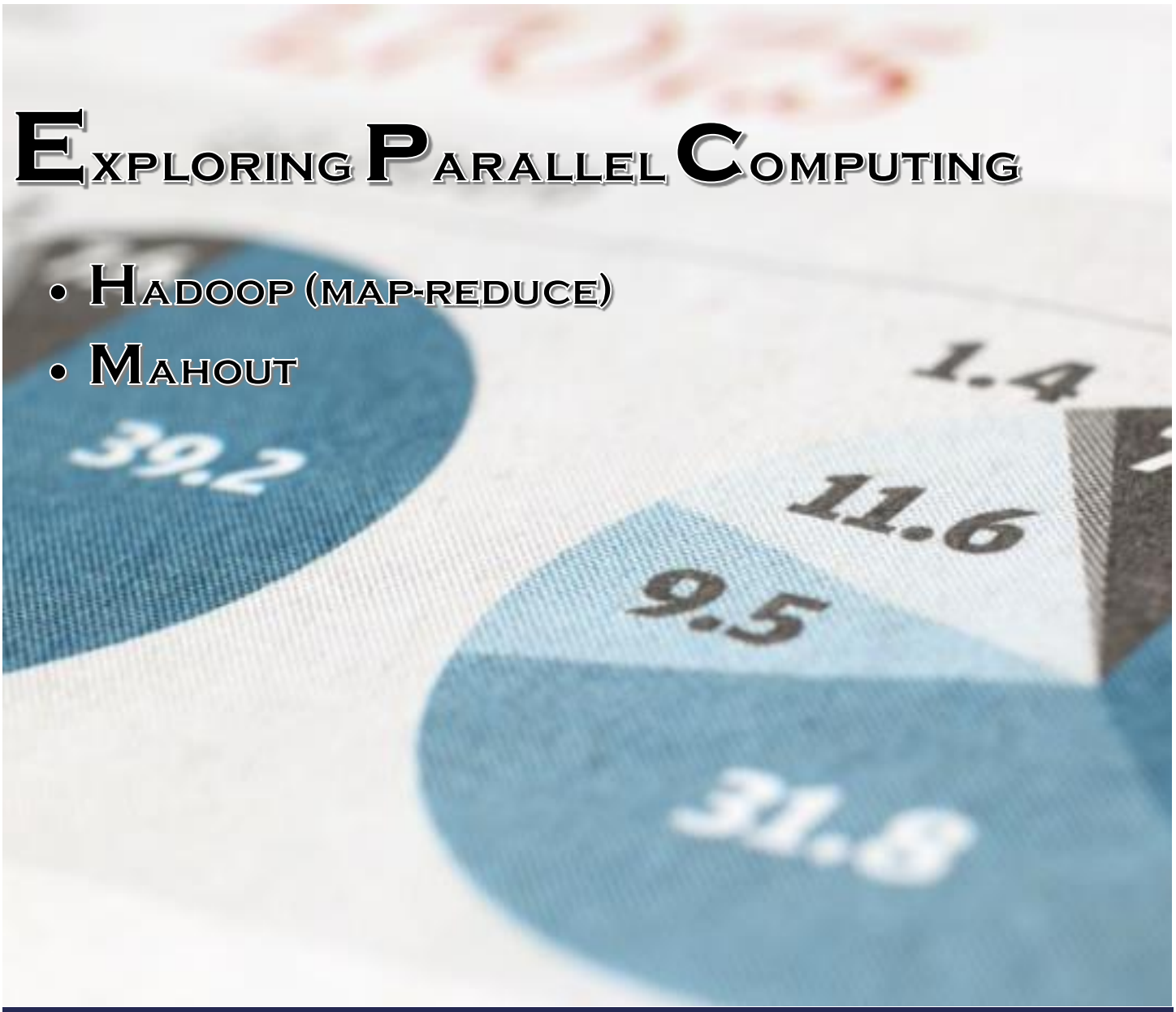


Parallel And Distributed Computing

Fall 2014 Assignment 4 : Report

Vijay Manoharan 12/6/14 50097716

TABLE OF CONTENTS

PART1 HADOOP

INTRODUCTION: SEQUENTIAL BUCKET SORT

ALGORITHM:

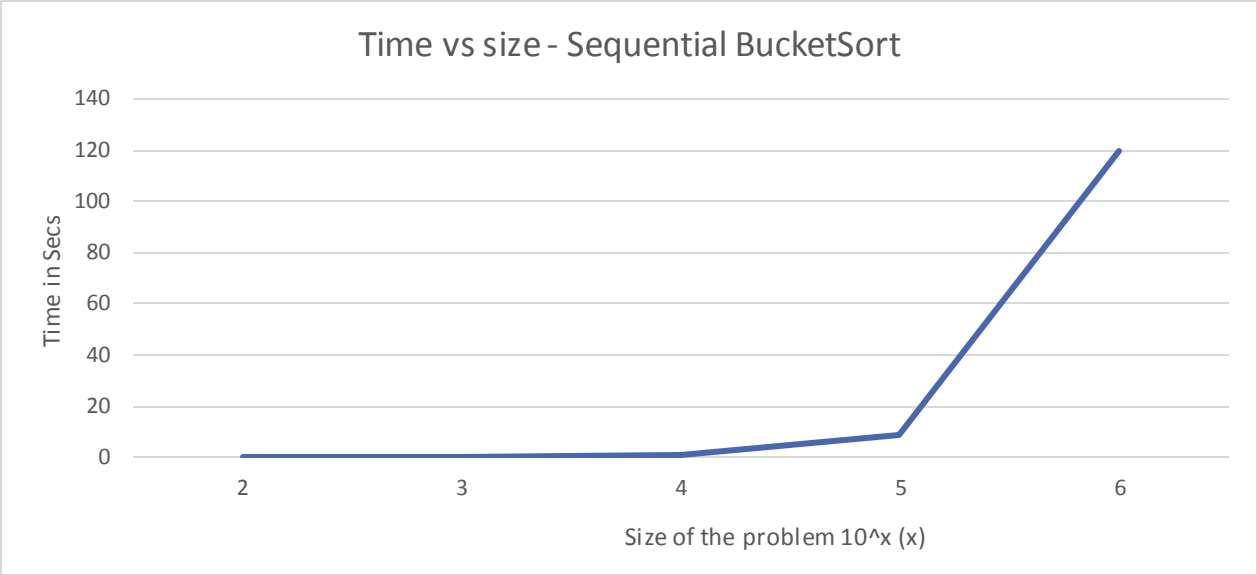
Problem: To sort the array of n number using bucket sort.

The input is taken from command line as "./sequential n Bucket_numbers". The bucket size is the number of buckets we make and n is the number of values in the array for the problem, for our convention we call that as "size of the problem".

- The input is normal distributed number between 1 and size of the problem.
- The algorithm uses the following to sort the values.
    - Combination of Quick and selection sort called "quickSelect". This does partially sorting in O(n) time complexity.
    - For sorting values inside each bucket we use java built in sort() for arrays.
- Therefore the entire complexity will be O(n)+O(nlogn) which is equal to O(nlogn)
- Distribute the numbers uniformly into all the buckets. Say we have 100 size and 4 buckets then we must divide the 25 values in each bucket.
- For this we must range of values for each bucket called pivots. The bucket one will the values from pivot(0) to pivot(1) and the last bucket i.e., say nth bucket will have the values from pivot(n) to pivot(n+1).
- The values in the buckets are assigned by using QuickSelect.
- Each bucket is sorted one by one using merge sort function.
- Finally the sorted values are printed, and stored.

TABULATION AND GRAPH

Size of the problem(10^x) x	sequential running
2	0.008
3	0.039
4	0.304
5	1.04
6	8.693
7	119.473



The number buckets is set to 10.

The comparison is made by varying the input size (100 to 1,000,000) and bucket size in the order of 1,2,4 and 8. The following results are obtained and tabulated. The program is ran locally in my system and results are obtained.

OBSERVATION FROM THE RESULTS

It is seen that the as the size increase the computation time also increases. We can say that the execution time is directly proportional to the size of the problem.

INTRODUCTION: BUCKET SORT USING HADOOP MAP-REDUCE

The program is to explore the parallelism of a simple sorting algorithm and analyze.

Given problem:

To sort a random floating type numbers using Bucket Sort method.

Objective: Find the computational time for the given size of the problem.

Problem size is number of values to be sorted and it is in the form of 10^x (1<x<8).

Input = Problem size;

Algorithm:

Step1: Generate a 'size' random numbers of floating type. The Random number are generated using `randomNumberGeneratorNormal()`, which uses Gaussian function provided in document specification.

Step2: Spilt the entire array into buckets evenly such that if size=n and buckets =m, then each bucket will have n/m floating values. The main intension of splitting into bucket is to implement divide and conquer to solve the problem, so that the problem can be done efficiently done.

And also, the values assigned to each buckets is within a defined range. Say buckets 0 will contain values that are less that bucket 1. Similarly the bucket n-1 will contain values that are greater than of the bucket n-2 and lesser than values in bucket n.

Step3. The values in the bucket are written in to a File, with file name "InputFile<#bucket_number>.txt" and the buckets-files are kept inside a folder "MapInput".

Step4. The will sort a bucket independently and return the sorted content of the file to the Reducer. The input path to the mapper is the "MapInput". This done in order to run one mapper for each Buckets. The mappers run in parallel.

Step5: The Sorting is done using Arrays.Sort(buckets). (bucket is an array from the file), which is in built java util library.

Step6: The mapper emits the bucket to reduce function, in between the combiner function will sort the buckets based on the first value in each Bucket and pass it to reduce one by one.

Step7: The reduce function writes the values received to the output file "output_sort/part-r-00000".

OVERVIEW OF HOW MAP-REDUCE RUNS.

The map function gets the input key-values in the form of <Object>, <Text> and emit key-values in the form of <Text>, <Text>. The input path to mapper is given, bas ed on the type of inputsplit the number of mapper will be initialized and called. The program uses FileInputSplit, so each mapper will run on separate files in parallel.

Now the mapper will get key-values from the file as key = byte-offset, and values = content of the line. Each line will be a key-value.

Bucket contents are written to a file in a single line separated by " ". This is done so that the mapper will only on Key-value for any size. Therefore, the number of input records to mapper will be number of buckets and number of values to reduce will also be number of buckets, so the processing by reducer and mapper will reduce, thus giving really good computation.

The reducer will get the key-values which is of the form <sorted-bucket>, <random value=1>. The combiner will give values to reducer in a sorted way, therefore the reducer will get bucket0 followed by bucket 1 and so-on so-forth till bucket n. By this, when reducer emits its value it will all be sorted.

OUTPUT FILES

There are two output files that is required for the result.

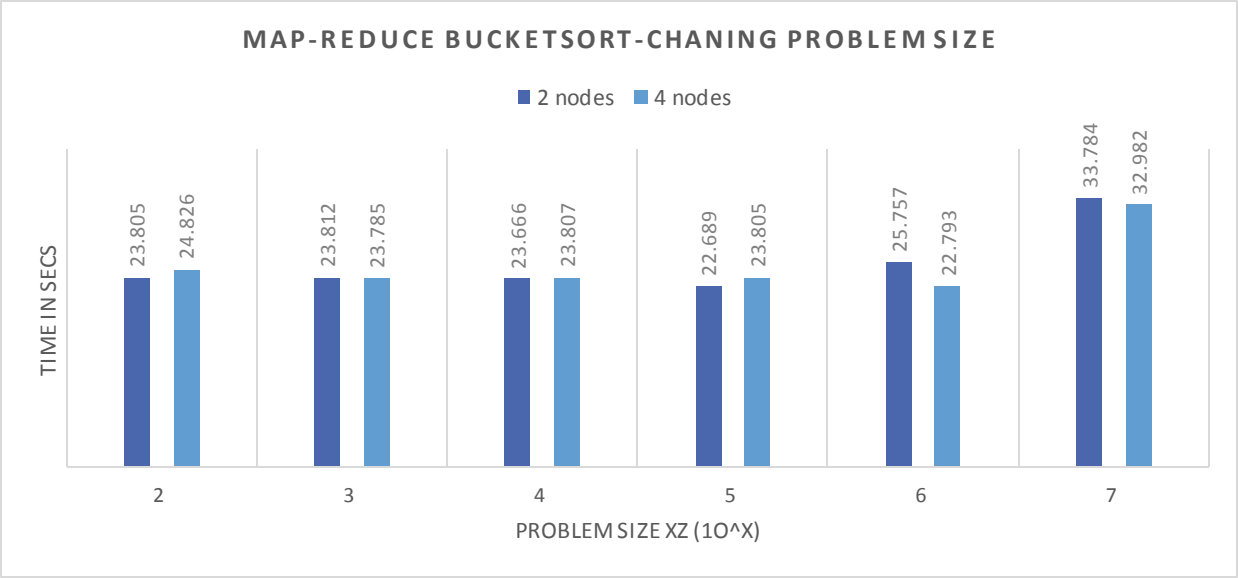
Sorted array file: the File that contains the final sorted array.

Result_bucketSort_size<size>_nodes<numberOfNodes>.out : the file contains the execution result and sequence of the program, the details of map-reduce job and the execution time for the program.

RESULTS

The running time of the program is tabulated for varying size of the problem.

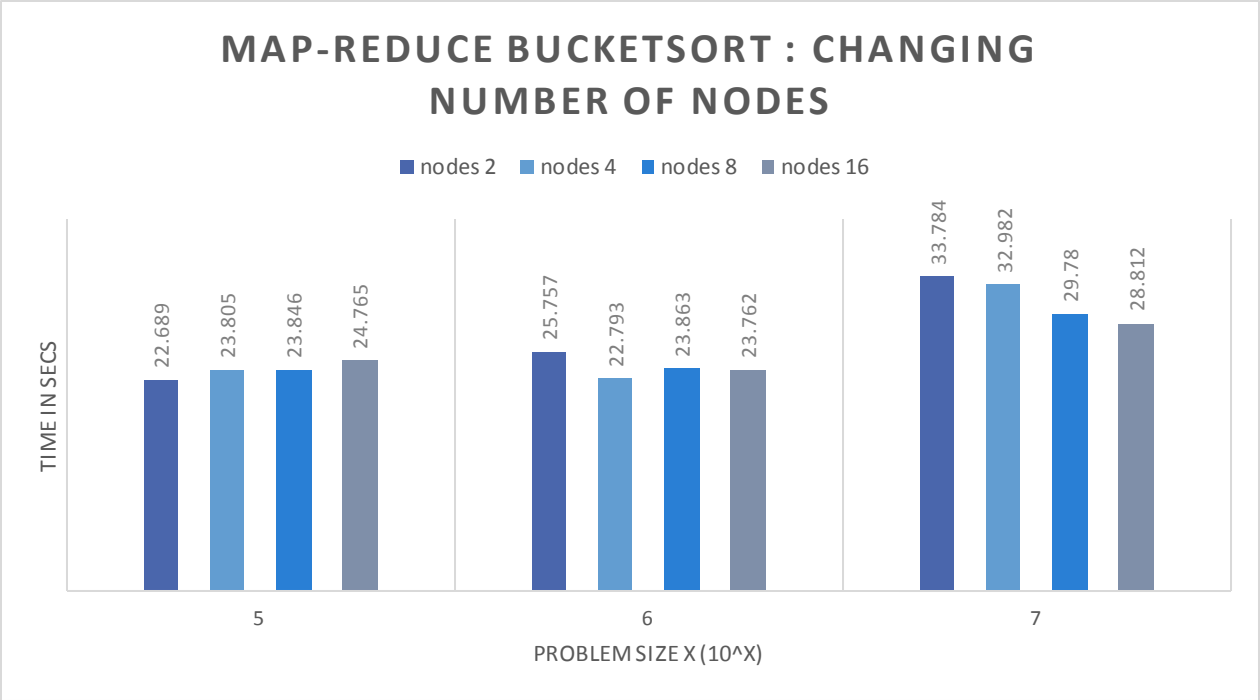
Size of the problem(10^x) x	number of nodes	
	2	**4**
2	23.805	24.826
3	23.812	23.785
4	23.666	23.807
5	22.689	23.805
6	25.757	22.793
7	29.862	32.982



The running time of the program is tabulated for varying number of nodes. It can be seen from the above graph that the value for execution time remains same, this is because of the fact that the job setup and assignment in map reduce task takes a constant amount of time. As the value of the size increase beyond the 10^6, an considerable significance is noticed and there is slight increase in the time.

Size of the problem(10^x) x	number of nodes			
	2	4	8	16
5	22.689	23.805	23.846	24.765
6	25.757	22.793	23.863	23.762
7	33.784	32.982	29.78	28.812

It can be seen from the result that there is slightly better performance as the number of nodes are increased. This is only noticed for larger size of the problem. For Smaller size of the problem the execution time is constant and it is almost the same.



COMPARISON BETWEEN SEQUENTIAL AND PARALLEL BUCKETSORT

The results of the sequential and Parallel BucketSort infers that the sequential time for the problem with smaller size performs better than the parallel size.

Because: Hadoop Map-Reduce is only optimal for problem with larger size and few in numbers. Not with smaller size values and large numbers. Therefore performance for larger problem size the hadoop map-reduce has a better scaling and execution time is better.

Also for job division time and assignment takes more time in hadoop for smaller problem.

PERFORMANCE ON LOAD BALANCE

There will be considerable performance impact when the load is balanced,

- If used an unbalanced data set there will a case where the final solution after sorting the buckets must be sorted again.
- The execution time will depend on the largest size of unbalanced load. This will determine the scaling factor not the number of parallelism used.

The important results that is asked written in the tabulation above.

Comparison between all the parallelism,

Program	Number of cores and device	Problem size	Time
Sequential	1 node 1 task-per-node 0 GPU	100,000	0.019233
Sequential	1 node 1 task-per-node 0 GPU	1,000,000	0.215815
GPU	1 node 1 task-per-node 1 GPU	100,000	0.063069
GPU	1 node 1 task-per-node 1 GPU	1,000,000	0.697863
GPU_MPI	1 node 2 task-per-node 2 GPU	100,000	2.388329
GPU_MPI	2 node 4 task-per-node 2 GPU	100,000	2.521338
GPU_MPI	2 node 4 task-per-node 2 GPU	1,000,000	4.39448
MPI	2 node 4 task-per-node 0 GPU	1,000,000	0.044095
OpenMP	1 node 8 task-per-node 0 GPU	1,000,000	0.101885

From the results of above table and also the tabulation made for hadoop, it can be said that the GPU parallelism performs better. As the problem size really small but using a larger problem size 100GB or more Hadoop map reduce will definitely perform better in terms of scalability and execution.

PART 2 MAHOUT

Objective of the part of the program is to run the classification for give tweets, using two approaches and analyze and compare the results..

1. Mahout- Naïve Bayes classification
2. Mahout- Kmeans clustering

1. MAHOUT- NAÏVE BAYES CLASSIFICATION

The data is given, is in the of sequence files. The job that is required to be done is run the command line arguments to configure mahout. Followed by training the classifier system with the training data set and testing the system with test data set.

The data set is split into training and test data by running the following command:

$mahout split -i <inputdataSet> --trainingOutput <train data path> --testOutput <test data path> --<options>

For running the mahout naïve bayes classification on train data set, the following command is used. This is done to train the classifier, once the classifier is trained we must test for accuracy of the classifier.

$mahout trainnb -i <path- training data> -el -li labelindex -o model -ow -c

To test using Train Data Set.

$mahout testnb -i <train vectors path> -m model -l labelindex -ow -o <output path> -c

For running the mahout naïve bayes classification on test data set.

$mahout testnb -i <path-test-vectors> -m model -l labelindex -ow -o <output path> -c.

Once the job is completed the output file will get the results.

2. MAHOUT-KMEANS CLUSTERING

We are using the same collection of data, and running the clustering. The clustering is not been trained or tested, it is only classified based on the kmeans algorithm. The algorithm takes some random k cluster points to start.

It takes new data set every time and computes the cluster to which the data set belongs to. To get more accuracy the number of iteration are set more. The number of clusters is equal to number of classification we want.

RUNNING K-MEANS USING MAHOUT

The following steps are required to the clustering result of k-means.

Step1 : Get a sequence file, which is already given to us in the tweet data set.

Step2: Parse the sequence in the form of tfidf-vector format by running the command:

$ mahout seq2sparse -i <path: sequence-data> -o <path: output-parsed-vector> --maxDFPercent 85 –namedVector

Step3: Run Kmeans using mahout on the data set by the command:

$mahout kmeans -i <input vector patht+/fidf-vectors/> -c <cluster-path> -o <output path:clusters>

-dm org.apache.mahout.common.distance.CosineDistanceMeasure -x <iterations> -k <number of clusters> -ow --clustering –cl

Step4: Extract the result of classification using cluster Dump method, this will give you how and is clustered. The result of this stored in separate file that is attached to submission.

$mahout clusterdump -i <kmeans-output-path+/clusters--final -d <Kmean-input-path+dictionary.file-0> -dt sequencefile -b 100 -n 20 --evaluate -dm org.apache.mahout.common.distance.CosineDistanceMeasure --pointsDir <kmeans output path+/clusteredPoints" -o <dump output file path>*

Step5: Extracting the clustering details in the form of cluster key-id to the values that is clustered.

$mahout seqdumper -i <kmeans-output-path+/clusteredPoints/part-m-00000> > <output file>

By analyzing the details of the final produced file, which is the clustered point details we can infer about the accuracy and a precision.

To parse the details of the clustering points, the following perl program is used.

Perlparse.

Which must be modified for each classification type : apparel, event, art, camera, health, home and tech, by changing the regular expression to extract the type of value and write to a file.

Once this is done, I am using perlcount.pl to number of time a data of a type is clustered to a cluster-id.

For example, lets run the perl code for the apparel data set and the following output is obtained.

For Camera.

```
[manohara@rush:~/assignment/4_hadoop_Mahout/Mahout]$ perl perlcount.pl count_cam.txt
cluster-ID is 128                numbe 3
cluster-ID is 143                numbe 1
cluster-ID is 146                numbe 9
cluster-ID is 16                 numbe 3
cluster-ID is 307                numbe 6
cluster-ID is 335                numbe 27
cluster-ID is 404                numbe 11
cluster-ID is line               numbe 60
```

1. NAÏVE BAYES

```
==================================================
Summary for Test data
--------------------------------------------------
Correctly Classified Instances       :      138    73.4043%
Incorrectly Classified Instances     :       50    26.5957%
Total Classified Instances           :      188
==================================================
Confusion Matrix
--------------------------------------------------
a       b       c       d       e       f       g       <--Classified as
31      0       1       0       1       0       1       | 34    a    = apparel
0       16      0       4       2       0       3       | 25    b    = art
0       1       22      1       0       0       0       | 24    c    = camera
0       0       0       19      6       1       0       | 26    d    = event
1       2       1       2       11      1       1       | 19    e    = health
2       4       1       1       2       20      1       | 31    f    = home
2       3       2       0       1       2       19      | 29    g    = tech
==================================================
Statistics
--------------------------------------------------
Kappa                           0.5777
Accuracy                        73.4043%
Reliability                     63.481%
Reliability (standard deviation)        0.2859
================================================================================
Summary for train data
--------------------------------------------------
Correctly Classified Instances       :      315    97.2222%
Incorrectly Classified Instances     :        9     2.7778%
Total Classified Instances           :      324
==================================================
Confusion Matrix
--------------------------------------------------
a       b       c       d       e       f       g       <--Classified as
65      0       0       0       0       0       0       | 65    a    = apparel
0       38      0       0       0       1       1       | 40    b    = art
0       0       35      0       1       0       0       | 36    c    = camera
0       0       0       35      0       0       0       | 35    d    = event
0       0       0       0       36      0       0       | 36    e    = health
1       2       0       0       0       23      0       | 26    f    = home
0       1       1       0       0       1       83      | 86    g    = tech
==================================================
Statistics
--------------------------------------------------
Kappa                           0.9164
Accuracy                        97.2222%
Reliability                     84.6494%
Reliability (standard deviation)        0.3442
```

2. KMEANS CLUSTERING

	128	143	146	16	307	335	404	total
Art	10	6	22	3	2	22	0	65
Apparel	8	4	23	10	11	43	0	99
Camera	3	1	9	3	6	27	11	60
Event	14	3	10	8	3	23	0	61
Home	6	1	9	8	2	29	2	57
Health	5	3	8	3	8	28	0	55
Tech	10	6	15	9	9	51	15	115
Total	56	24	96	44	41	223	28	512

OBSERVATION OF RESULTS

- It can noticed from the results that the clustering the data using kmeans classification gives really poor results. It is notice that for a single cluster with id 335 there are 223 data set have been clustered. Thus most of the data set clustered to a cluster-id is classified wrongly. Thus the accuracy can calculated with approximately from 25-40% only.
- Another reason for the poor performance is because of the random initial random points that is chosen for cluster-id, which is random.
- The accuracy can be increased by increasing the number of iterations.
- And also it is difficult to identify correctly which data set belongs to what cluster. Only means to classy is using cosine distance measure and classify the data set based on the proximity to the cluster.

- Naïve bayes classification produces a better result with an accuracy range of 65% to 74% for the test data set.
- Naïve bayes is a classification techniques which is trained and tested for performance.
- Naïve bayes classification can be improved by training the data with large data set, and providing the error back propagation by feedbackward process.