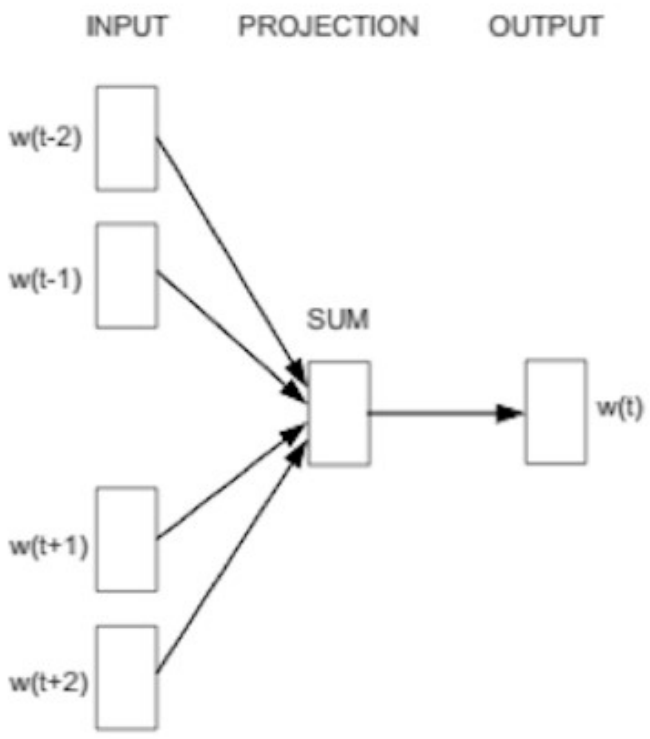
```
some feature names  ['baby baby', 'baby back', 'baby born', 'baby bottles', 'baby boy', 'baby brezza', 'baby carrots',
'baby cereal', 'baby could', 'baby eat']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text Tfidf vectorizer for Train set  (364171, 99325)
the number of unique words including both unigrams and bigrams in Train set  99325
```

[4.4] Word2Vec

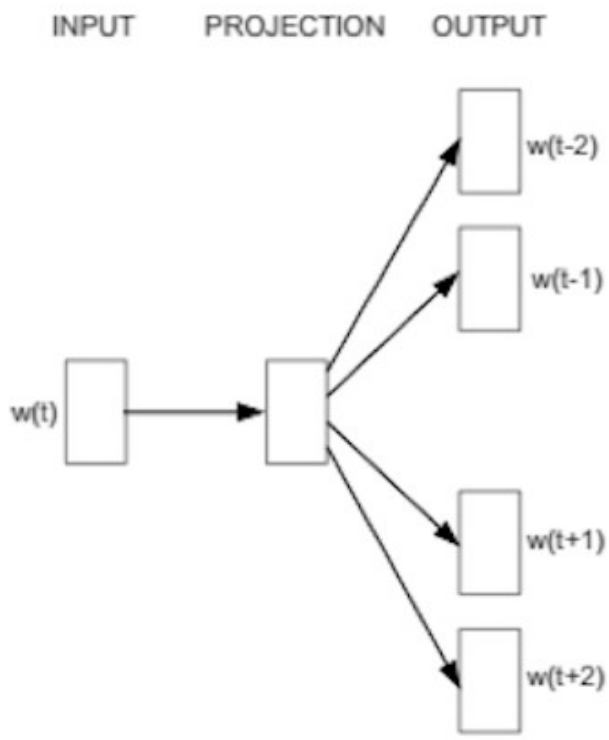
In [26]:
```
# Train our own Word2Vec model using preprocessed reviews
sentancesList=[]
for eachSentance in preprocessed_reviews:
```

Reference:

1. https://towardsdatascience.com/a-beginners-guide-to-word-embedding-with-gensim-word2vec-model-5970fa56cc92 (https://towardsdatascience.com/a-beginners-guide-to-word-embedding-with-gensim-word2vec-model-5970fa56cc92)
2. https://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/ (https://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/)



CBOW



Skip-gram

In [27]:
```
# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
w2v_model=Word2Vec(sentancesList,min_count=5,size=100, workers=-1)
print(w2v_model.wv.most_similar('tasty'))
print('='*50)
```

```
[('stewed', 0.38705143332481384), ('everthing', 0.3821856379508972), ('soured', 0.3590002954006195), ('elated', 0.35897
16851711273), ('gordita', 0.3567555546760559), ('teisseire', 0.3557489812374115), ('planted', 0.34962496161460876), ('c
hx', 0.34796518087387085), ('amendment', 0.3421073853969574), ('apparent', 0.3417196273803711)]
==================================================
[('karavitis', 0.40957337617874146), ('typing', 0.37931838631629944), ('jeez', 0.3723851442337036), ('heirloom', 0.3708
122968673706), ('earthy', 0.36364302039146423), ('pinky', 0.3628316819667816), ('carbohydrates', 0.3568650484085083),
('seized', 0.35214871168136597), ('windex', 0.35021790862083435), ('show', 0.3472427427768707)]
```

In [28]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
```

```
number of words that occured minimum 5 times  33573
sample words :  ['ordering', 'regularly', 'thses', 'glad', 'online', 'thanks', 'account', 'couple', 'needs', 'value']
```

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

[4.4.1.1] Avg W2v

In [29]:
```
# average Word2Vec
# computing average word2vec for each review.
Word2Vectors = [] # the avg-w2v for each train sentence/review is stored in this list
for eachSentance in tqdm(sentancesList):
    sentanceVector = np.zeros(100) # as word vectors are of zero length 50
    validWordCounts =0 # num of words with a valid vector in the sentence/review
    for eachWord in eachSentance:
        if eachWord in w2v_words:
            vector = w2v_model.wv[eachWord]
            sentanceVector += vector
            validWordCounts += 1
    if validWordCounts != 0:
        sentanceVector /= validWordCounts
    Word2Vectors.append(sentanceVector)
standardizer = StandardScaler()
Word2Vectors = standardizer.fit_transform(Word2Vectors)
print(len(Word2Vectors))
```

```
100%|[redacted]| 364171/364171 [25:45<00:00, 235.63it/s]

364171
100
```

[4.4.1.2] TFIDF weighted W2v

In [30]:
```
tfidfW2VModel = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
tfidfW2VModelVectors = tfidfW2VModel.fit_transform(preprocessed_reviews[:50000])
# creating hashmap with word as key and inverse document frequency as value
```

In [31]:
```
# TF-IDF weighted Word2Vec
tfidfWords = tfidfW2VModel.get_feature_names() # tfidf words

TfidfWord2Vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
for eachSentance in tqdm(sentancesList):
    sentanceVector = np.zeros(100) # as word vectors are of zero length
    weightedSum =0; # num of words with a valid vector in the sentence/review
    for eachWord in eachSentance:
        if eachWord in w2v_words and eachWord in tfidfWords:
            vector = w2v_model.wv[eachWord]
            tf_idf = wordsHashMap[eachWord]*(eachSentance.count(eachWord)/len(eachSentance))
            sentanceVector += (vector * tf_idf)
            weightedSum += tf_idf
    if weightedSum != 0:
        sentanceVector /= weightedSum
    TfidfWord2Vectors.append(sentanceVector)

standardizer = StandardScaler()
TfidfWord2Vectors = standardizer.fit_transform(TfidfWord2Vectors)
print(len(TfidfWord2Vectors))
```

```
100%|██████████████████████████████████████████████████████████████████████| 364171/364171 [47:38<00:00, 127.42it/s]

364171
100
```

[5.1] K-Means Clustering

The ***KMeans*** algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares (see below). This algorithm requires the number of clusters to be specified. It scales well to large number of samples and has been used across a large range of application areas in many different fields.

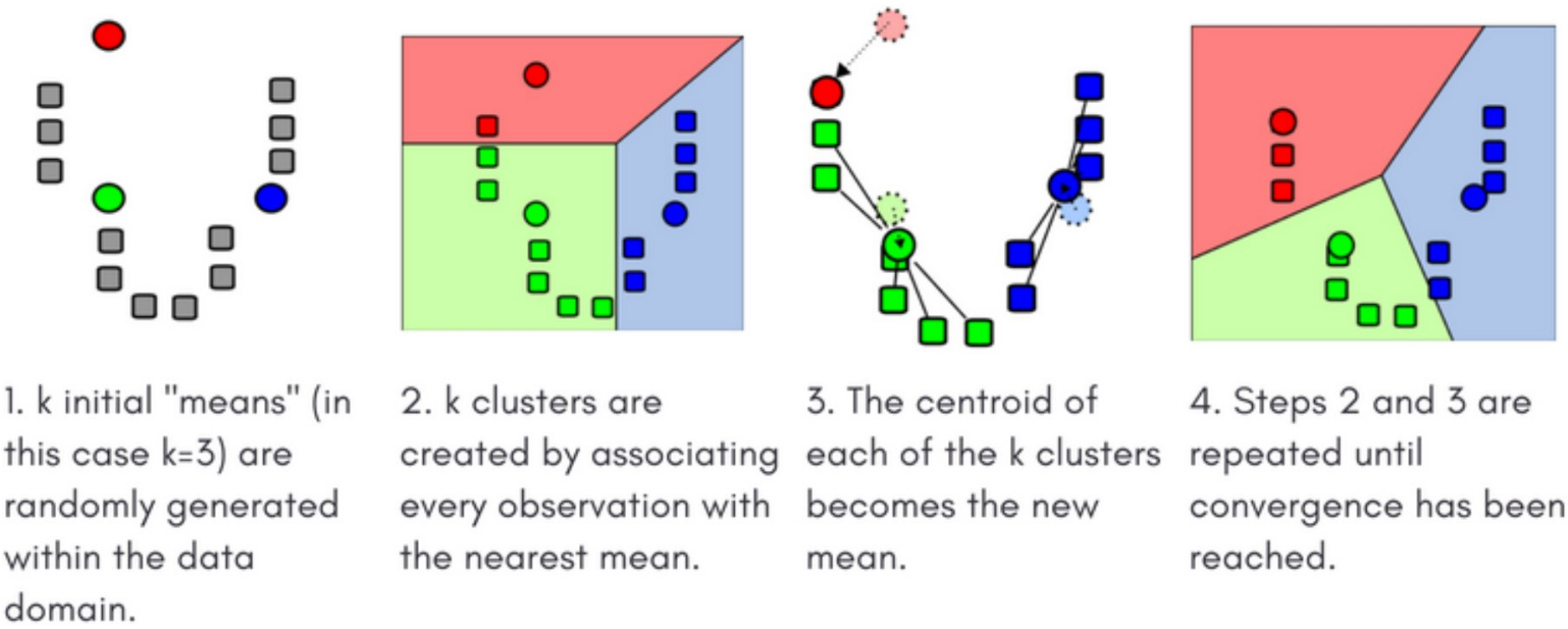
The k-means algorithm divides a set of N samples X into K disjoint clusters , each described by the mean of the samples in the cluster. The means are commonly called the cluster “centroids”; note that they are not, in general, points from X, although they live in the same space.

The K-means algorithm aims to choose centroids that minimise the **inertia, or within-cluster sum-of-squares criterion**:

$$J := \underset{c_1,c_2,\ldots,c_k}{\arg\min} \sum_{j=1}^{k} \sum_{i=1}^{n} ||x_i^{(j)} - c_j||^2.$$

where,

- J - Objective Function
- k - number of clusters
- n - number of cases
- x_i - case i
- c_j - centroid for cluster j
- $||x_i^j - c_j||^2$ - distance function



K-means is often referred to as Lloyd’s algorithm. In basic terms, the algorithm has three steps. The first step chooses the initial centroids, with the most basic method being to choose K samples from the dataset X. After initialization, K-means consists of looping between the two other steps. The first step assigns each sample to its nearest centroid. The second step creates new centroids by taking the mean value of all of the samples assigned to each previous centroid. The difference between the old and the new centroids are computed and the algorithm repeats these last two steps until this value is less than a threshold. In other words, it repeats until the centroids do not move significantly.

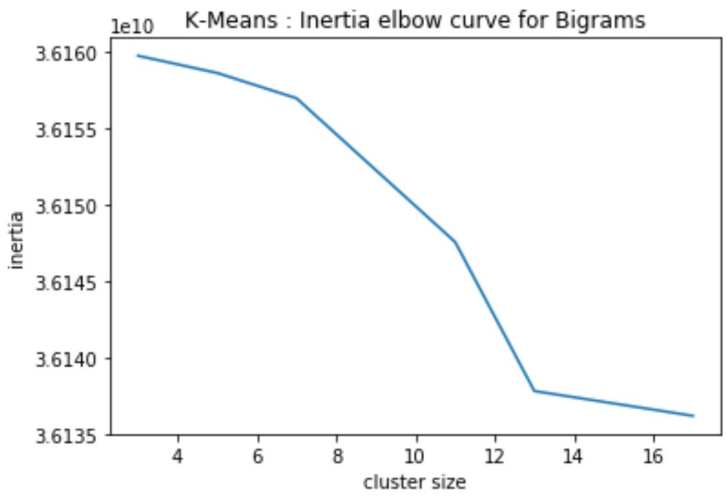
Source: Scikit-Learn

[5.1.1] Applying K-Means Clustering on BOW

In [32]:
```
import time
from sklearn.cluster import KMeans
range_n_clusters = [3, 5, 7, 11, 13, 17]
bigrams_inertia = []
for n_clusters in range_n_clusters:
    bigrams_clusterer = KMeans(n_clusters=n_clusters, random_state=10, n_jobs=-1)
    start = time.time()
    bigrams_clusterer.fit(bigrams)
    timeTaken = time.time() - start
    bigrams_inertia.append(np.round(bigrams_clusterer.inertia_,2))
    print("inertia for cluster size {:d} is :{:0.2f} and time taken:{:0.2f}mins".
          format(n_clusters,bigrams_clusterer.inertia_,(timeTaken/60)))
plt.plot(range_n_clusters,bigrams_inertia)
plt.xlabel("cluster size")
plt.ylabel("inertia")
```

```
inertia for cluster size 3 is :36159696393.68 and time taken:15.79mins
inertia for cluster size 5 is :36158567785.55 and time taken:14.54mins
inertia for cluster size 7 is :36156915825.98 and time taken:39.13mins
inertia for cluster size 11 is :36147528372.94 and time taken:22.45mins
inertia for cluster size 13 is :36137781197.30 and time taken:35.17mins
inertia for cluster size 17 is :36136162865.90 and time taken:28.77mins
```

Out[32]:
```
Text(0.5, 1.0, 'K-Means : Inertia elbow curve for Bigrams')
```



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

In [33]:
```
#choosing cluster size as 13, deciphered from above graph
bigrams_clusterer = KMeans(n_clusters=13, random_state=10, n_jobs=-1)
```

Out[33]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=13, n_init=10, n_jobs=-1, precompute_distances='auto',
    random_state=10, tol=0.0001, verbose=0)
```

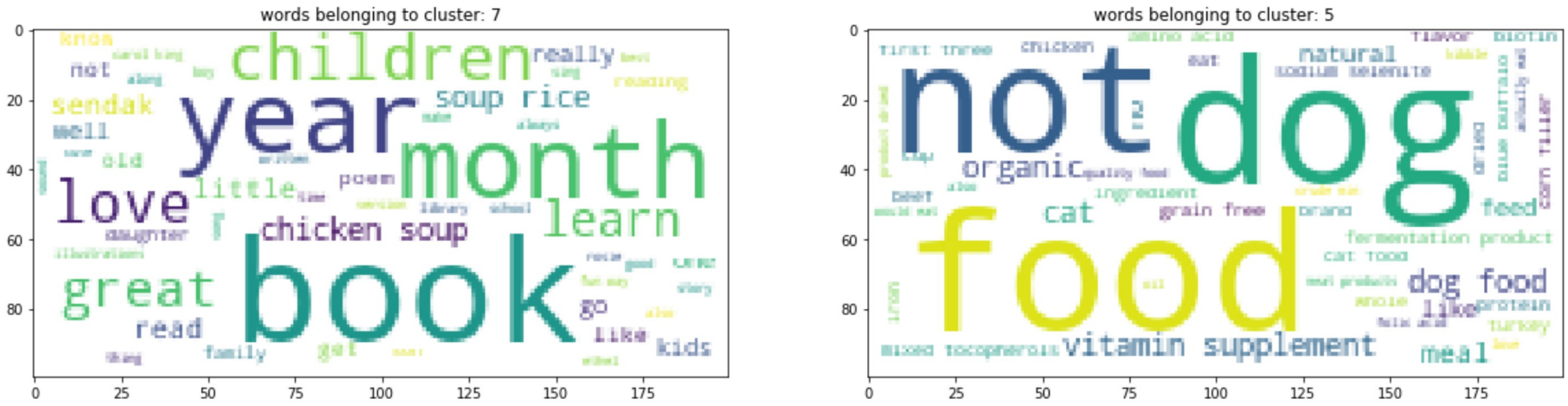
In [34]:
```
print("number of reviews in each cluster")
bigram_df = pd.DataFrame(data={"text":preprocessed_reviews,"labels":bigrams_clusterer.labels_})
```

```
number of reviews in each cluster
```

Out[34]:
```
7     363912
5         84
6         60
4         41
3         41
11        23
12         2
9          2
1          2
10         1
8          1
2          1
0          1
Name: labels, dtype: int64
```

In [108]:
```
import wordcloud
import random
from IPython.display import Markdown, display
display(Markdown("#       displaying only 6 clusters with random 20 review words"))
fig, axs = plt.subplots(3,2, figsize=(20,20))
cluster_list = [7,5,6,4,3,11]
for ax,eachCluster in zip(axs.ravel(),cluster_list): #https://stackoverflow.com/a/17211410/12901493
    wordList = []
    #https://docs.python.org/3/library/random.html#random.sample
    for eachReview in random.sample(bigram_df[bigram_df.labels == eachCluster].text[:20].to_list(),20):
        for eachWord in eachReview.split(' '):
            wordList.append(eachWord)
    wordClud = wordcloud.WordCloud(width = 200, height = 100,
                                   background_color ='white',
                                   stopwords = stopwords).generate(' '.join(wordList))
    ax.imshow(wordClud)
    ax.set_title("words belonging to cluster: {:d}".format(eachCluster))
```

```
<IPython.core.display.Markdown object>
```

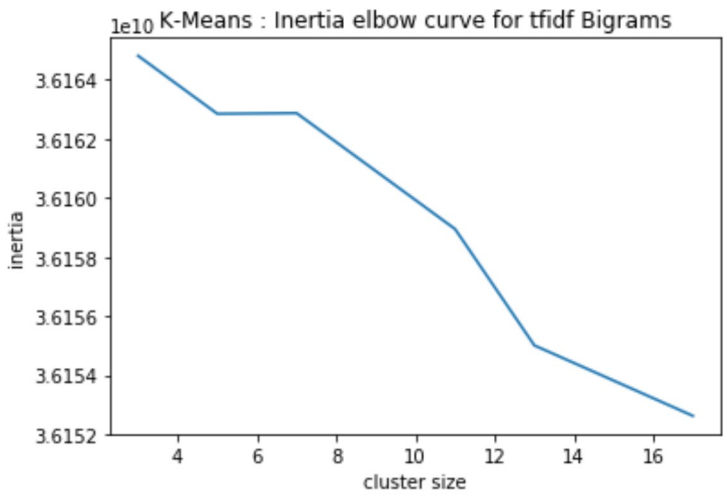


[5.1.3] Applying K-Means Clustering on TFIDF, SET 2

In [36]:
```
range_n_clusters = [3, 5, 7, 11, 13, 17]
tfidf_inertia = []
for n_clusters in range_n_clusters:
    tfidf_clusterer = KMeans(n_clusters=n_clusters, random_state=10, n_jobs=-1)
    start = time.time()
    tfidf_clusterer.fit(tfidf_bigrams)
    timeTaken = time.time() - start
    tfidf_inertia.append(np.round(tfidf_clusterer.inertia_,2))
    print("inertia for cluster size {:d} is :{:0.2f} and time taken:{:0.2f}mins".
          format(n_clusters,tfidf_clusterer.inertia_,(timeTaken/60)))
plt.plot(range_n_clusters, tfidf_inertia)
plt.xlabel("cluster size")
plt.ylabel("inertia")
```

```
inertia for cluster size 3 is :36164813148.31 and time taken:20.34mins
inertia for cluster size 5 is :36162848471.42 and time taken:35.99mins
inertia for cluster size 7 is :36162871404.09 and time taken:27.96mins
inertia for cluster size 11 is :36158944493.58 and time taken:34.01mins
inertia for cluster size 13 is :36155005157.84 and time taken:45.92mins
inertia for cluster size 17 is :36152618832.49 and time taken:48.52mins
```

Out[36]:
```
Text(0.5, 1.0, 'K-Means : Inertia elbow curve for tfidf Bigrams')
```



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

In [37]:
```
#choosing cluster size as 5, deciphered from above graph
tfidf_clusterer = KMeans(n_clusters=5, random_state=10, n_jobs=-1)
```

Out[37]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=5, n_init=10, n_jobs=-1, precompute_distances='auto',
    random_state=10, tol=0.0001, verbose=0)
```

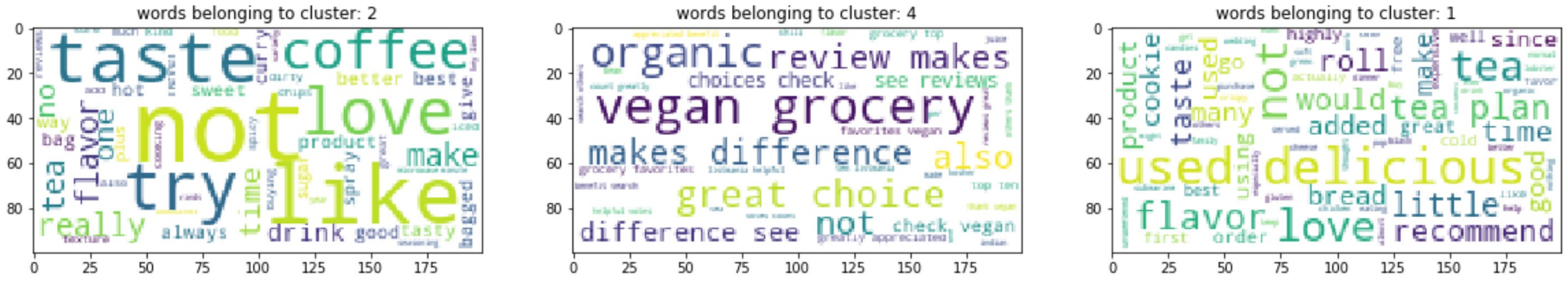
In [38]:
```
print("number of reviews in each cluster")
tfidf_df = pd.DataFrame(data={"text":preprocessed_reviews,"labels":tfidf_clusterer.labels_})
```

```
number of reviews in each cluster
```

Out[38]:
```
2    364083
4        63
1        23
3         1
0         1
Name: labels, dtype: int64
```

In [107]:
```
display(Markdown("#       displaying only 3 clusters with random 20 review words"))
fig, axs = plt.subplots(1,3, figsize=(20,10))
cluster_list = [2,4,1]
for ax,eachCluster in zip(axs.ravel(),cluster_list):#https://stackoverflow.com/a/17211410/12901493
    wordList = []
    #https://docs.python.org/3/library/random.html#random.sample
    for eachReview in random.sample(tfidf_df[tfidf_df.labels == eachCluster].text.to_list(),20):
        for eachWord in eachReview.split(' '):
            wordList.append(eachWord)
    wordClud = wordcloud.WordCloud(width = 200, height = 100,
                                   background_color ='white',
                                   stopwords = stopwords).generate(' '.join(wordList))
    ax.imshow(wordClud)
    ax.set_title("words belonging to cluster: {:d}".format(eachCluster))
```

```
<IPython.core.display.Markdown object>
```

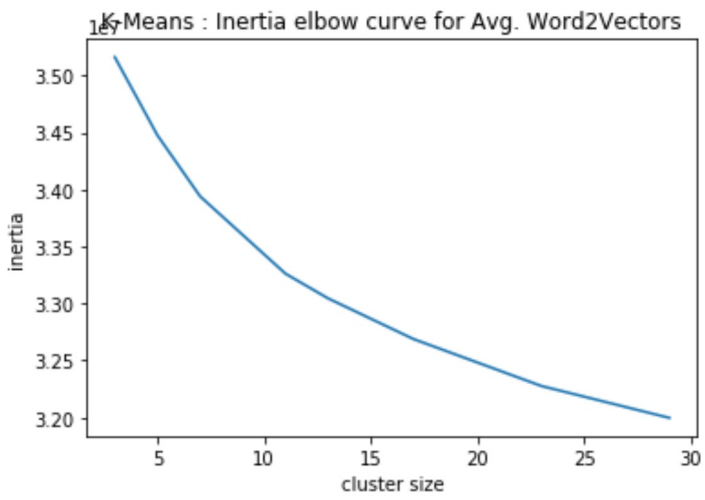


[5.1.5] Applying K-Means Clustering on AVG W2V, SET 3

In [40]:
```
range_n_clusters = [3, 5, 7, 11, 13, 17, 23, 29]
avgw2v_inertia = []
for n_clusters in range_n_clusters:
    avgw2v_clusterer = KMeans(n_clusters=n_clusters, random_state=10, n_jobs=-1)
    start = time.time()
    avgw2v_clusterer.fit(Word2Vectors)
    timeTaken = time.time() - start
    avgw2v_inertia.append(np.round(avgw2v_clusterer.inertia_,2))
    print("inertia for cluster size {:d} is :{:0.2f} and time taken:{:0.2f}mins".
          format(n_clusters,avgw2v_clusterer.inertia_,(timeTaken/60)))
plt.plot(range_n_clusters,avgw2v_inertia)
plt.xlabel("cluster size")
plt.ylabel("inertia")
```

```
inertia for cluster size 3 is :35160724.50 and time taken:0.57mins
inertia for cluster size 5 is :34474762.23 and time taken:1.94mins
inertia for cluster size 7 is :33939578.80 and time taken:0.93mins
inertia for cluster size 11 is :33261689.09 and time taken:1.27mins
inertia for cluster size 13 is :33044577.74 and time taken:2.31mins
inertia for cluster size 17 is :32688094.69 and time taken:2.71mins
inertia for cluster size 23 is :32276209.05 and time taken:3.73mins
inertia for cluster size 29 is :31996361.79 and time taken:4.21mins
```

Out[40]:
```
Text(0.5, 1.0, 'K-Means : Inertia elbow curve for Avg. Word2Vectors')
```



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

In [41]:
```
#choosing cluster size as 7, deciphered from above graph
avgw2v_clusterer = KMeans(n_clusters=7, random_state=10, n_jobs=-1)
```

Out[41]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=7, n_init=10, n_jobs=-1, precompute_distances='auto',
    random_state=10, tol=0.0001, verbose=0)
```

In [42]:
```
print("number of reviews in each cluster")
avgw2v_df = pd.DataFrame(data={"text":preprocessed_reviews,"labels":avgw2v_clusterer.labels_})
```

```
number of reviews in each cluster
```

Out[42]:
```
4    138682
6     66367
0     58440
1     31445
3     28290
5     22167
2     18780
Name: labels, dtype: int64
```

In [106]:
```
display(Markdown("#       displaying only 7 clusters with random 20 review words"))
fig, axs = plt.subplots(4,2, figsize=(20,20))
cluster_list = [4,5,3,0,1,2,6]
for ax,eachCluster in zip(axs.ravel(),cluster_list):#https://stackoverflow.com/a/17211410/12901493
    wordList = []
    #https://docs.python.org/3/library/random.html#random.sample
    for eachReview in random.sample(avgw2v_df[avgw2v_df.labels == eachCluster].text.to_list(),20):
        for eachWord in eachReview.split(' '):
            wordList.append(eachWord)
    wordClud = wordcloud.WordCloud(width = 200, height = 100,
                                   background_color ='white',
                                   stopwords = stopwords).generate(' '.join(wordList))
    ax.imshow(wordClud)
    ax.set_title("words belonging to cluster: {:d}".format(eachCluster))
axs[3,1].remove() #https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.axes.Axes.remove.html
```

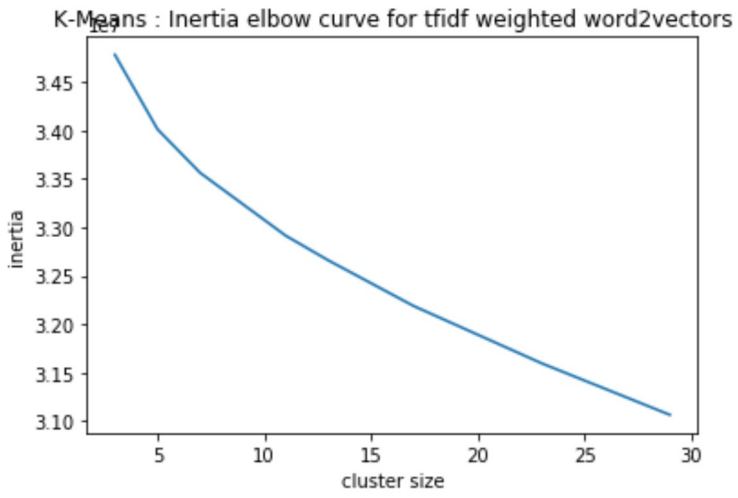
```
<IPython.core.display.Markdown object>
```

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

In [44]:
```
range_n_clusters = [3, 5, 7, 11, 13, 17, 23, 29]
tfidfavgw2v_inertia = []
for n_clusters in range_n_clusters:
    tfidfavgw2v_clusterer = KMeans(n_clusters=n_clusters, random_state=10, n_jobs=-1)
    start = time.time()
    tfidfavgw2v_clusterer.fit(TfidfWord2Vectors)
    timeTaken = time.time() - start
    tfidfavgw2v_inertia.append(np.round(tfidfavgw2v_clusterer.inertia_,2))
    print("inertia for cluster size {:d} is :{:0.2f} and time taken:{:0.2f}mins".
          format(n_clusters,tfidfavgw2v_clusterer.inertia_,(timeTaken/60)))
plt.plot(range_n_clusters,tfidfavgw2v_inertia)
plt.xlabel("cluster size")
plt.ylabel("inertia")
```

```
inertia for cluster size 3 is :34780447.63 and time taken:0.46mins
inertia for cluster size 5 is :34010507.24 and time taken:0.95mins
inertia for cluster size 7 is :33560667.18 and time taken:1.40mins
inertia for cluster size 11 is :32913193.57 and time taken:2.64mins
inertia for cluster size 13 is :32657775.94 and time taken:2.27mins
inertia for cluster size 17 is :32186352.40 and time taken:2.89mins
inertia for cluster size 23 is :31594360.01 and time taken:3.36mins
inertia for cluster size 29 is :31063158.24 and time taken:4.49mins
```

Out[44]:
```
Text(0.5, 1.0, 'K-Means : Inertia elbow curve for tfidf weighted word2vectors')
```



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

In [45]:
```
#choosing cluster size as 5, deciphered from above graph
tfidfavgw2v_clusterer = KMeans(n_clusters=5, random_state=10, n_jobs=-1)
```

Out[45]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=5, n_init=10, n_jobs=-1, precompute_distances='auto',
    random_state=10, tol=0.0001, verbose=0)
```

In [46]:
```
print("number of reviews in each cluster")
tfidfavgw2v_df = pd.DataFrame(data={"text":preprocessed_reviews,"labels":tfidfavgw2v_clusterer.labels_})
```

```
number of reviews in each cluster
```

Out[46]:
```
4    149927
0    104530
2     84559
3     13378
1     11777
Name: labels, dtype: int64
```

In [105]:
```
display(Markdown("#       displaying only 5 clusters with random 20 review words"))
fig, axs = plt.subplots(3,2, figsize=(20,20))
cluster_list = [1,2,0,3,2]
for ax,eachCluster in zip(axs.ravel(),cluster_list):#https://stackoverflow.com/a/17211410/12901493
    wordList = []
    #https://docs.python.org/3/library/random.html#random.sample
    for eachReview in random.sample(tfidfavgw2v_df[tfidfavgw2v_df.labels == eachCluster].text.to_list(),20):
        for eachWord in eachReview.split(' '):
            wordList.append(eachWord)
    wordClud = wordcloud.WordCloud(width = 200, height = 100,
                                   background_color ='white',
                                   stopwords = stopwords).generate(' '.join(wordList))
    ax.imshow(wordClud)
    ax.set_title("words belonging to cluster: {:d}".format(eachCluster))
axs[2,1].remove() #https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.axes.Axes.remove.html
```

```
<IPython.core.display.Markdown object>
```

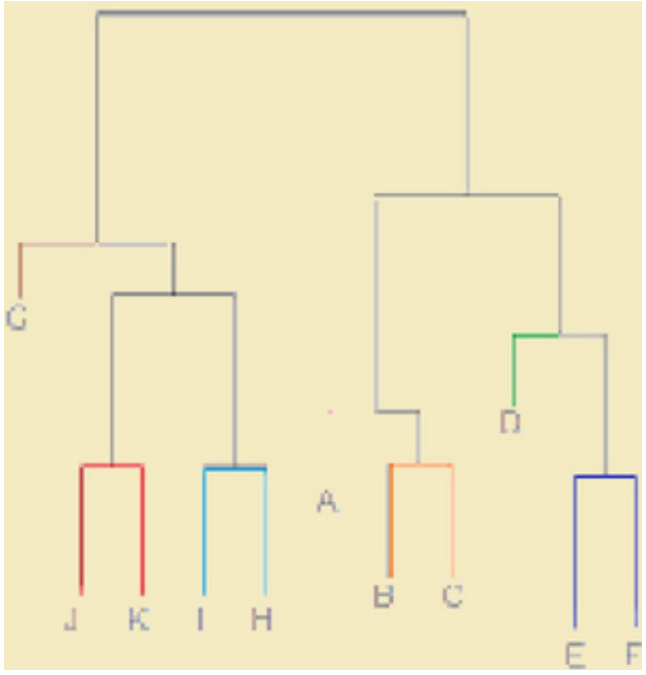
[5.2] Agglomerative Clustering

Hierarchical clustering is a general family of clustering algorithms that build nested clusters by merging or splitting them successively. This hierarchy of clusters is represented as a tree (or dendrogram). The root of the tree is the unique cluster that gathers all the samples, the leaves being the clusters with only one sample.

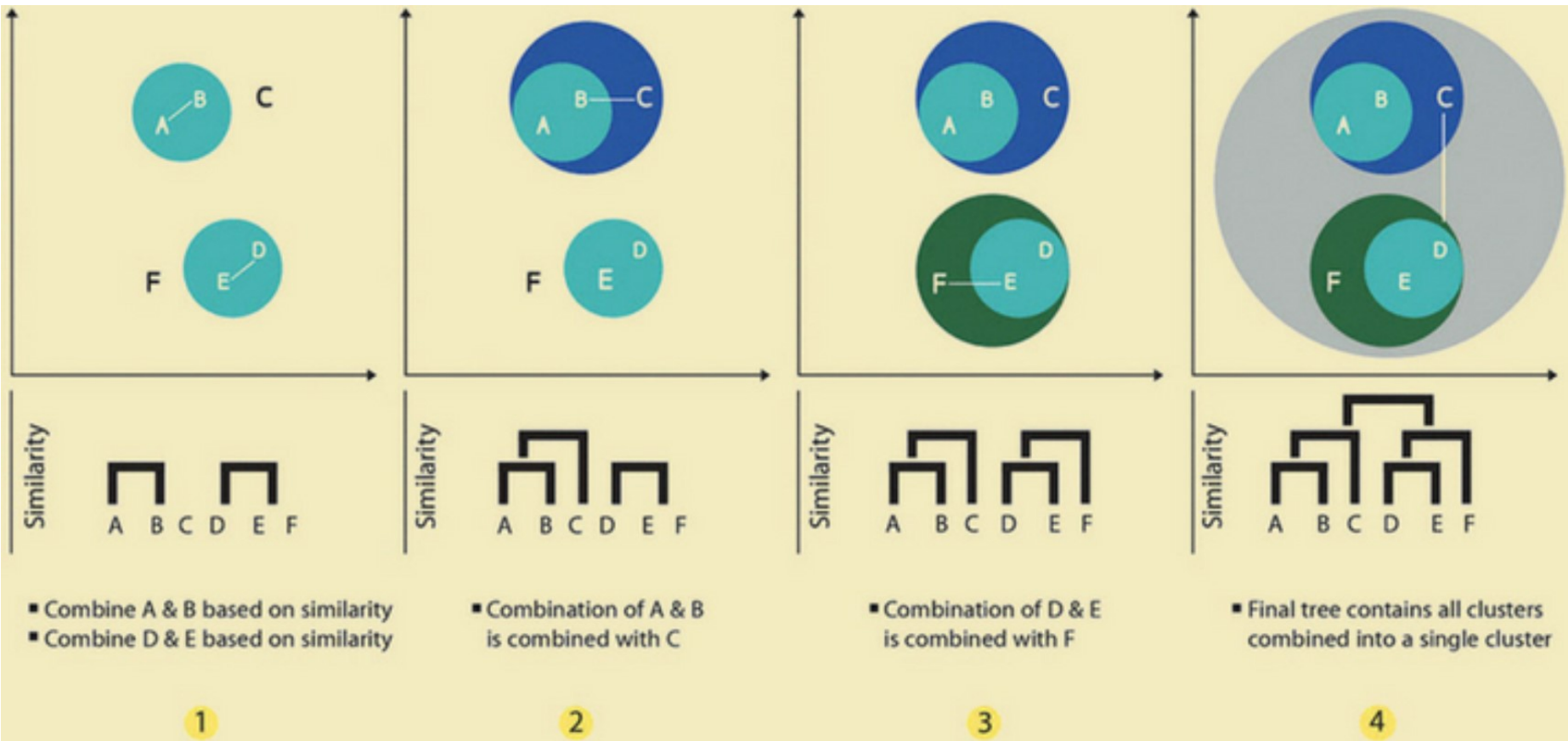
The ***AgglomerativeClustering*** object performs a hierarchical clustering using a bottom up approach: each observation starts in its own cluster, and clusters are successively merged together. The linkage criteria determines the metric used for the merge strategy:

- Ward minimizes the sum of squared differences within all clusters. It is a variance-minimizing approach and in this sense is similar to the k-means objective function but tackled with an agglomerative hierarchical approach.
- Maximum or complete linkage minimizes the maximum distance between observations of pairs of clusters.
- Average linkage minimizes the average of the distances between all observations of pairs of clusters.
- Single linkage minimizes the distance between the closest observations of pairs of clusters.

AgglomerativeClustering can also scale to large number of samples when it is used jointly with a connectivity matrix, but is computationally expensive when no connectivity constraints are added between samples: it considers at each step all the possible merges.



Dendogram the root of the tree is the unique cluster that gathers all the samples. The leaves being the clusters with only one sample. The decision of the no. of clusters that can best depict different groups can be chosen by obeserving the dendogram. The best chioce of the no. of clusters is the no. of vertical lines in the dendogram cut by a horizontal line that can transverse the maximum distance vertically without intersecting a cluster



source: Scikit-Learn & [Avik-Jain Github (https://github.com/Avik-Jain/100-Days-Of-ML-Code#hierarchical-clustering--day-54)](https://github.com/Avik-Jain/100-Days-Of-ML-Code#hierarchical-clustering--day-54)

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

In [50]:
```
from sklearn.cluster import AgglomerativeClustering
avgw2v_agglo = AgglomerativeClustering(n_clusters=5,
                                       affinity='euclidean',
                                       memory=None,
                                       connectivity=None,
                                       compute_full_tree='auto',
                                       linkage='ward')
start = time.time()
avgw2v_agglo.fit(Word2Vectors[:50000])
timeTaken = time.time() - start
```

```
time taken for fitting :6.73 mins
```

In [51]:
```
print("number of reviews in each cluster")
avgw2v_agglo_df = pd.DataFrame(data={"text":preprocessed_reviews[:50000],"labels":avgw2v_agglo.labels_})
```

```
number of reviews in each cluster
```

Out[51]:
```
4    29381
2     7984
0     4540
1     4463
3     3632
Name: labels, dtype: int64
```

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

In [104]:
```
display(Markdown("#       displaying only 5 clusters with random 20 review words"))
fig, axs = plt.subplots(3,2, figsize=(20,20))
cluster_list = [4,2,0,1,3]
for ax,eachCluster in zip(axs.ravel(),cluster_list):#https://stackoverflow.com/a/17211410/12901493
    wordList = []
    #https://docs.python.org/3/library/random.html#random.sample
    for eachReview in random.sample(avgw2v_agglo_df[avgw2v_agglo_df.labels == eachCluster].text.to_list(),20):
        for eachWord in eachReview.split(' '):
            wordList.append(eachWord)
    wordClud = wordcloud.WordCloud(width = 200, height = 100,
                                   background_color ='white',
                                   stopwords = stopwords).generate(' '.join(wordList))
    ax.imshow(wordClud)
    ax.set_title("words belonging to cluster: {:d}".format(eachCluster))
axs[2,1].remove() #https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.axes.Axes.remove.html
```

```
<IPython.core.display.Markdown object>
```

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

In [66]:
```
from sklearn.cluster import AgglomerativeClustering
tfidfw2v_agglo = AgglomerativeClustering(n_clusters=5,
                                         affinity='euclidean',
                                         memory=None,
                                         connectivity=None,
                                         compute_full_tree='auto',
                                         linkage='ward')
start = time.time()
tfidfw2v_agglo.fit(TfidfWord2Vectors[100000:150000])
timeTaken = time.time() - start
```

```
time taken for fitting :6.61 mins
```

In [67]:
```
print("number of reviews in each cluster")
tfidfw2v_agglo_df = pd.DataFrame(data={"text":preprocessed_reviews[100000:150000],"labels":tfidfw2v_agglo.labels_})
```

```
number of reviews in each cluster
```

Out[67]:
```
1    43775
0     3394
4     1551
2      927
3      353
Name: labels, dtype: int64
```

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

In [68]:
```
display(Markdown("#       displaying only 5 clusters with random 20 review words"))
fig, axs = plt.subplots(3,2, figsize=(20,20))
cluster_list = [1,0,4,2,3]
for ax,eachCluster in zip(axs.ravel(),cluster_list):#https://stackoverflow.com/a/17211410/12901493
    wordList = []
    #https://docs.python.org/3/library/random.html#random.sample
    for eachReview in random.sample(tfidfw2v_agglo_df[tfidfw2v_agglo_df.labels == eachCluster].text.to_list(),20):
        for eachWord in eachReview.split(' '):
            wordList.append(eachWord)
    wordClud = wordcloud.WordCloud(width = 200, height = 100,
                                   background_color ='white',
                                   stopwords = stopwords).generate(' '.join(wordList))
    ax.imshow(wordClud)
    ax.set_title("words belonging to cluster: {:d}".format(eachCluster))
axs[2,1].remove() #https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.axes.Axes.remove.html
```

```
<IPython.core.display.Markdown object>
```

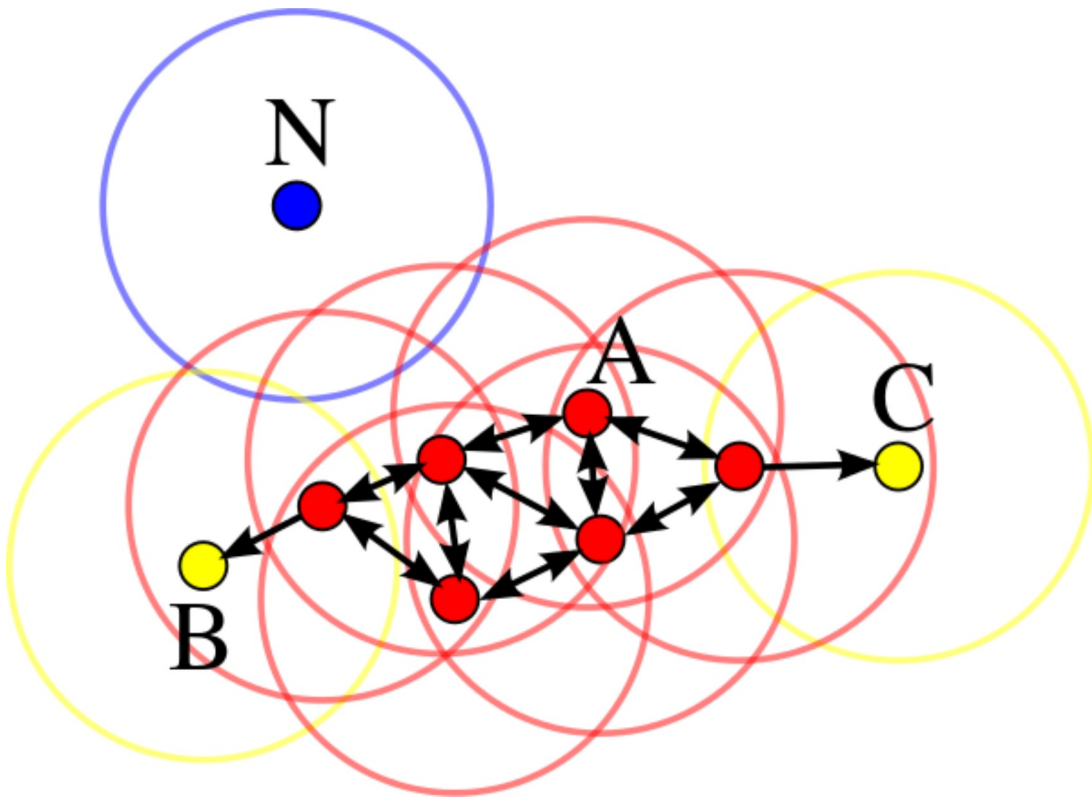
[5.3] DBSCAN Clustering

The **DBSCAN** algorithm views clusters as areas of high density separated by areas of low density. Due to this rather generic view, clusters found by DBSCAN can be any shape, as opposed to k-means which assumes that clusters are convex shaped. The central component to the DBSCAN is the concept of ***core samples***, which are samples that are in areas of high density. A cluster is therefore a set of core samples, each close to each other (measured by some distance measure) and a set of non-core samples that are close to a core sample (but are not themselves core samples). There are two parameters to the algorithm, ***min_samples*** and ε, which define formally what we mean when we say dense. Higher ***min_samples*** or lower ε indicate higher density necessary to form a cluster.

Consider a set of points in some space to be clustered. Let ε be a parameter specifying the radius of a neighborhood with respect to some point. For the purpose of DBSCAN clustering, the points are classified as *core points, (density-)reachable points* and *outliers*, as follows:

- A point p is a core point if at least **minPts** points are within distance ε of it (including p).
- A point q is directly reachable from p if point q is within distance ε from core point p. Points are only said to be directly reachable from core points.
- A point q is reachable from p if there is a path $p_1, \ldots, p_n$ with $p_1 = p$ and $p_n = q$, where each p_{i+1} is directly reachable from p_i. Note that this implies that all points on the path must be core points, with the possible exception of q.
- All points not reachable from any other point are outliers or noise points.

Now if p is a core point, then it forms a *cluster* together with all points (core or non-core) that are reachable from it. Each cluster contains at least one core point; non-core points can be part of a cluster, but they form its "edge", since they cannot be used to reach more points.



Reachability is not a symmetric relation since, by definition, no point may be reachable from a non-core point, regardless of distance (so a non-core point may be reachable, but nothing can be reached from it). Therefore, a further notion of connectedness is needed to formally define the extent of the clusters found by DBSCAN. Two points p and q are density-connected if there is a point o such that both p and q are reachable from o. Density-connectedness is symmetric.

A cluster then satisfies two properties:

1. All points within the cluster are mutually density-connected.
2. If a point is density-reachable from some point of the cluster, it is part of the cluster as well.

Abstract Algorithm

The DBSCAN algorithm can be abstracted into the following steps:

1. Find the points in the ε neighborhood of every point, and identify the core points with more than **minPts** neighbors.
2. Find the connected components of core points on the neighbor graph, ignoring all non-core points.
3. Assign each non-core point to a nearby cluster if the cluster is an ε neighbor, otherwise assign it to noise.

A naive implementation of this requires storing the neighborhoods in step 1, thus requiring substantial memory. The original DBSCAN algorithm does not require this by performing these steps for one point at a time.

Source: Wikipedia

[5.3.1] Applying DBSCAN on AVG W2V, SET 3

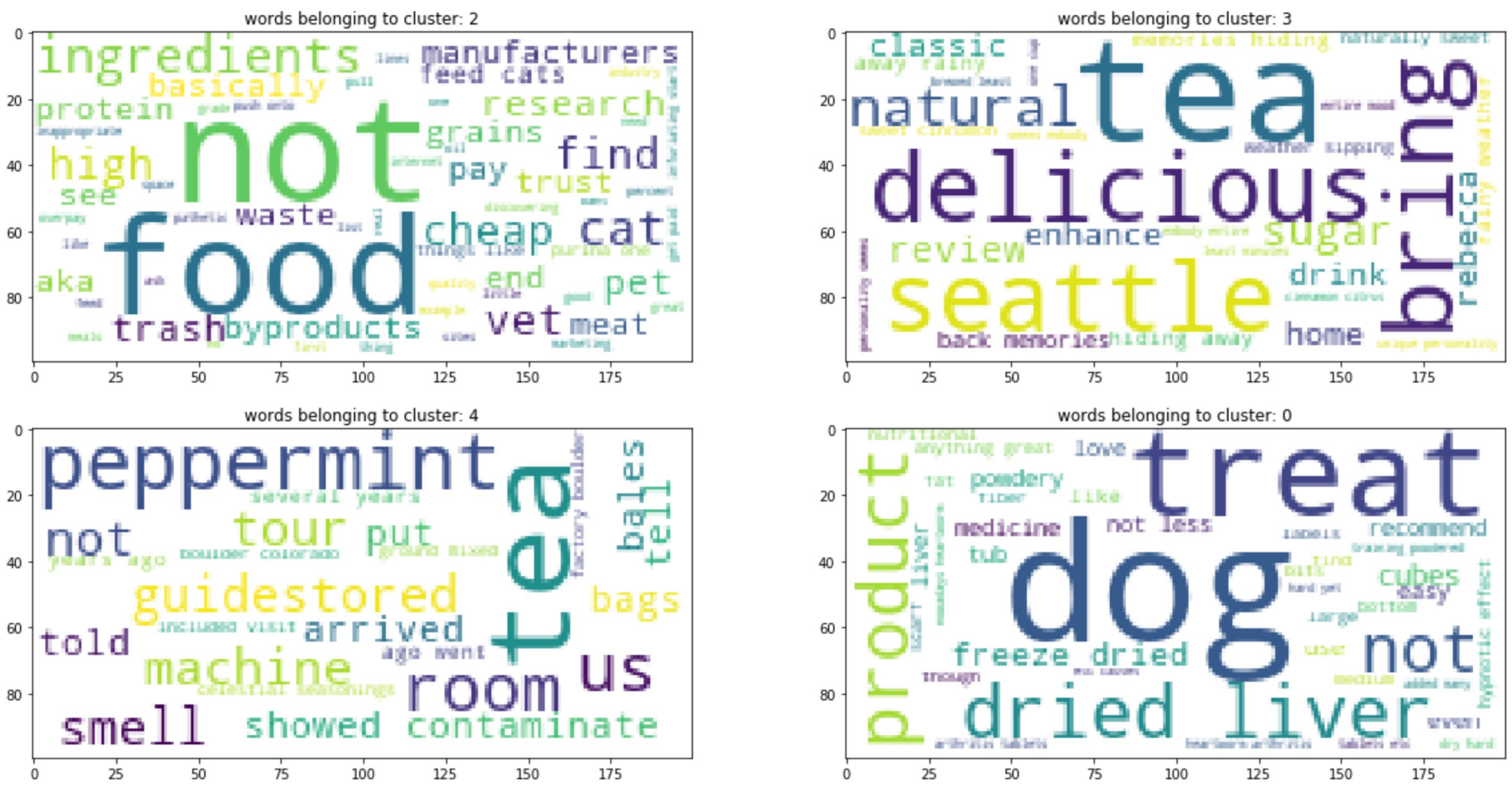
In [79]:
```
from sklearn.cluster import DBSCAN
print("-1 cluster label is outliers")
for eachEpsilon in [0.5,1,2]:
    avgw2v_DBSCAN = DBSCAN(eps=eachEpsilon,
                           min_samples=5,
                           metric='euclidean',
                           metric_params=None,
                           algorithm='auto',
                           leaf_size=30,
                           p=None,
                           n_jobs=-1)
    start = time.time()
    avgw2v_DBSCAN.fit(Word2Vectors[:50000])
    timeTaken = time.time() - start
    print("epsilon - {:.2f}, core samples - {:d}, time taken - {:.2f}mins".
          format(eachEpsilon,len(avgw2v_DBSCAN.core_sample_indices_),(timeTaken/60)))
    print("number of reviews in each cluster")
    avgw2v_DBSCAN_df = pd.DataFrame(data={"text":preprocessed_reviews[:50000],"labels":avgw2v_DBSCAN.labels_})
```

```
-1 cluster label is outliers
epsilon - 0.50, core samples - 146, time taken - 0.22mins
number of reviews in each cluster
-1    49854
 0      141
 1        5
Name: labels, dtype: int64
epsilon - 1.00, core samples - 151, time taken - 0.97mins
number of reviews in each cluster
-1    49849
 0      141
 2        5
 1        5
Name: labels, dtype: int64
epsilon - 2.00, core samples - 160, time taken - 3.59mins
number of reviews in each cluster
-1    49837
 1      141
 3        6
 2        6
 4        5
 0        5
Name: labels, dtype: int64
```

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

In [101]:
```
display(Markdown("#       displaying only 4 clusters with random 4 review's words"))
fig, axs = plt.subplots(2,2, figsize=(20,10))
cluster_list = [2,3,4,0]
for ax,eachCluster in zip(axs.ravel(),cluster_list):#https://stackoverflow.com/a/17211410/12901493
    wordList = []
    #https://docs.python.org/3/library/random.html#random.sample
    for eachReview in random.sample(avgw2v_DBSCAN_df[avgw2v_DBSCAN_df.labels == eachCluster].text.to_list(),3):
        for eachWord in eachReview.split(' '):
            wordList.append(eachWord)
    wordClud = wordcloud.WordCloud(width = 200, height = 100,
                                   background_color ='white',
                                   stopwords = stopwords).generate(' '.join(wordList))
    ax.imshow(wordClud)
    ax.set_title("words belonging to cluster: {:d}".format(eachCluster))
```

```
<IPython.core.display.Markdown object>
```



[5.3.3] Applying DBSCAN on TFIDF W2V, SET 4

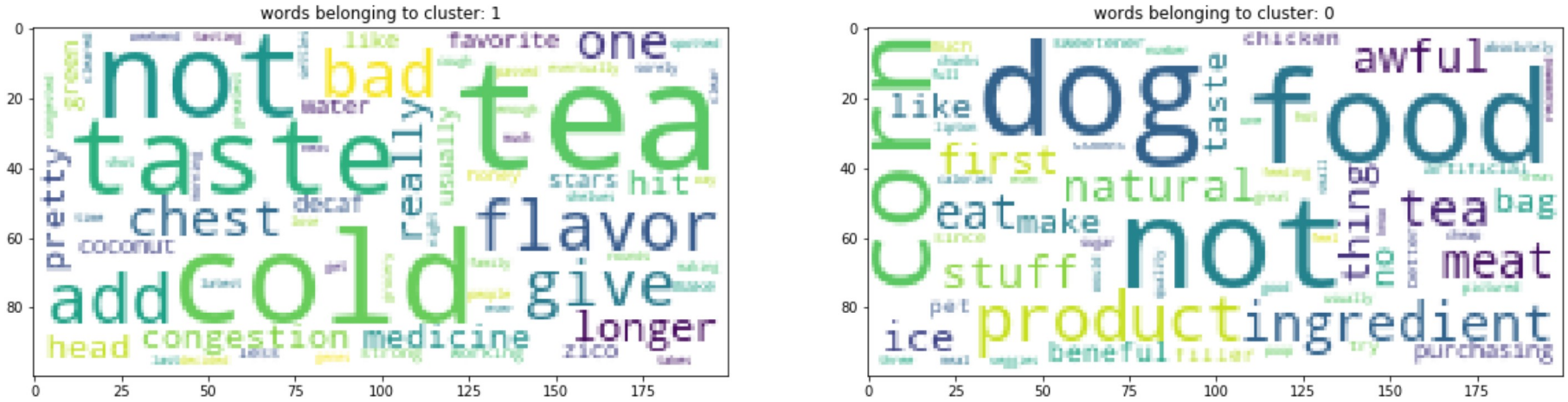
In [102]:
```
print("-1 cluster label is outliers")
for eachEpsilon in [0.5,1,2]:
    tfidfw2v_DBSCAN = DBSCAN(eps=eachEpsilon,
                             min_samples=5,
                             metric='euclidean',
                             metric_params=None,
                             algorithm='auto',
                             leaf_size=30,
                             p=None,
                             n_jobs=-1)
    start = time.time()
    tfidfw2v_DBSCAN.fit(TfidfWord2Vectors[100000:150000])
    timeTaken = time.time() - start
    print("epsilon - {:.2f}, core samples - {:d}, time taken - {:.2f}mins".
          format(eachEpsilon,len(avgw2v_DBSCAN.core_sample_indices_),(timeTaken/60)))
    print("number of reviews in each cluster")
    tfidfw2v_DBSCAN_df = pd.DataFrame(data={"text":preprocessed_reviews[:50000],"labels":tfidfw2v_DBSCAN.labels_})
```

```
-1 cluster label is outliers
epsilon - 0.50, core samples - 160, time taken - 0.26mins
number of reviews in each cluster
-1    49822
 0      170
 1        8
Name: labels, dtype: int64
epsilon - 1.00, core samples - 160, time taken - 1.03mins
number of reviews in each cluster
-1    49821
 0      170
 1        9
Name: labels, dtype: int64
epsilon - 2.00, core samples - 160, time taken - 4.51mins
number of reviews in each cluster
-1    49821
 0      170
 1        9
Name: labels, dtype: int64
```

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

In [103]:
```
display(Markdown("#       displaying only 2 clusters with random 4 review's words"))
fig, axs = plt.subplots(1,2, figsize=(20,10))
cluster_list = [1,0]
for ax,eachCluster in zip(axs.ravel(),cluster_list):#https://stackoverflow.com/a/17211410/12901493
    wordList = []
    #https://docs.python.org/3/library/random.html#random.sample
    for eachReview in random.sample(tfidfw2v_DBSCAN_df[tfidfw2v_DBSCAN_df.labels == eachCluster].text.to_list(),5):
        for eachWord in eachReview.split(' '):
            wordList.append(eachWord)
    wordClud = wordcloud.WordCloud(width = 200, height = 100,
                                   background_color ='white',
                                   stopwords = stopwords).generate(' '.join(wordList))
    ax.imshow(wordClud)
    ax.set_title("words belonging to cluster: {:d}".format(eachCluster))
```

```
<IPython.core.display.Markdown object>
```



[6] Conclusions

In [109]:
```
from prettytable import PrettyTable

table = PrettyTable()
table.field_names = ["K-Means++", "No. of Clusters", "Inertia"]
table.add_row(["bigrams(BOW)", bigrams_clusterer.n_clusters, bigrams_clusterer.inertia_])
table.add_row(["bigrams(tfidf)",tfidf_clusterer.n_clusters,tfidf_clusterer.inertia_])
table.add_row(["Avg. Word2Vec",avgw2v_clusterer.n_clusters,avgw2v_clusterer.inertia_])
```

In [110]:

```
+----------------+-----------------+--------------------+
|   K-Means++    | No. of Clusters |      Inertia       |
+----------------+-----------------+--------------------+
|  bigrams(BOW)  |        13       | 36137781197.304436 |
| bigrams(tfidf) |        5        | 36162848471.42067  |
| Avg. Word2Vec  |        7        | 33939578.799190745 |
| tfidf Word2Vec |        5        | 34010507.23937587  |
+----------------+-----------------+--------------------+
```

In [ ]: