

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

Objective:

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

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

[Ans] We could use Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
```

```

import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

import re
# Tutorial about Python regular expressions: https://pymotw.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 [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)
filtered_data.head(3)

```

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

Out[2]:

0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian		1		1	1303862400	Good Quality Dog Food
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa		0		0	1346976000	Not as Advertised
2	3	B000LQOCHO	ABXLMWJIXXAIN	Natalia Corres "Natalia		1		1	1219017600	"Delight" says it all

Id	ProductId	UserId	ProfileName	Score	Time	Summary

In [3]:

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

In [4]:

```
print(display.shape)
display.head()
```

(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-R12KPBDL2B5ZD	B007OSBEV0	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [5]:

```
display[display['UserId']=='AZY10LLTJ71NX']
```

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]:

```
display['COUNT(*)'].sum()
```

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
```

```

FROM reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
"""
, con)
display.head()

```

Out [7] :

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANILLA WAFE
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANILLA WAFE
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANILLA WAFE
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANILLA WAFE
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRA VANILLA WAFE

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 delete 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='last')

```

In [9] :

```

#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape

```

Out [9] :

(364173, 10)

In [10] :

```

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

```

```
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 calculations

```
In [11]:
```

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

```
Out[11]:
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3		1	5	1224892800 Bought This for My Son at College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3		2	4	1212883200 Pure cocoa taste with crunchy almonds inside

```
In [12]:
```

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
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?  
final['Score'].value_counts()
```

```
(364171, 10)
```

```
Out[13]:
```

```
1    307061  
0    57110  
Name: Score, dtype: int64
```

[3] Preprocessing

[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 observed 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)
print("=="*50)
```

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.... imagine my surprise when I ordered 2 boxes - both were expired! One expired back in 2005 for gosh sakes. I admit that Amazon 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 belongs in it is Canola oil. Canola or rapeseed is not something 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 until 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, delicious. Perfect. 3 ingredients: Water, Maltitol, Natural Maple Flavor. PERIOD. No chemicals. No garbage.

Have numerous friends & family members hooked on this stuff. My husband & son, who do NOT like "sugar free" prefer this over major label regular syrup.

I use this as my SWEETENER in baking: cheesecakes, white brownies, muffins, pumpkin pies, etc... Unbelievably delicious...

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_150)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

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

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()
print(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.... imagine my surprise when I ordered 2 boxes - both were expired! One expired back in 2005 for gosh sakes. I admit that Amazon 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 belongs in it is Canola oil. Canola or rapeseed is not something 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 until 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, delicious. Perfect. 3 ingredients: Water, Maltitol, Natural Maple Flavor. PERIOD. No chemicals. No garbage. Have numerous friends & family members hooked on this stuff. My husband & son, who do NOT like "sugar free" prefer this over major label regular syrup. I use this as my SWEETENER in baking: cheesecakes, white brownies, muffins, pumpkin pies, etc... Unbelievably 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)
    return phrase

```

In [18]:

```

sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("=="*50)

```

Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing I do not think belongs in it is Canola oil. Canola or rapeseed is not something 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 until they figured out a way to fix that. I still like it but it could be better.

I do not think belongs in it is Canola oil. Canola or rapeseed is not something 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 and even better oil than olive or virgin coconut, facts though say otherwise. Until the late 70 is it was poisonous until 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_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

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

In [20]:

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

Great ingredients although chicken should have been 1st 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 and even better oil than olive or virgin coconut facts though say otherwise Until the late 70 is it was poisonous until they figured out a way to fix that I still like it but it could be better

In [21]:

```
# 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 removed 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 students
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)
    preprocessed_reviews.append(sentance.strip())

```

100%|██████████| 364171/364171
[03:02<00:00, 1994.51it/s]

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

'great ingredients although chicken rather chicken broth thing not think belongs canola oil canola rapeseed not someting dog would ever find nature find rapeseed nature eat would poison today food industries convinced masses canola oil safe even better oil olive virgin coconut facts though say otherwise late poisonous figured way fix still like could better'

[3.2] Preprocessing Review Summary

In [24]:

```
## Similalrly you can do preprocessing for review summary also.
```

In [25]:

```

# sample data for kmeans cluster
data_kmeans = preprocessed_reviews[:50000]
scores_kmeans = final['Score'][:50000]
# sample data for Agglomerative and DBSCAN clusters
data = preprocessed_reviews[:5000]
scores = final['Score'][:5000]

```

In [26]:

```

from sklearn.model_selection import train_test_split

data_train_kmeans,data_test_kmeans,scores_train_kmeans,scores_test_kmeans =
train_test_split(data_kmeans, scores_kmeans, test_size=0.2, random_state=42,shuffle = False)
data_train_kmeans,data_cv_kmeans,scores_train_kmeans,scores_cv_kmeans =
train_test_split(data_train_kmeans, scores_train_kmeans, test_size=0.25, random_state=42,shuffle = False)

data_train,data_test,scores_train,scores_test = train_test_split(data, scores, test_size=0.2, random_state=42,shuffle = False)
data_train,data_cv,scores_train,scores_cv = train_test_split(data_train, scores_train, test_size=0.25, random_state=42,shuffle = False)

```

[4] Featurization

[4.1] BAG OF WORDS

In [27]:

```

# BoW
# kmeans
bow_vect_kmeans = CountVectorizer() #in scikit-learn
bow_vect_kmeans.fit(data_train_kmeans)

```

```
bow_train_kmeans = bow_vect_kmeans.fit_transform(data_train_kmeans)
bow_cv_kmeans = bow_vect_kmeans.transform(data_cv_kmeans)
bow_test_kmeans = bow_vect_kmeans.transform(data_test_kmeans)
```

[4.2] TF-IDF

In [28]:

```
#TF-IDF
# kmeans
tf_idf_vect_kmeans = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect_kmeans.fit(data_train_kmeans)
tf_idf_train_kmeans = tf_idf_vect_kmeans.fit_transform(data_train_kmeans)
tf_idf_cv_kmeans = tf_idf_vect_kmeans.transform(data_cv_kmeans)
tf_idf_test_kmeans = tf_idf_vect_kmeans.transform(data_test_kmeans)
```

[4.3] Word2Vec

In [29]:

```
# Train your own Word2Vec model using your own text corpus
# kmeans
X_train_kmeans = []
for i in data_train_kmeans:
    X_train_kmeans.append(i.split())

w2v_model_kmeans = Word2Vec(X_train_kmeans,min_count=5,size=50, workers=4)
w2v_words_kmeans = list(w2v_model_kmeans.wv.vocab)

# Agglomerative and DBSCAN clusters
X_train = []
for i in data_train:
    X_train.append(i.split())

w2v_model = Word2Vec(X_train,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
```

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

[4.3.1.1] Avg W2v

In [30]:

```
def avg_W2V(list_of_sentance,w2v_model,w2v_words):
    # 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 change this to 300 if you use google's w2v
        cnt_words = 0; # num of words with a valid vector in the sentence/review
        for word in sent.split(): # 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)
    return sent_vectors

# kmeans
avgw2v_train_kmeans = avg_W2V(data_train_kmeans,w2v_model_kmeans,w2v_words_kmeans)
avgw2v_cv_kmeans = avg_W2V(data_cv_kmeans,w2v_model_kmeans,w2v_words_kmeans)
avgw2v_test_kmeans = avg_W2V(data_test_kmeans,w2v_model_kmeans,w2v_words_kmeans)

# Agglomerative and DBSCAN clusters
avgw2v_train = avg_W2V(data_train,w2v_model,w2v_words)
```

```
avgw2v_cv = avg_W2V(data_cv,w2v_model,w2v_words)
avgw2v_test = avg_W2V(data_test,w2v_model,w2v_words)
```

100% [██████████] | 30000/30000 [01:
18<00:00, 382.53it/s]
100% [██████████] | 10000/10000 [00:
24<00:00, 404.50it/s]
100% [██████████] | 10000/10000 [00:
27<00:00, 370.01it/s]
100% [██████████] | 3000/3000
[00:04<00:00, 709.48it/s]
100% [██████████] | 1000/1000
[00:01<00:00, 697.23it/s]
100% [██████████] | 1000/1000
[00:01<00:00, 783.71it/s]

[4.3.1.2] TFIDF weighted W2v

In [54]:

```
# kmeans
tf_idf_w2v_model_kmeans = TfidfVectorizer()
tf_idf_w2v_matrix_kmeans = tf_idf_w2v_model_kmeans.fit_transform(data_train_kmeans)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_kmeans = dict(zip(tf_idf_w2v_model_kmeans.get_feature_names(),
list(tf_idf_w2v_model_kmeans.idf_)))
# TF-IDF weighted Word2Vec
tfidf_w2v_feat_kmeans = tf_idf_w2v_model_kmeans.get_feature_names() # tfidf words/col-names

# Agglomerative and DBSCAN clusters
tf_idf_w2v_model = TfidfVectorizer()
tf_idf_w2v_matrix = tf_idf_w2v_model.fit_transform(data_train)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tf_idf_w2v_model.get_feature_names(), list(tf_idf_w2v_model.idf_)))
# TF-IDF weighted Word2Vec
tfidf_w2v_feat = tf_idf_w2v_model.get_feature_names() # tfidf words/col-names
```

In [55]:

```
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

def tf_idf_w2v(list_of_sentance,w2v_model,w2v_words,tfidf_feat,dictionary):
    tfidf_sent_vectors = [] # the tfidf-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
        weight_sum = 0; # num of words with a valid vector in the sentence/review
        for word in sent.split(): # 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)

    return tfidf_sent_vectors

# kmeans
tf_idf_w2v_train_kmeans =
tf_idf_w2v(data_train_kmeans,w2v_model_kmeans,w2v_words_kmeans,tfidf_w2v_feat_kmeans,dictionary_kmeans)
tf_idf_w2v_cv_kmeans =
tf_idf_w2v(data_cv_kmeans,w2v_model_kmeans,w2v_words_kmeans,tfidf_w2v_feat_kmeans,dictionary_kmeans)
tf_idf_w2v_test_kmeans =
tf_idf_w2v(data_test_kmeans,w2v_model_kmeans,w2v_words_kmeans,tfidf_w2v_feat_kmeans,dictionary_kmeans)
```

```
# Agglomerative and DBSCAN clusters
tf_idf_w2v_train = tf_idf_w2v(data_train,w2v_model,w2v_words,tfidf_w2v_feat,dictionary)
tf_idf_w2v_cv = tf_idf_w2v(data_cv,w2v_model,w2v_words,tfidf_w2v_feat,dictionary)
tf_idf_w2v_test = tf_idf_w2v(data_test,w2v_model,w2v_words,tfidf_w2v_feat,dictionary)

100%|██████████| 30000/30000 [15:15<00:00, 32.77it/s]
100%|██████████| 10000/10000 [04:49<00:00, 36.23it/s]
100%|██████████| 10000/10000 [04:36<00:00, 36.16it/s]
100%|██████████| 3000/3000 [00:23<00:00, 125.90it/s]
100%|██████████| 1000/1000 [00:06<00:00, 148.81it/s]
100%|██████████| 1000/1000 [00:06<00:00, 153.21it/s]
```

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

1. Apply K-means Clustering on these feature sets:

- **SET 1:**Review text, preprocessed one converted into vectors using (BOW)
- **SET 2:**Review text, preprocessed one converted into vectors using (TFIDF)
- **SET 3:**Review text, preprocessed one converted into vectors using (AVG W2v)
- **SET 4:**Review text, preprocessed one converted into vectors using (TFIDF W2v)
- Find the best 'k' using the elbow-knee method (plot k vs inertia_)
- Once after you find the k clusters, plot the word cloud per each cluster so that at a single go we can analyze the words in a cluster.

2. Apply Agglomerative Clustering on these feature sets:

- **SET 3:**Review text, preprocessed one converted into vectors using (AVG W2v)
- **SET 4:**Review text, preprocessed one converted into vectors using (TFIDF W2v)
- Apply agglomerative algorithm and try a different number of clusters like 2,5 etc.
- Same as that of K-means, plot word clouds for each cluster and summarize in your own words what that cluster is representing.
- You can take around 5000 reviews or so(as this is very computationally expensive one)

3. Apply DBSCAN Clustering on these feature sets:

- **SET 3:**Review text, preprocessed one converted into vectors using (AVG W2v)
- **SET 4:**Review text, preprocessed one converted into vectors using (TFIDF W2v)
- Find the best 'Eps' using the [elbow-knee method](#).
- Same as before, plot word clouds for each cluster and summarize in your own words what that cluster is representing.
- You can take around 5000 reviews for this as well.

[5.1] K-Means Clustering

In [33]:

```
from sklearn.cluster import KMeans

def apply_KMeans_Classifier(X_train,X_cv,clusters):
    """This method apply KMeans cluster on given data"""
    inertia = []
    for k in clusters:
        clf = KMeans(n_clusters = k,random_state = 42)
        clf.fit(X_train)
        clf.predict(X_cv)
        inertia.append(clf.inertia_)

    return inertia
```

```

def plot_clusters_vs_inertia(clusters,inertia,title):
    """This method plot clusters vs inertia """
    sns.set_style("whitegrid",{'axes.grid' : False})
    plt.plot(clusters,inertia,"b-")
    plt.scatter(clusters,inertia)
    plt.xlabel("Clusters")
    plt.ylabel("Inertia")
    plt.title(title)
    plt.show()

def get_clusters(trainData,testData,k):
    """This method apply KMeans cluster and returns the clusters"""
    clf = KMeans(n_clusters = k,random_state = 42)
    clf.fit(trainData)
    clf.predict(testData)
    cluster_centers = clf.cluster_centers_
    cluster_labels = clf.labels_.tolist()
    return cluster_centers,cluster_labels

def plot_wordcloud(words,title):
    """This method plot the word cloud image"""
    # WordCloud
    # Reference : https://www.geeksforgeeks.org/generating-word-cloud-python/
    from wordcloud import WordCloud

    WC = WordCloud(background_color ='white').generate(words)
    # plot the WordCloud image
    plt.figure(figsize = (8, 8), facecolor = None)
    plt.imshow(WC)
    plt.axis("off")
    plt.tight_layout(pad = 0)
    plt.title(title,fontdict={'fontsize' : 20,'fontweight' : 20})
    plt.show()

```

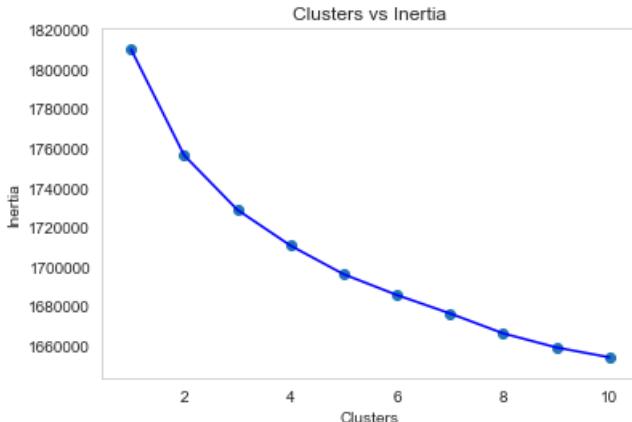
[5.1.1] Applying K-Means Clustering on BOW, SET 1

In [34]:

```

# clusters
K_clusters = [1,2,3,4,5,6,7,8,9,10]
# apply KMeans
inertia = apply_KMeans_Classifier(bow_train_kmeans,bow_cv_kmeans,K_clusters)
# plot inertia vs clusters
plot_clusters_vs_inertia(K_clusters,inertia,"Clusters vs Inertia")
bow_best_K_kmeans = K_clusters[inertia.index(min(inertia))]
kmeans_bow_inertia = min(inertia)
# apply KMeans with best K
cluster_centers_bow_kmeans,cluster_labels =
get_clusters(bow_train_kmeans,bow_test_kmeans,bow_best_K_kmeans)

```

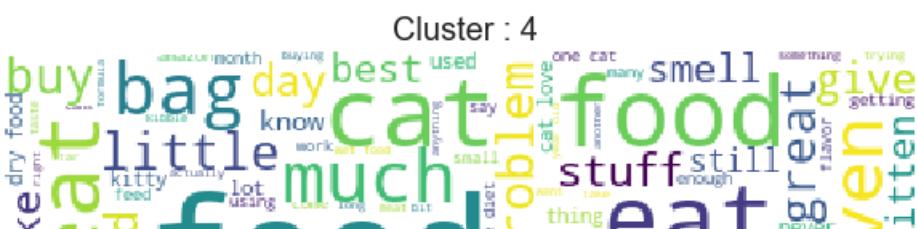
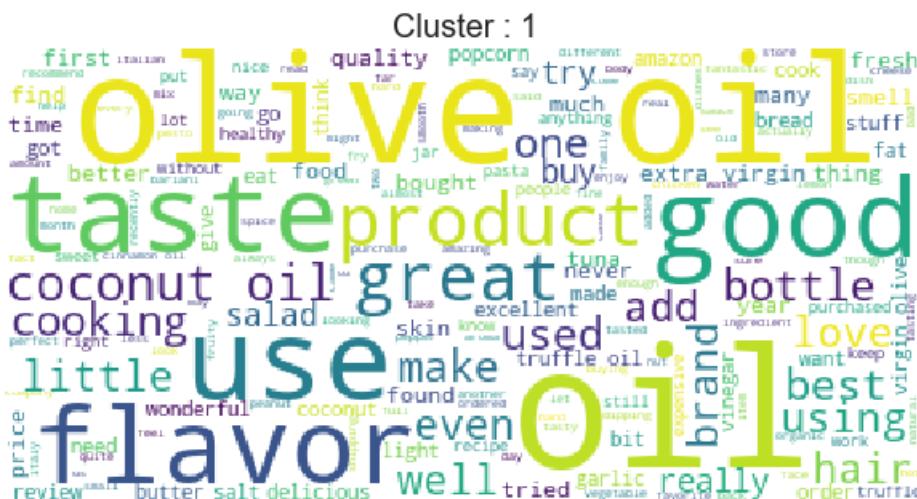


[5.1.2] Wordclouds of clusters obtained after applying k-means on BOW SET 1

In [46]:

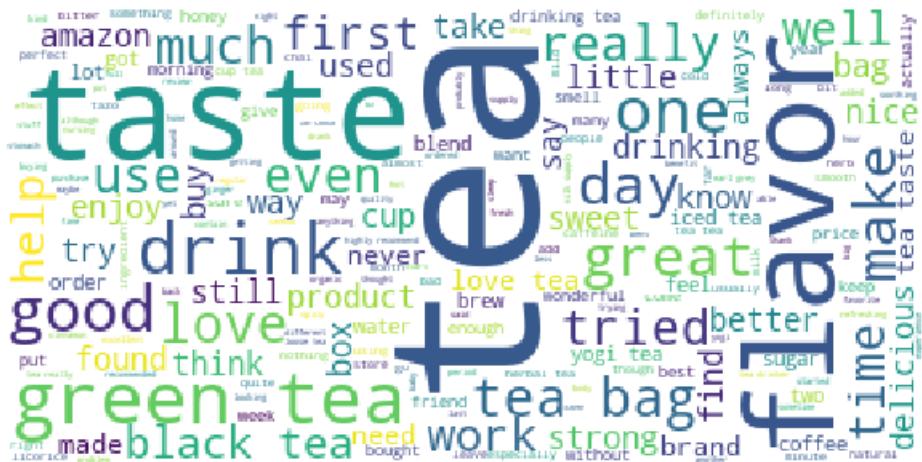
```
# wordclouds
```

```
# WUJIAOLUO
for i in range(0,bow_best_K_kmeans):
    index = [j for j,val in enumerate(cluster_labels) if val==i]
    features = list(np.array(data_train_kmeans)[index])
    words = " ".join(feature for feature in features)
    plot_wordcloud(words,"Cluster : "+str(i+1))
```





Cluster : 5



Cluster : 6



Cluster : 7



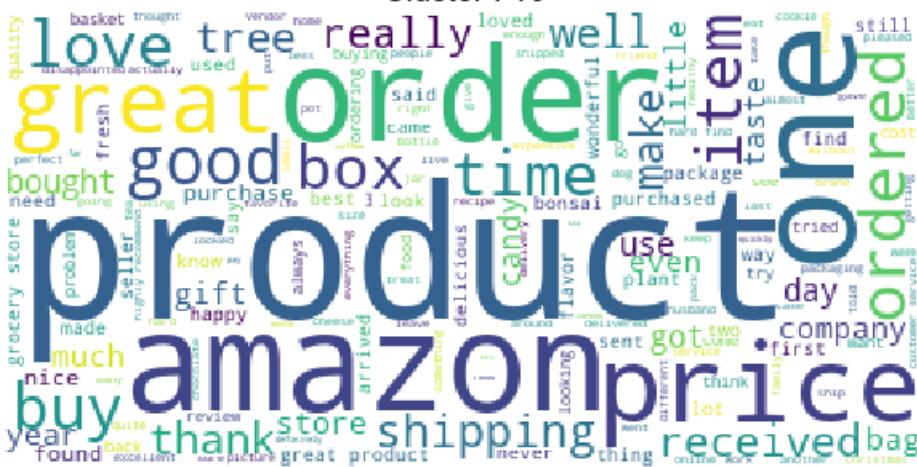
Cluster : 8



Cluster : 9



Cluster : 10



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

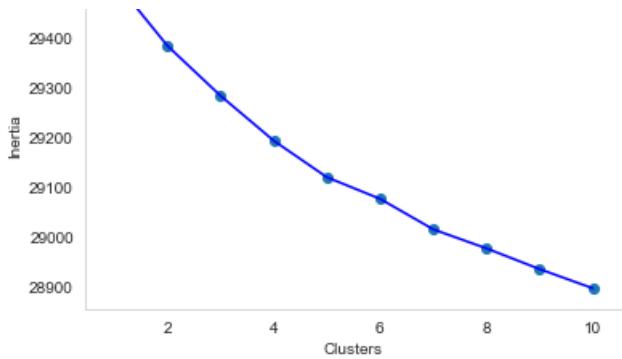
In [36]:

```

# clusters
K_clusters = [1,2,3,4,5,6,7,8,9,10]
# apply KMeans
inertia = apply_KMeans_Classifier(tf_idf_train_kmeans,tf_idf_cv_kmeans,K_clusters)
# plot inertia vs clusters
plot_clusters_vs_inertia(K_clusters,inertia,"Clusters vs Inertia")
tf_idf_best_K_kmeans = K_clusters[inertia.index(min(inertia))]
kmeans tf idf inertia = min(inertia)

```





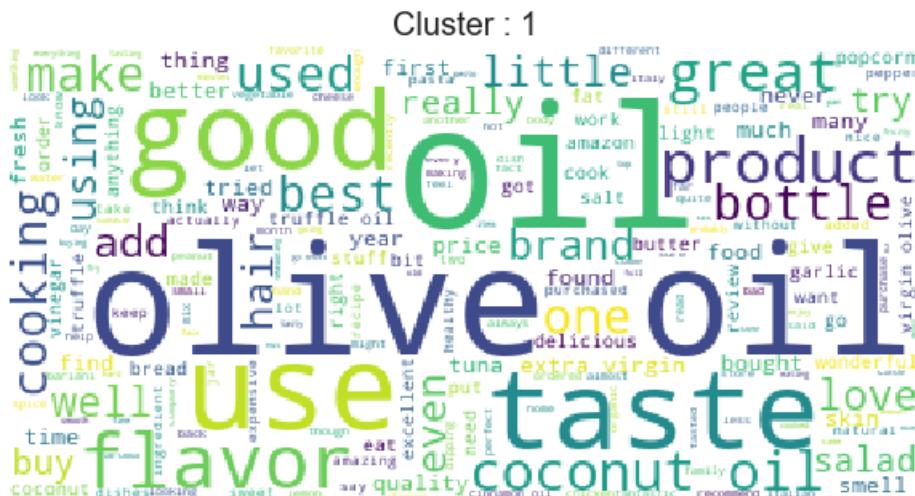
In [39]:

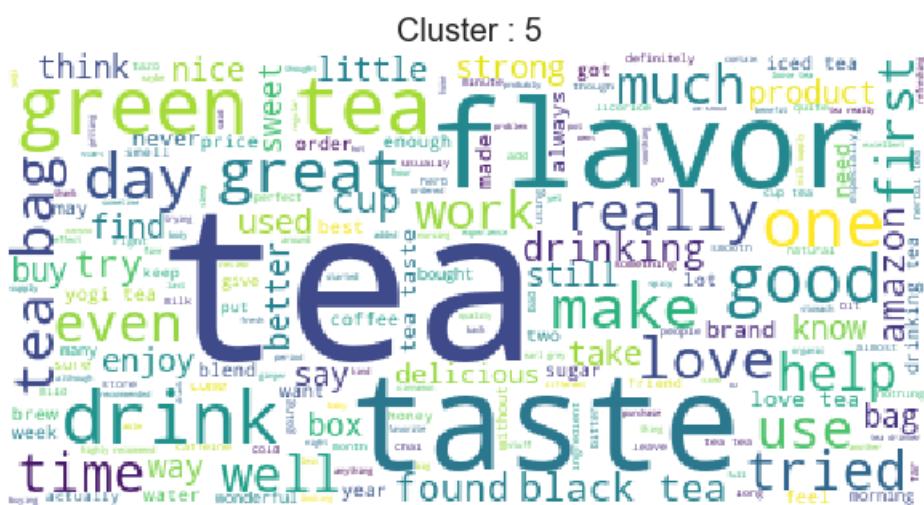
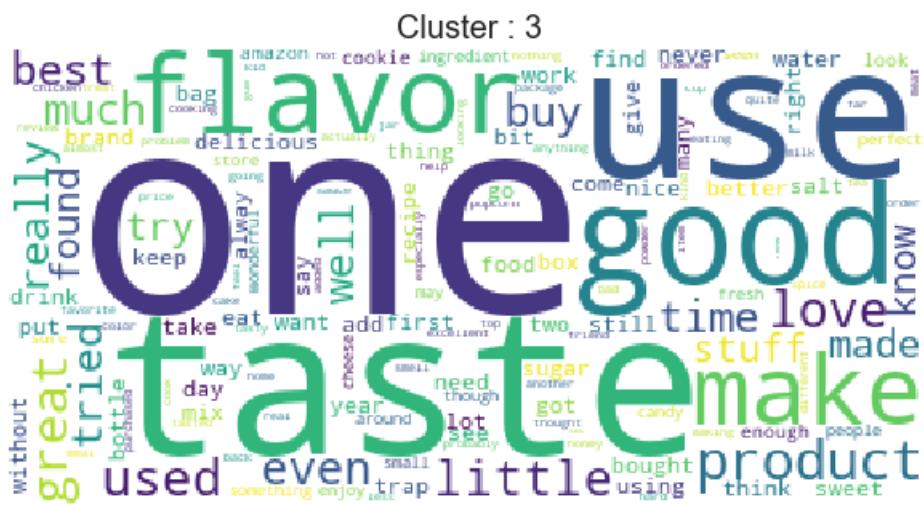
```
# apply KMeans with best K
cluster_centers_tf_idf_kmeans,tf_idf_cluster_labels =
get_clusters(tf_idf_train_kmeans,tf_idf_test_kmeans,tf_idf_best_K_kmeans)
```

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

In [45]:

```
# WordCloud
tf_idf_features = tf_idf_vect_kmeans.get_feature_names()
for i in range(0,tf_idf_best_K_kmeans):
    index = [j for j, val in enumerate(cluster_labels) if val==i]
    features = list(np.array(data_train_kmeans)[index])
    words = " ".join(feature for feature in features)
    plot_wordcloud(words,"Cluster : "+str(i+1))
```







Cluster : 7

Cluster : 8

Cluster : 9

Cluster : 10



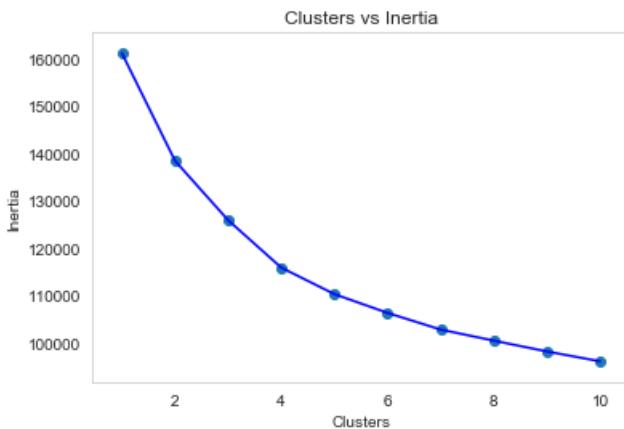
[5.1.5] Applying K-Means Clustering on AVG W2V, SET 3

In [47]:

```

# clusters
K_clusters = [1,2,3,4,5,6,7,8,9,10]
# apply KMeans
inertia = apply_KMeans_Classifier(avgw2v_train_kmeans,avgw2v_cv_kmeans,K_clusters)
# plot inertia vs clusters
plot_clusters_vs_inertia(K_clusters,inertia,"Clusters vs Inertia")
avgw2v_best_K_kmeans = K_clusters[inertia.index(min(inertia))]
kmeans_avgw2v_inertia = min(inertia)
# apply KMeans with best K
cluster_centers_avgw2v_kmeans,cluster_labels = get_clusters(avgw2v_train_kmeans,avgw2v_test_kmeans
,avgw2v_best_K_kmeans)

```



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

In [49]:

```
# WordCloud
for i in range(0,avgw2v_best_K_kmeans):
    index = [j for j,val in enumerate(cluster_labels) if val==i]
    features = list(np.array(data_train_kmeans)[index])
    words = " ".join(feature for feature in features)
    plot wordcloud(words,"Cluster : "+str(i+1))
```



Cluster : 2



Cluster : 3



Cluster : 4



Cluster : 5





Cluster : 6



Cluster : 7

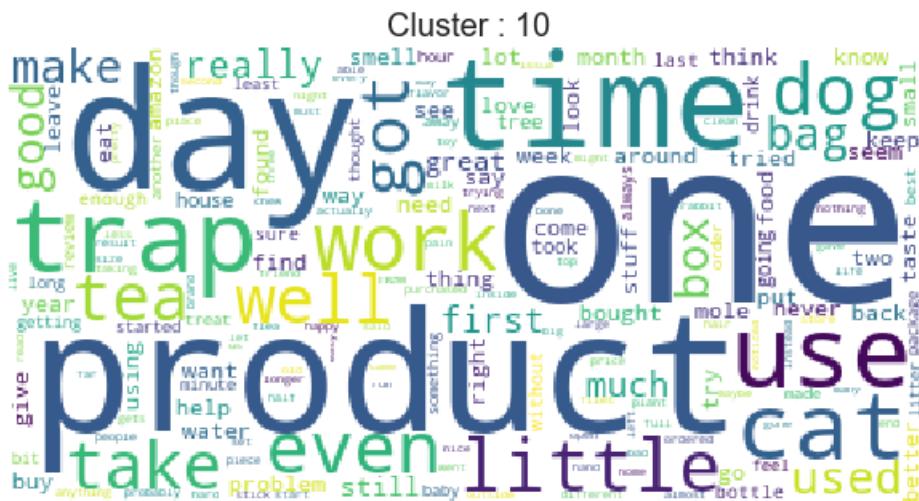
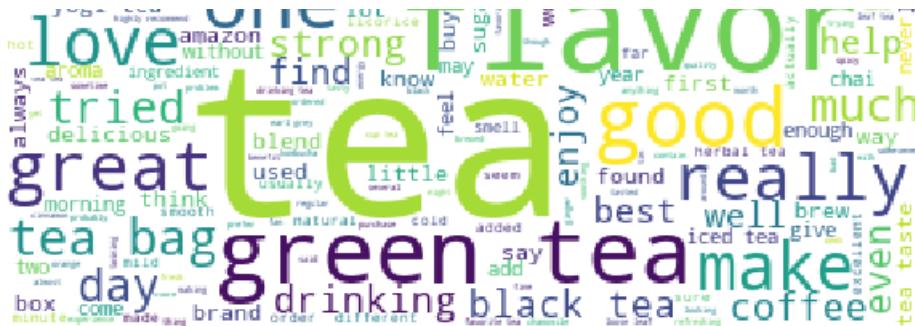


Cluster : 8



Cluster : 9





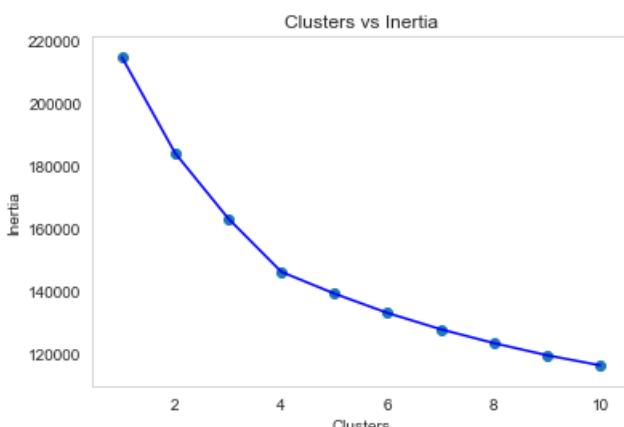
[5.1.7] Applying K-Means Clustering on TFIDF W2V, SET 4

In [56]:

```

# clusters
K_clusters = [1,2,3,4,5,6,7,8,9,10]
# apply KMeans
inertia = apply_KMeans_Classifier(tf_idf_w2v_train_kmeans,tf_idf_w2v_cv_kmeans,K_clusters)
# plot inertia vs clusters
plot_clusters_vs_inertia(K_clusters,inertia,"Clusters vs Inertia")
tf_idf_w2v_best_K_kmeans = K_clusters[inertia.index(min(inertia))]
kmeans_tf_idf_w2v_inertia = min(inertia)
# apply KMeans with best K
cluster_centers_tf_idf_w2v_kmeans,cluster_labels =
get_clusters(tf_idf_w2v_train_kmeans,tf_idf_w2v_test_kmeans,tf_idf_w2v_best_K_kmeans)

```

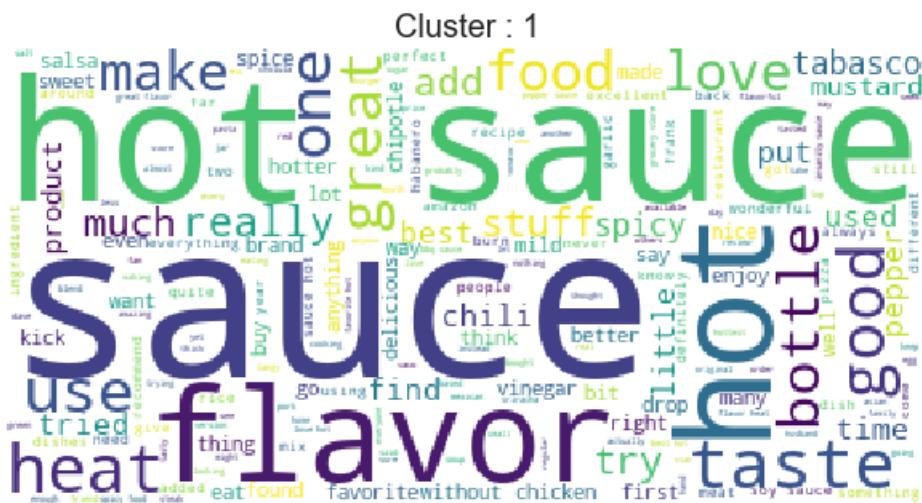


[5.1.8] Wordclouds of clusters obtained after applying k-means on TFIDF W2V SET 4

In [57]:

```
# WordCloud
```

```
for i in range(0,tf_idf_w2v_best_K_kmeans):
    index = [j for j,val in enumerate(cluster_labels) if val==i]
    features = list(np.array(data_train_kmeans)[index])
    words = " ".join(feature for feature in features)
    plot_wordcloud(words,"Cluster : "+str(i+1))
```



Cluster : 2



Cluster : 3



Cluster : 4





Cluster : 5



Cluster : 6

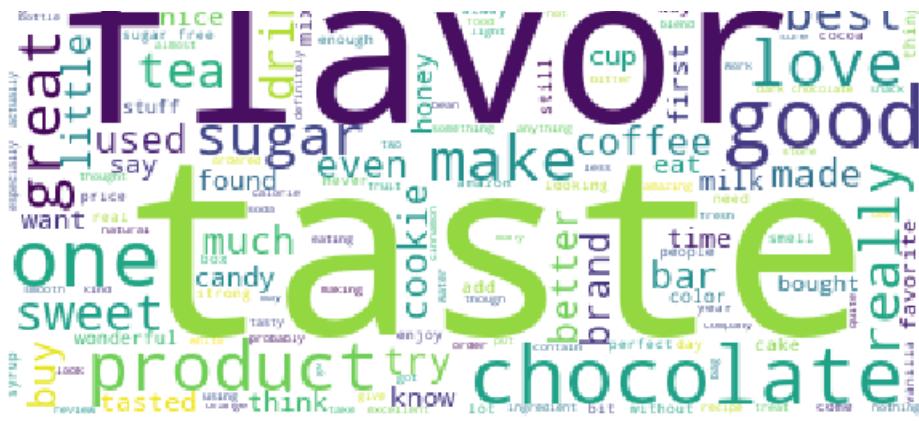


Cluster : 7



Cluster : 8





Cluster : 9



Cluster : 10



[5.2] Agglomerative Clustering

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

In [69]:

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import davies_bouldin_score

def apply_Agglomerative_Cluster(X_train,X_cv,clusters):
    """This method apply Agglomerative cluster on given data"""
    DBI = []
    for k in clusters:
        clf = AgglomerativeClustering(n_clusters = k)
        clf.fit(X_train)
        -clf.fit_predict(X_cv)
```

```

    clf.fit_predict(X_cv)
    cluster_labels = clf.labels_
    DBI.append(davies_bouldin_score(X_cv,cluster_labels))

    return DBI

def plot_clusters_vs_dbIndex(clusters,DBI,title):
    """This method plot clusters vs DB Index """
    sns.set_style("whitegrid",{'axes.grid' : False})
    plt.plot(clusters,DBI,"b-")
    plt.scatter(clusters,DBI)
    plt.xlabel("Clusters")
    plt.ylabel("DB Index")
    plt.title(title)
    plt.show()

def get_clusters_and_labels(trainData,testData,k):
    """This method apply Agglomerative cluster and returns the clusters and labels"""
    clf = AgglomerativeClustering(n_clusters = k)
    clf.fit(trainData)
    clf.fit_predict(testData)
    cluster_labels = clf.labels_.tolist()
    return cluster_labels

def plot_wordcloud(words,title):
    """This method plot the word cloud image"""
    # WordCloud
    # Reference : https://www.geeksforgeeks.org/generating-word-cloud-python/
    from wordcloud import WordCloud
    WC = WordCloud(background_color ='white').generate(words)
    # plot the WordCloud image
    plt.figure(figsize = (8, 8), facecolor = None)
    plt.imshow(WC)
    plt.axis("off")
    plt.tight_layout(pad = 0)
    plt.title(title,fontdict={'fontsize' : 20,'fontweight' : 20})
    plt.show()

```

In [70]:

```

# clusters
clusters = [2,3,4,5,6,7,8,9,10]
# apply Agglomerative cluster
db_index = apply_Agglomerative_Cluster(avgw2v_train,avgw2v_cv,clusters)
# plot db_index vs clusters
plot_clusters_vs_dbIndex(clusters,db_index,"Clusters vs DB_Index")
avgw2v_best_K_agglomerative = clusters[db_index.index(max(db_index)) ]
agglo_db_index_avgw2v = max(db_index)
# apply Agglomerative with best K
cluster_labels = get_clusters_and_labels(avgw2v_train,avgw2v_test,avgw2v_best_K_agglomerative)

```

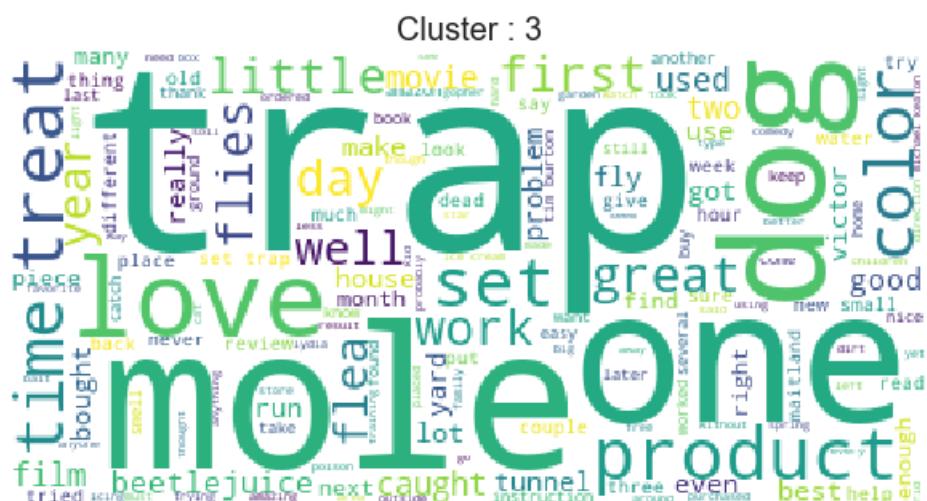
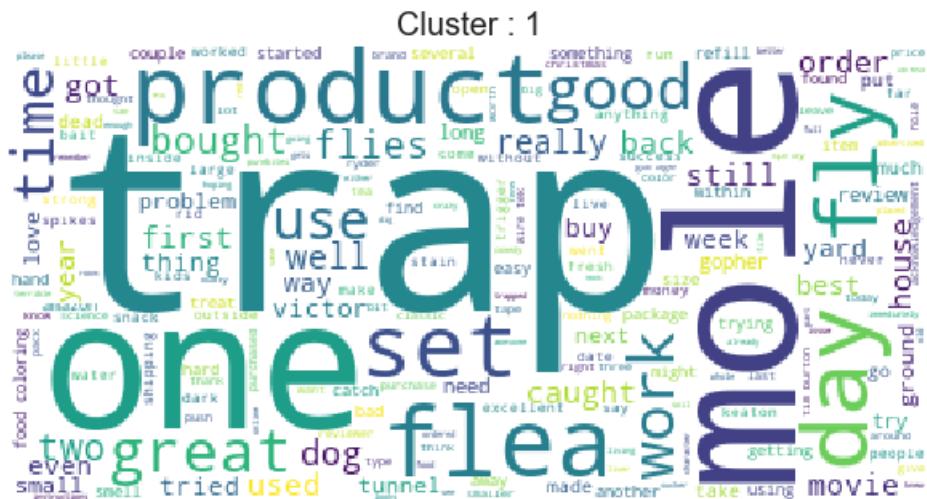


[5.2.2] Wordclouds of clusters obtained after applying Agglomerative Clustering on AVG W2V SET 3

In [71]:

...

```
# WORACIOUOA
for i in range(0,avgw2v_best_K_agglomerative):
    index = [j for j,val in enumerate(cluster_labels) if val==i]
    features = list(np.array(data_train)[index])
    words = " ".join(feature for feature in features)
    plot wordcloud(words,"Cluster : "+str(i+1))
```





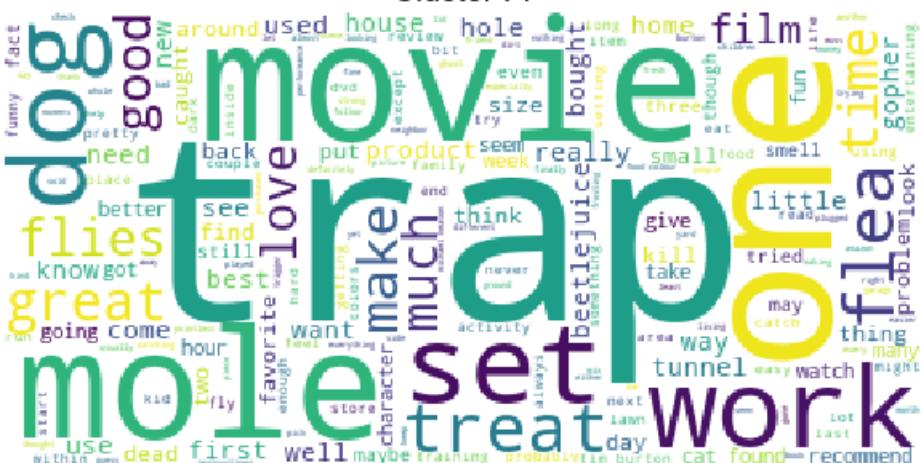
Cluster : 5



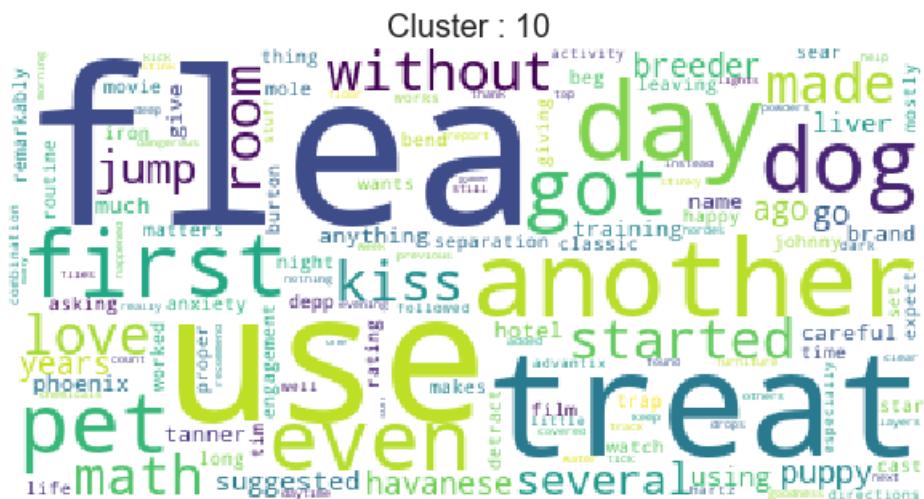
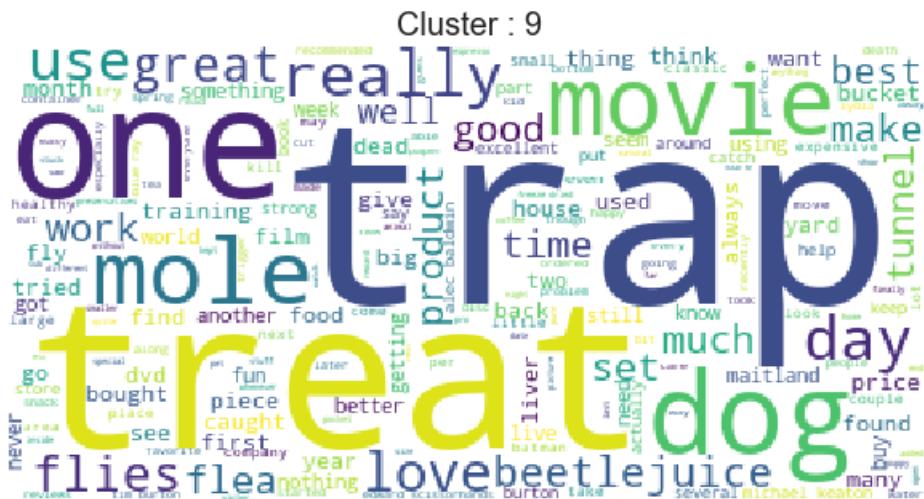
Cluster : 6



Cluster : 7



Cluster : 8



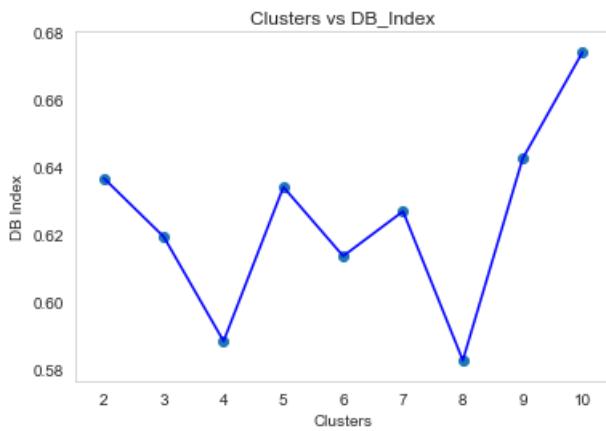
[5.2.3] Applying Agglomerative Clustering on TFIDF W2V, SET 4

In [73]:

```

# clusters
clusters = [2,3,4,5,6,7,8,9,10]
# apply Agglomerative cluster
db_index = apply_Agglomerative_Cluster(tf_idf_w2v_train,tf_idf_w2v_cv,clusters)
# plot db_index vs clusters
plot_clusters_vs_dbIndex(clusters,db_index,"Clusters vs DB_Index")
tfidf_w2v_best_K_agglomerative = clusters[db_index.index(max(db_index))]
agglo_db_index_tf_idf_w2v = max(db_index)
# apply Agglomerative with best K
cluster_labels =
get clusters and labels(tf idf w2v train,tf idf w2v test,tfidf w2v best K agglomerative)

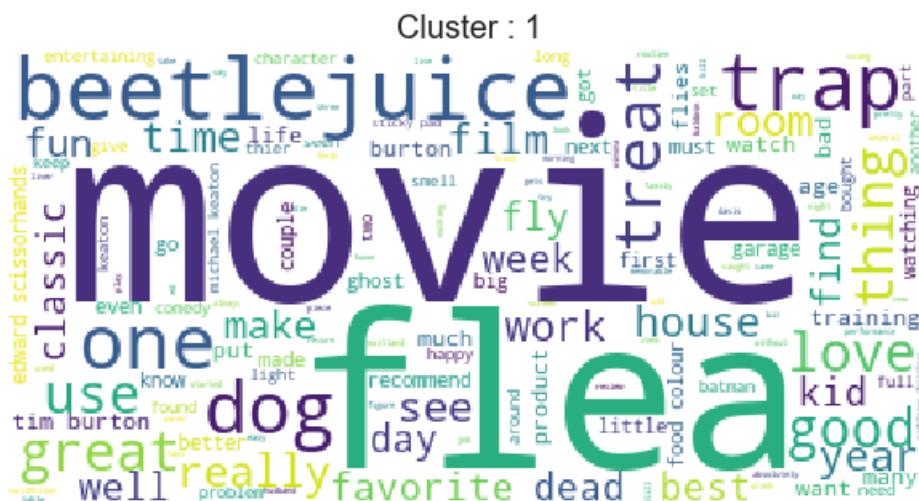
```



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

In [74]:

```
# wordcloud
for i in range(0,tfidf_w2v_best_K_agglomerative):
    index = [j for j,val in enumerate(cluster_labels) if val==i]
    features = list(np.array(data_train)[index])
    words = " ".join(feature for feature in features)
    plot_wordcloud(words,"Cluster : "+str(i+1))
```





Cluster : 4



Cluster : 5



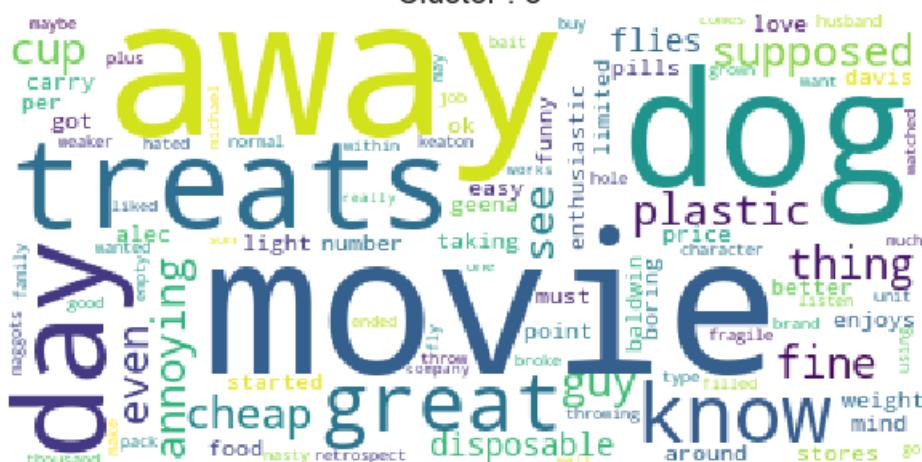
Cluster : 6



Cluster : 7



Cluster : 8



Cluster : 9



Cluster : 10





[5.3] DBSCAN Clustering

In [124]:

```
from sklearn.cluster import DBSCAN
from sklearn.metrics import davies_bouldin_score

def apply_DBSCAN_Cluster(X_train,X_cv,epsilon):
    """This method apply DBSCAN Cluster on given data"""
    DBI = []
    for e in epsilon:
        clf = DBSCAN(eps = e)
        clf.fit(X_train)
        clf.fit_predict(X_cv)
        cluster_labels = clf.labels_
        DBI.append(davies_bouldin_score(X_cv,cluster_labels))

    return DBI

def plot_Epsilon_vs_dbIndex(clusters,DBI,title):
    """This method plot epsilon vs DB Index """
    sns.set_style("whitegrid",{'axes.grid' : False})
    plt.plot(clusters,DBI,"b-")
    plt.scatter(clusters,DBI)
    plt.xlabel("Epsilon")
    plt.ylabel("DB Index")
    plt.title(title)
    plt.show()

def get_clusters_and_labels(trainData,testData,e):
    """This method apply DBSCAN cluster and returns the clusters and labels"""
    clf = DBSCAN(eps=e)
    clf.fit(trainData)
    clf.fit_predict(testData)
    cluster_labels = clf.labels_.tolist()
    return cluster_labels

def plot_wordcloud(words,title):
    """This method plot the word cloud image"""
    # WordCloud
    # Reference : https://www.geeksforgeeks.org/generating-word-cloud-python/
    from wordcloud import WordCloud
    WC = WordCloud(background_color ='white').generate(words)
    # plot the WordCloud image
    plt.figure(figsize = (8, 8), facecolor = None)
    plt.imshow(WC)
    plt.axis("off")
    plt.tight_layout(pad = 0)
    plt.title(title,fontdict={'fontsize' : 20,'fontweight' : 20})
    plt.show()
```

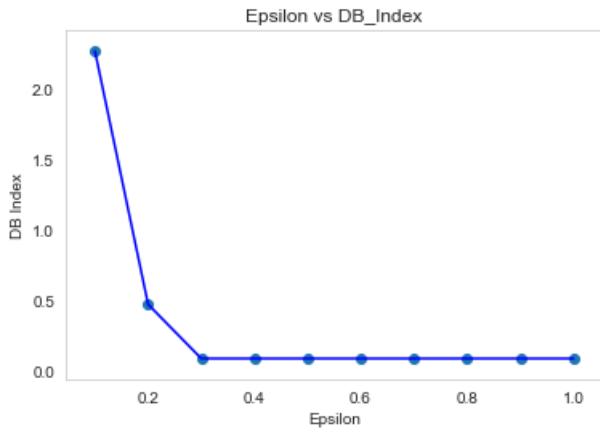
[5.3.1] Applying DBSCAN on AVG W2V, SET 3

In [125]:

```

# Epsilon
epsilon = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
# apply DBSCAN cluster
db_index = apply_DBSCAN_Cluster(avgw2v_train,avgw2v_cv,epsilon)
# plot db_index vs epsilon
plot_Epsilon_vs_dbIndex(epsilon,db_index,"Epsilon vs DB_Index")
avgw2v_best_eps_dbscan = epsilon[db_index.index(max(db_index)) ]
dbscan_db_index_avgw2v = max(db_index)
# apply DBSCAN with best K
cluster_labels = get_clusters_and_labels(avgw2v_train,avgw2v_test,avgw2v_best_eps_dbscan)

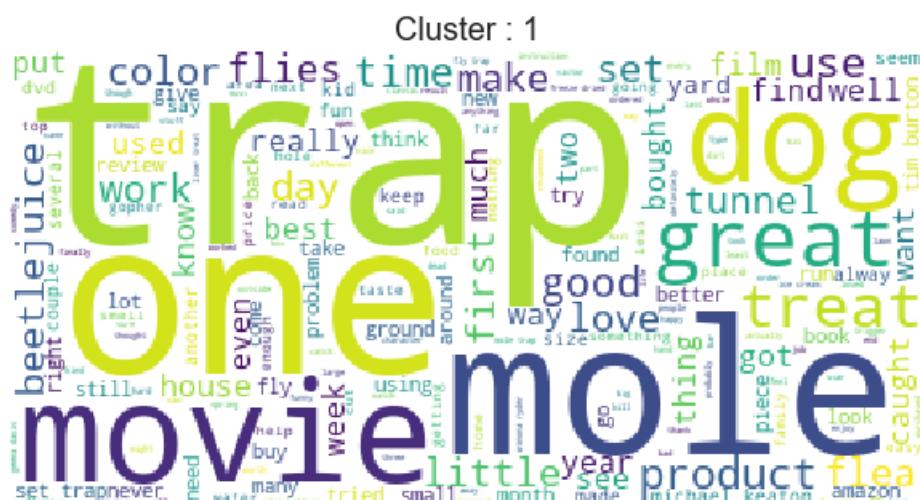
```



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

In [126]:

```
# wordcloud
lst = np.unique(np.array(cluster_labels))
for i in lst:
    index = [j for j, val in enumerate(cluster_labels) if val==i]
    features = list(np.array(data_train)[index])
    words = " ".join(feature for feature in features)
    plot wordcloud(words, "Cluster : "+str(i+1))
```



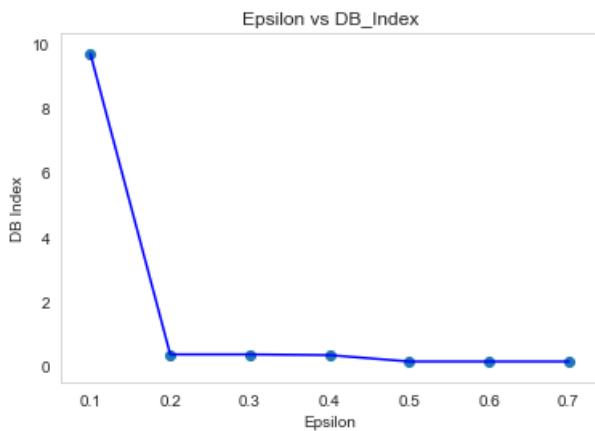
דָּבָר, זְמָן, וְעֵדָה שֶׁאַתָּה מִזְמְנָרָה לְעֵדָה

In [127]:

```

# Epsilon
epsilon = [0.1,0.2,0.3,0.4,0.5,0.6,0.7]
# apply DBSCAN cluster
db_index = apply_DBSCAN_Cluster(tf_idf_w2v_train,tf_idf_w2v_cv,epsilon)
# plot db_index vs clusters
plot_clusters_vs_dbIndex(epsilon,db_index,"Epsilon vs DB_Index")
tfidf_w2v_best_eps_dbscan = epsilon[db_index.index(max(db_index)) ]
dbscan_db_index_tf_idf_w2v = max(db_index)
# apply DBSCAN with best K
cluster_labels =
get_clusters_and_labels(tf_idf_w2v_train,tf_idf_w2v_test,tfidf_w2v_best_eps_dbscan)

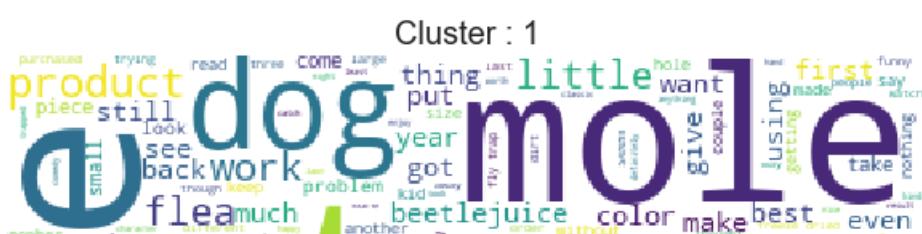
```

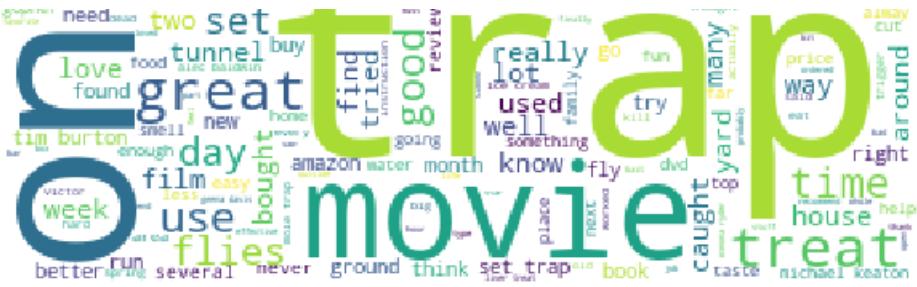


[5.3.4] Wordclouds of clusters obtained after applying DBSCAN on TFIDF W2V SET 4

In [128]:

```
# wordcloud
lst = np.unique(np.array(cluster_labels))
for i in lst:
    index = [j for j, val in enumerate(cluster_labels) if val==i]
    features = list(np.array(data_train)[index])
    words = " ".join(feature for feature in features)
    plot wordcloud(words, "Cluster : "+str(i+1))
```





[6] Conclusions

In [129]:

```
# Please compare all your models using Prettytable library.
# You can have 3 tables, one each for kmeans, agglomerative and dbscan

from prettytable import PrettyTable

# KMeans
table = PrettyTable()
table.field_names = ["Vectorization", "K", "Inertia"]
table.add_row(["BOW", bow_best_K_kmeans, round(kmeans_bow_inertia, 2)])
table.add_row(["TF-IDF", tf_idf_best_K_kmeans, round(kmeans_tf_idf_inertia, 2)])
table.add_row(["Avg-W2V", avgw2v_best_K_kmeans, round(kmeans_avgw2v_inertia, 2)])
table.add_row(["TF-IDF W2V", tf_idf_w2v_best_K_kmeans, round(kmeans_tf_idf_w2v_inertia, 2)])

print(table.get_string(title="KMeans Cluster Results"))

# Agglomerative
table = PrettyTable()
table.field_names = ["Vectorization", "K", "DB Index"]
table.add_row(["Avg-W2V", avgw2v_best_K_agglomerative, round(agglo_db_index_avgw2v, 2)])
table.add_row(["TF-IDF W2V", tfidf_w2v_best_K_agglomerative, round(agglo_db_index_tf_idf_w2v, 2)])

print(table.get_string(title="Agglomerative Cluster Results"))

# DBSCAN
table = PrettyTable()
table.field_names = ["Vectorization", "Epsilon", "DB Index"]
table.add_row(["Avg-W2V", avgw2v_best_eps_dbscan, round(dbscan_db_index_avgw2v, 2)])
table.add_row(["TF-IDF W2V", tfidf_w2v_best_eps_dbscan, round(dbscan_db_index_tf_idf_w2v, 2)])

print(table.get_string(title="DBSCAN Cluster Results"))
```

KMeans Cluster Results		
Vectorization	K	Inertia
BOW	10	1654056.51
TF-IDF	10	28896.04
Avg-W2V	10	96006.53
TF-IDF W2V	10	116429.13

Agglomerative Cluster Results		
Vectorization	K	DB Index
Avg-W2V	10	0.7
TF-IDF W2V	10	0.67

DBSCAN Cluster Results		
Vectorization	Epsilon	DB Index
Avg-W2V	0.1	2.26
TF-IDF W2V	0.1	9.69

