K-Means Clustering on Amazon Food Reviews

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

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

Objective: Given Amazon Food reviews, convert all the reviews into a vector by taking 10000 data points using three techniques:
1. BoW.
2. TFIDF.
3. Average W2V.

Then perform following tasks under each technique:
Task 1. Perform Cross Validation on K-Means and use elbow method to find optimal number of clusters.
Task 2. Apply K-Means.

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

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

Loading the data

SQLite Database

In order to load the data, We have used the SQLITE dataset as it 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 id above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [130]:
```
import sqlite3
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from gensim.models import Word2Vec
import gensim
from sklearn.preprocessing import StandardScaler

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

In [2]:
```
connection = sqlite3.connect("FinalAmazonFoodReviewsDataset.sqlite")
```

In [3]:
```
data = pd.read_sql_query("SELECT * FROM Reviews", connection)
```

In [4]:
```
data.head()
```

Out[4]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	
0	0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	Positive	1303862400	Good Quality Dog Food	se
1	1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	Negative	1346976000	Not as Advertised	lal
2	2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	Positive	1219017600	"Delight" says it all	F . cc a
3	4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	Positive	1350777600	Great taffy	t
4	5	6	B006K2ZZ7K	ADT0SRK1MGOEU	Twoapennything	0	0	Positive	1342051200	Nice Taffy	.

In [5]:
```
data.shape
```

Out[5]: (364171, 12)

In [6]:
```
data["Score"].value_counts()
```

Out[6]:
```
Positive    307061
Negative     57110
Name: Score, dtype: int64
```

In [7]:
```
def changingScores(score):
    if score == "Positive":
        return 1
    else:
        return 0
```

In [8]:
```
# changing score
# Positive = 1
# Negative = 0
actualScore = list(data["Score"])
positiveNegative = list(map(changingScores, actualScore)) #map(function, list of numbers)
data['Score'] = positiveNegative
```

In [9]:
```
data.head()
```

Out[9]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	
0	0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food	I b seve V ca
1	1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertised	Pr a label J S Pea
2	2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all	Thi confe tha aro
3	4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	1	1350777600	Great taffy	taff . \
4	5	6	B006K2ZZ7K	ADT0SRK1MGOEU	Twoapennything	0	0	1	1342051200	Nice Taffy	I wil fo ore tl

In [10]:
```
#taking 5000 random samples
data = data.sample(n = 5000)
```

In [11]:
```
data.shape
```
Out[11]:
```
(5000, 12)
```

In [12]:
```
data["Score"].value_counts()
```
Out[12]:
```
1    4241
0     759
Name: Score, dtype: int64
```

In [13]:
```
Data = data
```

In [14]:
```
Data_Labels = data["Score"]
```

In [15]:
```
print(Data.shape)
print(Data_Labels.shape)
```
```
(5000, 12)
(5000,)
```

In [16]:
```
Data.head()
```

Out[16]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summai
145007	205776	222967	B000UUWECC	ASAOKINJWVPXY	Michael J. Price	0	0	1	1272931200	Gre Produ f Sensitiv Stomach of the El
150202	213516	231402	B003FDC2I2	AB5TNEYXHO7JX	Jenn	1	1	1	1339632000	Worth
318913	458489	495717	B0098WV8F2	A4SBF0TQN6S5L	JGM "Moi"	0	0	1	1325980800	Yur
25238	36951	40147	B001EQ4ONI	A2524C1GBZCL0B	mickey	0	0	1	1295568000	wor buyir
340602	491574	531500	B000DZFMFA	A1HKBX2L0DV258	Dena Leasure	1	1	1	1259625600	Glute fre Cookie

BoW

In [17]:
```
count_vect = CountVectorizer()
Data_BoW = count_vect.fit_transform(Data["ProcessedText"].values)
```

In [18]:
```
print(type(Data_BoW))
print(Data_BoW.shape)
```

```
<class 'scipy.sparse.csr.csr_matrix'>
(5000, 9525)
```

In [19]:
```
#Standardizing our data matrix
Data_BoW_Std = StandardScaler(with_mean = False).fit_transform(Data_BoW)
print(Data_BoW_Std.shape)
print(type(Data_BoW_Std))
```

```
(5000, 9525)
<class 'scipy.sparse.csr.csr_matrix'>
```

```
C:\Users\GauravP\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning: Data with input dt
ype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

Task 1. Perform Cross Validation on K-Means and use elbow method to find optimal number of clusters.

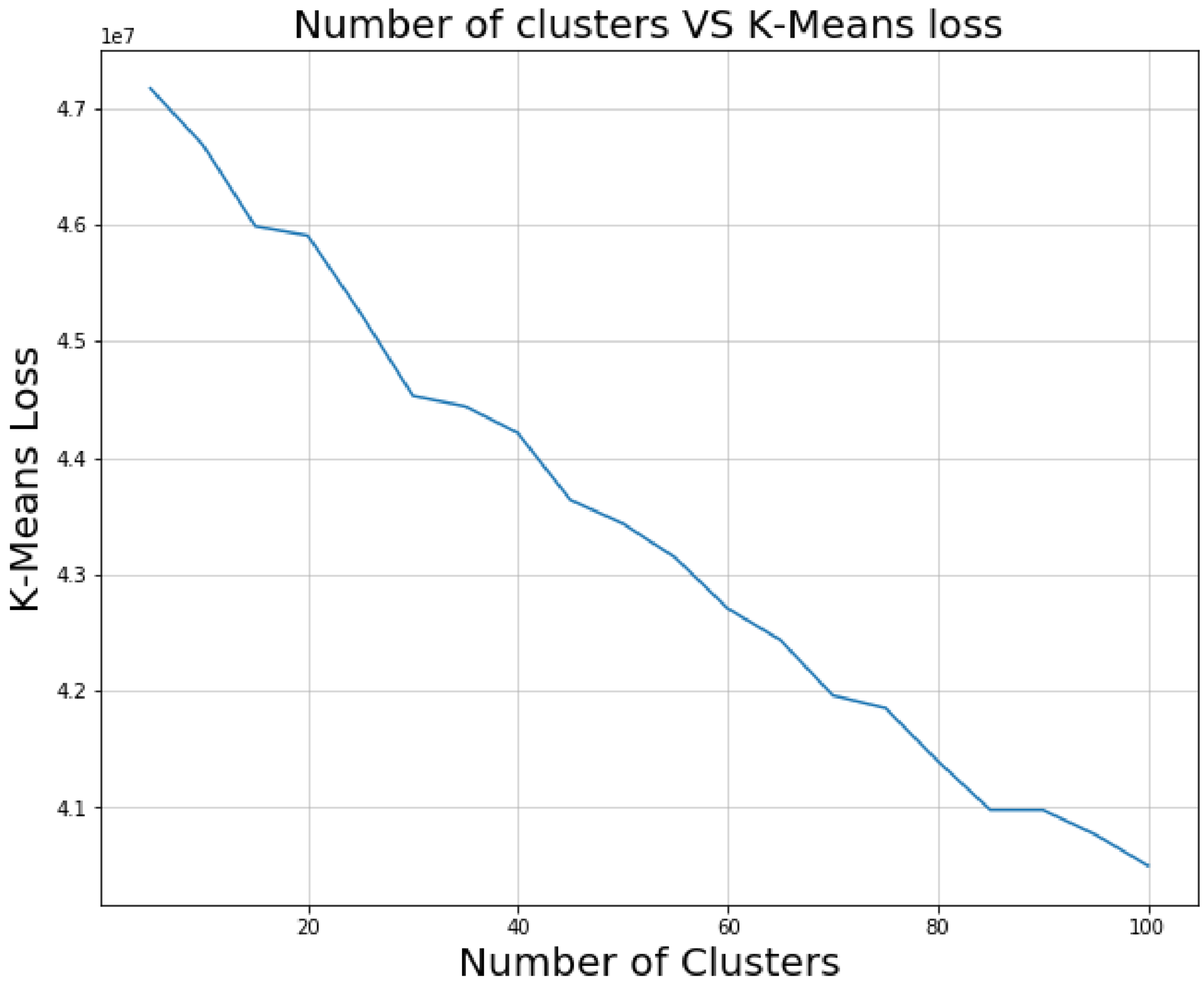
In [89]:

```
clusters = [i for i in range(5, 101, 5)]
loss = []

for k in clusters:
    kmeans = KMeans(n_clusters=k, init = "k-means++", max_iter = 100, n_jobs = -1).fit(Data_BoW_Std)
    loss.append(kmeans.inertia_)
    print("Loss = "+str(kmeans.inertia_)+" for number of clusters = "+str(k))
```

```
Loss = 47166302.51309344 for number of clusters = 5
Loss = 46683033.998859026 for number of clusters = 10
Loss = 45983307.65827241 for number of clusters = 15
Loss = 45904929.340340376 for number of clusters = 20
Loss = 45241843.63682336 for number of clusters = 25
Loss = 44530682.43206152 for number of clusters = 30
Loss = 44440129.81190301 for number of clusters = 35
Loss = 44211087.691596895 for number of clusters = 40
Loss = 43638532.68453365 for number of clusters = 45
Loss = 43435494.04941922 for number of clusters = 50
Loss = 43142392.832236886 for number of clusters = 55
Loss = 42706547.89267114 for number of clusters = 60
Loss = 42437052.260163665 for number of clusters = 65
Loss = 41960745.961246125 for number of clusters = 70
Loss = 41853228.18844861 for number of clusters = 75
Loss = 41398023.98524521 for number of clusters = 80
Loss = 40980132.24941102 for number of clusters = 85
Loss = 40979784.38578416 for number of clusters = 90
Loss = 40768713.20894561 for number of clusters = 95
Loss = 40502027.59807339 for number of clusters = 100
```

In [90]:
```
plt.figure(figsize = (10, 8))
plt.plot(clusters, loss)
plt.title("Number of clusters VS K-Means loss", fontsize=20)
plt.xlabel("Number of Clusters", fontsize=20)
plt.ylabel("K-Means Loss", fontsize=20)
plt.grid(linestyle='-', linewidth=0.5)
```



From the above graph it can be seen that when number of clusters increased from 85 to 95, the loss has not been reduced much. Hence, we are considering our number of clusters to be 85.

Task 2. Apply K-Means.

In [91]:
```
KMeans_Apply = KMeans(n_clusters=85, init = "k-means++", max_iter = 100, n_jobs = -1).fit(Data_BoW_Std)
```

In [91]:
```
# Cluster_indices = {}
# cnt = 0
# for i in kmeansApply.labels_:
# #     print("Cluster Number= "+str(i)+", corresponding point from dataset = "+str(cnt))
#     cnt += 1
```

In [119]:
```
#"KMeans.labels_" returns an array where each element of an array signifies the cluster number and its corresponding inde
#number signifies the data-point number.
```

In [ ]:
```
# a = np.arange(5,10)
# np.where(a < 8)     # tell me where in a, entries are < 8

# >>>(array([0, 1, 2]),)       # answer: entries indexed by 0, 1, 2

#Intuitively, np.where is like asking "tell me where in this array, entries satisfy a given condition"
```

In [92]:
```
Cluster_indices = {i: np.where(KMeans_Apply.labels_ == i) for i in range(KMeans_Apply.n_clusters)}
#above line can be read as, Cluster_indices is a dictionary where keys are the cluster number and value will be which poi
#there in which cluster number.
```

In [93]:
```
for i in range(10):
    length = len(Cluster_indices[i][0])
    if length > 1:
        print("Cluster "+str(i)+" has length "+str(length))
```

```
Cluster 0 has length 6
Cluster 6 has length 3
```

It has been found that only cluster number- '0', '6' contains more than one point, rest all of the clusters contains one point only. It indicates that except 2 clusters, rest of the clusters contains just one point which means that rest of the clusters are point in itself.

Silhouette Analysis

Assume the data have been clustered via any technique, such as k-means, into k clusters. For each datum i, let a(i) be the average distance between i and all other data within the same cluster. We can interpret a(i) as a measure of how well i is assigned to its cluster (the smaller the value, the better the assignment). We then define the average dissimilarity of point i to a cluster c as the average of the distance from i to all points in c.

Let b(i) be the lowest average distance of i to all points in any other cluster, of which i is not a member. The cluster with this lowest average dissimilarity is said to be the "neighbouring cluster" of i because it is the next best fit cluster for point i. We now define a silhouette:
title
Which can be also written as:

title
From the above definition it is clear that:

title

In [95]:
```
a = silhouette_score(Data_BoW_Std, KMeans_Apply.labels_)
print("Silhouette Score = "+str(a))
```

```
Silhouette Score = 0.15773976790229466
```

Silhouette Score of 0.1577 suggests that clusters are very close to each other and very few of them are overlapping.

TFIDF

In [96]:
```
Data.shape
```

Out[96]:
```
(5000, 12)
```

In [97]:
```
tfidf_vect = TfidfVectorizer(ngram_range = (1, 1))
Data_TFIDF = tfidf_vect.fit_transform(Data["ProcessedText"].values)
```

In [98]:
```
print(type(Data_TFIDF))
print(Data_TFIDF.shape)
```

```
<class 'scipy.sparse.csr.csr_matrix'>
(5000, 9525)
```

In [99]:
```
Data_TFIDF_Std = StandardScaler(with_mean = False).fit_transform(Data_TFIDF)
print(Data_TFIDF_Std.shape)
print(type(Data_TFIDF_Std))
```

```
(5000, 9525)
<class 'scipy.sparse.csr.csr_matrix'>
```

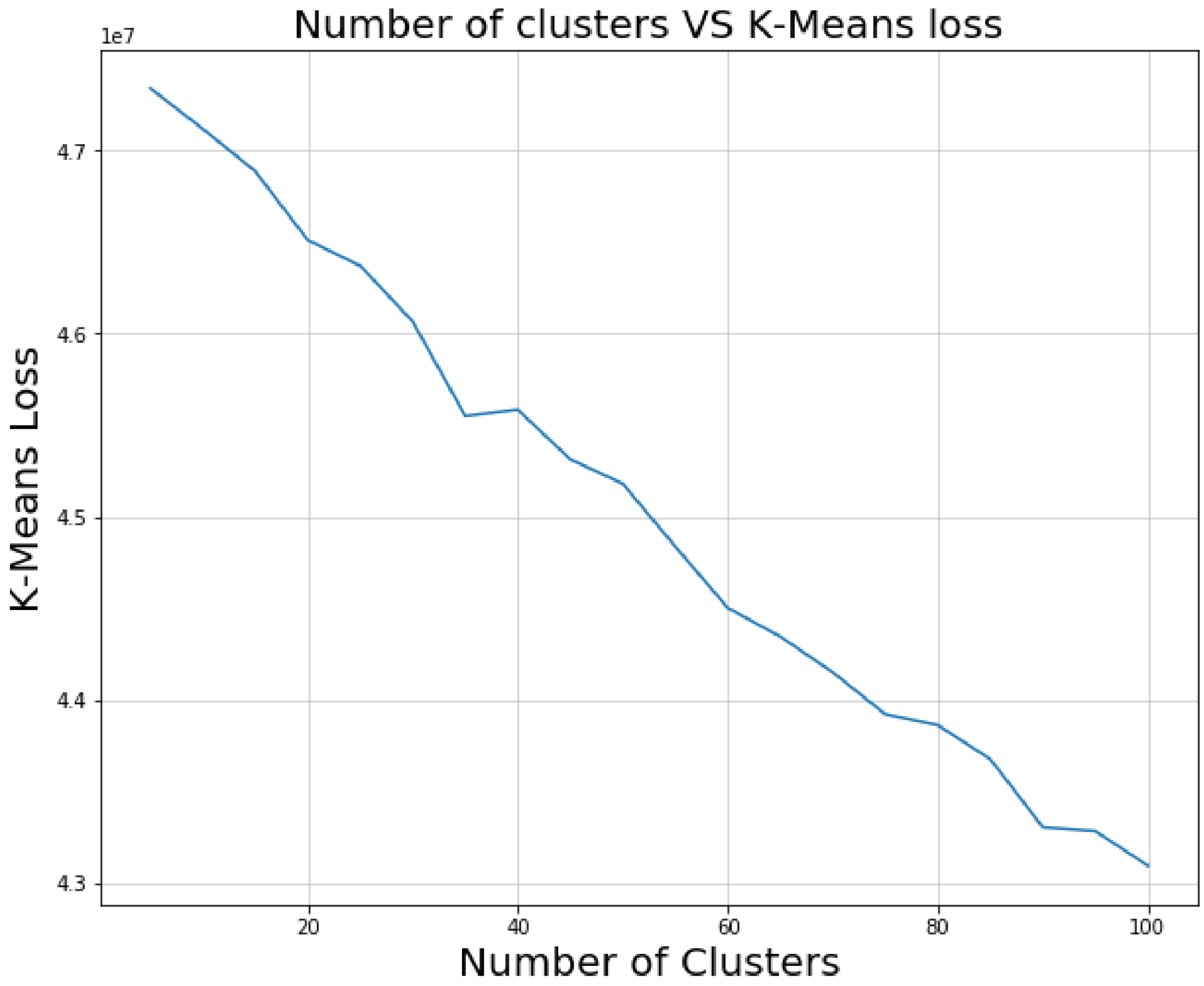
Task 1. Perform Cross Validation on K-Means and use elbow method to find optimal number of clusters.

In [100]:
```
clusters = [i for i in range(5, 101, 5)]
loss = []

for k in clusters:
    kmeans = KMeans(n_clusters=k, init = "k-means++", max_iter = 100, n_jobs = -1).fit(Data_TFIDF_Std)
    loss.append(kmeans.inertia_)
    print("Loss = "+str(kmeans.inertia_)+" for number of clusters = "+str(k))
```

```
Loss = 47339337.986976594 for number of clusters = 5
Loss = 47118184.54616616 for number of clusters = 10
Loss = 46886055.477668636 for number of clusters = 15
Loss = 46510960.707166076 for number of clusters = 20
Loss = 46370556.17121081 for number of clusters = 25
Loss = 46066041.568040736 for number of clusters = 30
Loss = 45550486.274419196 for number of clusters = 35
Loss = 45585696.67229239 for number of clusters = 40
Loss = 45314041.41387355 for number of clusters = 45
Loss = 45181511.34722852 for number of clusters = 50
Loss = 44838057.22835021 for number of clusters = 55
Loss = 44503133.119289204 for number of clusters = 60
Loss = 44346918.24666942 for number of clusters = 65
Loss = 44151681.86486965 for number of clusters = 70
Loss = 43923041.10219855 for number of clusters = 75
Loss = 43863659.16083323 for number of clusters = 80
Loss = 43677825.587510385 for number of clusters = 85
Loss = 43306202.02029817 for number of clusters = 90
Loss = 43285027.13223802 for number of clusters = 95
Loss = 43096827.38681849 for number of clusters = 100
```

In [102]:
```
plt.figure(figsize = (10, 8))
plt.plot(clusters, loss)
plt.title("Number of clusters VS K-Means loss", fontsize=20)
plt.xlabel("Number of Clusters", fontsize=20)
plt.ylabel("K-Means Loss", fontsize=20)
plt.grid(linestyle='-', linewidth=0.5)
```



From the above graph it has been seen that the loss has been increased when number to clusters increased from 35 to 40. Hence, we are considering our number of cluster to be 35.

Task 2. Apply K-Means.

In [120]:
```
KMeans_Apply = KMeans(n_clusters=35, init = "k-means++", max_iter = 100, n_jobs = -1).fit(Data_TFIDF_Std)
```

In [121]:
```
Cluster_indices = {i: np.where(KMeans_Apply.labels_ == i) for i in range(KMeans_Apply.n_clusters)}
```

In [123]:
```
for i in range(35):
    length = len(Cluster_indices[i][0])
    if length > 1:
        print("Cluster "+str(i)+" has length "+str(length))
```

```
Cluster 1 has length 2
Cluster 2 has length 131
Cluster 12 has length 42
Cluster 16 has length 4794
```

It has been found that only cluster number- '1','2','12'and '16' contains more than one point, rest all of the clusters contains one point only. It indicates that except 4 clusters, rest of the clusters contains just one point which means that rest of the clusters are point in itself.

In [124]:
```
a = silhouette_score(Data_TFIDF_Std, KMeans_Apply.labels_)
print("Silhouette Score = "+str(a))
```

```
Silhouette Score = -0.01061621507670845
```

Silhouette Score of -0.0106 suggests that there are few points which are assigned to wrong clusters.

Avg- W2V

In [30]:
```
i = 0
listOfSentences = []
for sentence in Data["ProcessedText"].values:
    subSentence = []
    for word in sentence.split():
        subSentence.append(word)

    listOfSentences.append(subSentence)
```

In [31]:
```
print(Data['ProcessedText'].values[0])
print("\n")
print(listOfSentences[0:2])
print("\n")
print(type(listOfSentences))
```

```
becam interest sooth drink for elder mother who has sensit stomach with recur nausea mani time shes unabl drink anyth o
ther than weak tea water the coconut water has made possibl for her keep thing down and reliev the nauseous feel now ma
ke sure that she has coconut water avail her ani day the week month this certain drink that restor her energi when she
has experienc few day not want eat high recommend this product you friend famili experi symptom mother doe


[['becam', 'interest', 'sooth', 'drink', 'for', 'elder', 'mother', 'who', 'has', 'sensit', 'stomach', 'with', 'recur',
'nausea', 'mani', 'time', 'shes', 'unabl', 'drink', 'anyth', 'other', 'than', 'weak', 'tea', 'water', 'the', 'coconut',
'water', 'has', 'made', 'possibl', 'for', 'her', 'keep', 'thing', 'down', 'and', 'reliev', 'the', 'nauseous', 'feel',
'now', 'make', 'sure', 'that', 'she', 'has', 'coconut', 'water', 'avail', 'her', 'ani', 'day', 'the', 'week', 'month',
'this', 'certain', 'drink', 'that', 'restor', 'her', 'energi', 'when', 'she', 'has', 'experienc', 'few', 'day', 'not',
'want', 'eat', 'high', 'recommend', 'this', 'product', 'you', 'friend', 'famili', 'experi', 'symptom', 'mother', 'do
e'], ['dont', 'normal', 'write', 'review', 'but', 'for', 'orgain', 'felt', 'compel', 'cant', 'stand', 'most', 'meal',
'replac', 'shake', 'protein', 'shake', 'becaus', 'the', 'textur', 'usual', 'seem', 'off', 'when', 'have', 'the', 'tim
e', 'use', 'unflavor', 'protein', 'powder', 'smoothi', 'but', 'for', 'day', 'where', 'that', 'just', 'not', 'possibl',
'orgain', 'has', 'been', 'excel', 'substitut', 'have', 'tri', 'chocol', 'and', 'mocha', 'and', 'both', 'are', 'great',
'tast', 'prefer', 'the', 'mocha', 'becaus', 'feel', 'like', 'get', 'coffe', 'fix', 'for', 'the', 'though', 'this', 'stu
ff', 'caffien', 'free', 'feel', 'energ', 'without', 'the', 'jitter', 'it', 'good', 'breakfast', 'though', 'occassion',
'get', 'upset', 'stomach', 'after', 'drink', 'havent', 'been', 'abl', 'pinpoint', 'whi', 'but', 'it', 'someth', 'keep',
'mind', 'you', 'have', 'sensit', 'stomach', 'also', 'drink', 'orgain', 'workout', 'day', 'half', 'shake', 'beforehand',
'and', 'the', 'other', 'half', 'afterward', 'even', 'though', 'it', 'expens', 'dont', 'drink', 'everi', 'day', 'and',
'month', 'subscript', 'for', 'just', 'has', 'done', 'well', 'for', 'while', 'now', 'for', 'it', 'worth', 'the', 'mone
y', 'have', 'meal', 'replac', 'that', 'easi', 'and', 'tasti']]


<class 'list'>
```

In [32]:
```
w2vModel = gensim.models.Word2Vec(listOfSentences, size=300, min_count=5, workers=4)
```

In [33]:
```
# compute average word2vec for each review.
sentenceAsW2V = []
for sentence in listOfSentences:
    sentenceVector = np.zeros(300)
    TotalWordsPerSentence = 0
    for word in sentence:
        try:
            vect = w2vModel.wv[word]
            sentenceVector += vect
            TotalWordsPerSentence += 1
        except:
            pass
    if TotalWordsPerSentence!= 0:
        sentenceVector /= TotalWordsPerSentence
        sentenceAsW2V.append(sentenceVector)

print(type(sentenceAsW2V))
print(len(sentenceAsW2V))
print(len(sentenceAsW2V[0]))
```

```
<class 'list'>
4999
300
```

In [34]:
```
Data_W2V_Std = StandardScaler(with_mean = False).fit_transform(sentenceAsW2V)
print(Data_W2V_Std.shape)
print(type(Data_W2V_Std))
```

```
(4999, 300)
<class 'numpy.ndarray'>
```

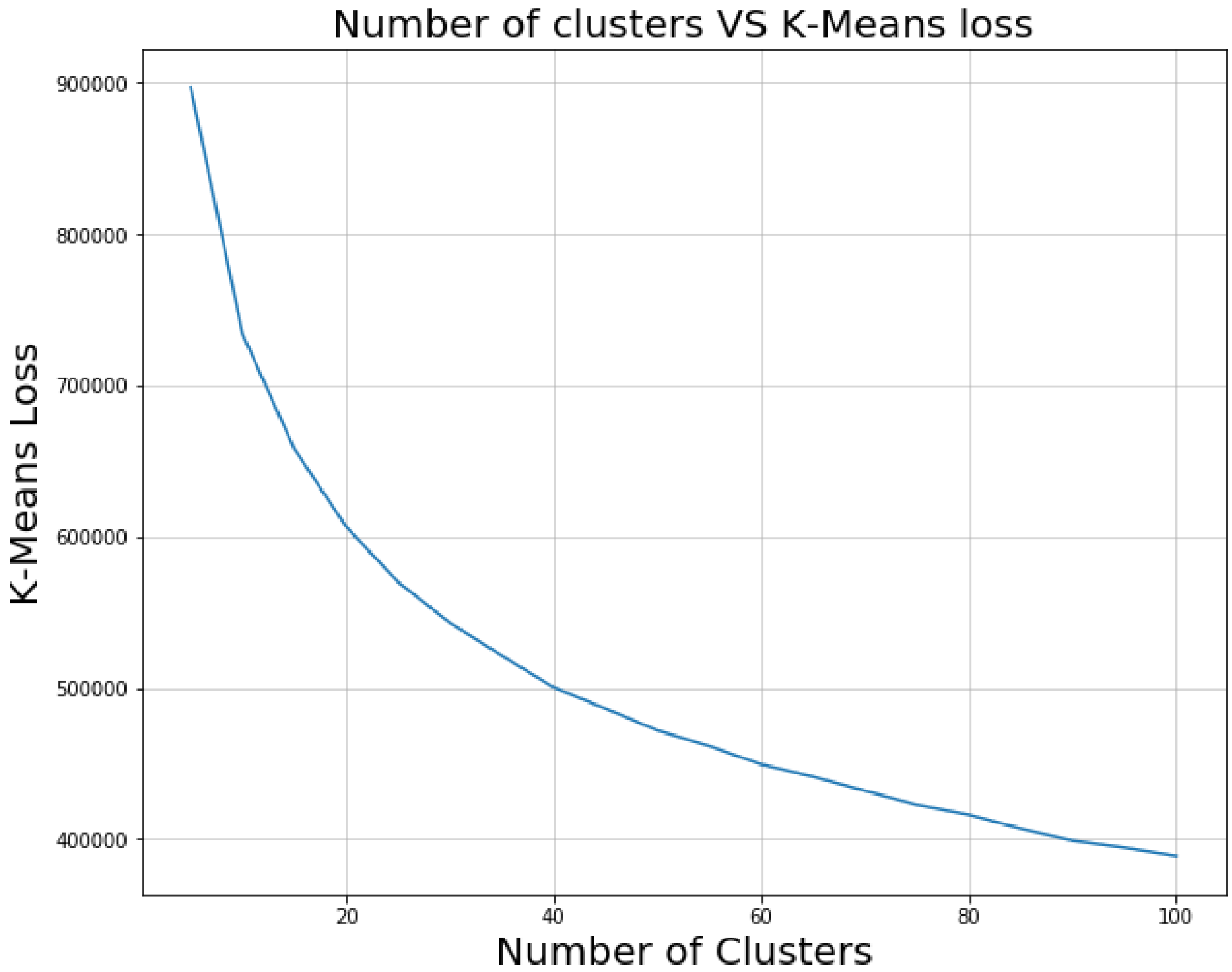
Task 1. Perform Cross Validation on K-Means and use elbow method to find optimal number of clusters.

In [67]:
```
clusters = [i for i in range(5, 101, 5)]
loss = []

for k in clusters:
    kmeans = KMeans(n_clusters=k, init = "k-means++", max_iter = 100, n_jobs = -1).fit(Data_W2V_Std)
    loss.append(kmeans.inertia_)
    print("Loss = "+str(kmeans.inertia_)+" for number of clusters = "+str(k))
```

```
Loss = 897102.3331058134 for number of clusters = 5
Loss = 733950.6608530049 for number of clusters = 10
Loss = 657883.5049771208 for number of clusters = 15
Loss = 606494.6896764381 for number of clusters = 20
Loss = 569952.1442334531 for number of clusters = 25
Loss = 543002.5369246344 for number of clusters = 30
Loss = 521310.5708185359 for number of clusters = 35
Loss = 500271.1697851699 for number of clusters = 40
Loss = 486296.71228270023 for number of clusters = 45
Loss = 471893.9243894333 for number of clusters = 50
Loss = 461507.20059220365 for number of clusters = 55
Loss = 449390.92591947905 for number of clusters = 60
Loss = 441276.24575699464 for number of clusters = 65
Loss = 431972.4620348339 for number of clusters = 70
Loss = 422709.3270709112 for number of clusters = 75
Loss = 416101.8757180755 for number of clusters = 80
Loss = 407036.94468270603 for number of clusters = 85
Loss = 399035.8103079409 for number of clusters = 90
Loss = 394430.87428248 for number of clusters = 95
Loss = 388925.3766700034 for number of clusters = 100
```

In [68]:
```
plt.figure(figsize = (10, 8))
plt.plot(clusters, loss)
plt.title("Number of clusters VS K-Means loss", fontsize=20)
plt.xlabel("Number of Clusters", fontsize=20)
plt.ylabel("K-Means Loss", fontsize=20)
plt.grid(linestyle='-', linewidth=0.5)
```



From the above graph it can be seen that when number of clusters equal to 90 then loss started reducing slowly. Hence, we are considering our number of clusters to be 90.

Task 2. Apply K-Means.

In [125]:
```
KMeans_Apply = KMeans(n_clusters=90, init = "k-means++", max_iter = 100, n_jobs = -1).fit(Data_W2V_Std)
```

In [126]:
```
Cluster_indices = {i: np.where(KMeans_Apply.labels_ == i) for i in range(KMeans_Apply.n_clusters)}
```

In [131]:
```
for i in range(90):
    length = len(Cluster_indices[i][0])
    print("Cluster "+str(i)+" has length "+str(length))
```

```
Cluster 0 has length 70
Cluster 1 has length 106
Cluster 2 has length 92
Cluster 3 has length 42
Cluster 4 has length 40
Cluster 5 has length 63
Cluster 6 has length 26
Cluster 7 has length 98
Cluster 8 has length 99
Cluster 9 has length 54
Cluster 10 has length 59
Cluster 11 has length 72
Cluster 12 has length 42
Cluster 13 has length 119
Cluster 14 has length 70
Cluster 15 has length 110
Cluster 16 has length 95
Cluster 17 has length 39
Cluster 18 has length 60
Cluster 19 has length 63
Cluster 20 has length 31
Cluster 21 has length 29
Cluster 22 has length 35
Cluster 23 has length 23
Cluster 24 has length 68
Cluster 25 has length 8
Cluster 26 has length 27
Cluster 27 has length 131
Cluster 28 has length 65
Cluster 29 has length 113
Cluster 30 has length 29
Cluster 31 has length 6
Cluster 32 has length 66
Cluster 33 has length 68
Cluster 34 has length 121
Cluster 35 has length 135
Cluster 36 has length 48
```

```
Cluster 37 has length 100
Cluster 38 has length 46
Cluster 39 has length 59
Cluster 40 has length 84
Cluster 41 has length 86
Cluster 42 has length 98
Cluster 43 has length 91
Cluster 44 has length 19
Cluster 45 has length 30
Cluster 46 has length 24
Cluster 47 has length 27
Cluster 48 has length 42
Cluster 49 has length 24
Cluster 50 has length 41
Cluster 51 has length 74
Cluster 52 has length 37
Cluster 53 has length 25
Cluster 54 has length 92
Cluster 55 has length 79
Cluster 56 has length 37
Cluster 57 has length 105
Cluster 58 has length 20
Cluster 59 has length 112
Cluster 60 has length 68
Cluster 61 has length 45
Cluster 62 has length 27
Cluster 63 has length 51
Cluster 64 has length 23
Cluster 65 has length 53
Cluster 66 has length 26
Cluster 67 has length 18
Cluster 68 has length 31
Cluster 69 has length 64
Cluster 70 has length 38
Cluster 71 has length 15
Cluster 72 has length 64
Cluster 73 has length 62
Cluster 74 has length 28
Cluster 75 has length 21
Cluster 76 has length 12
Cluster 77 has length 89
Cluster 78 has length 81
```

```
Cluster 79 has length 68
Cluster 80 has length 39
Cluster 81 has length 20
Cluster 82 has length 51
Cluster 83 has length 41
Cluster 84 has length 35
Cluster 85 has length 58
Cluster 86 has length 29
Cluster 87 has length 39
Cluster 88 has length 2
Cluster 89 has length 27
```

Length of each cluster has been printed above.

In [129]:
```
a = silhouette_score(Data_W2V_Std, KMeans_Apply.labels_)
print("Silhouette Score = "+str(a))
```

```
Silhouette Score = 0.078508037734488857
```

Silhouette Score of 0.0785 suggests that many clusters overlap each other.