International Journal for Research in Engineering Application & Management (IJREAM)
ISSN : 2454-9150 Vol-03, Issue 04, May 2017

K-Nearest Neighbor Approach Based On Map Reduce for Big Data Classification

[1]Madhavi Bhamare, [2]Suvidha Patil. [3]Kalpita Kuwar, [4]Sampresha Shinde

[1,2,3,4]UG Student, Department Of Computer Engineering, Late. G.N. Sapkal Collage of Engineering, Nashik, Maharashtra, India.

[1]bhamaremadhavi@gmail.com , [2]suvidha.patil114@gmail.com, [3]kuwarkalpita@gmail.com, [4]sampreshashinde@gmail.com

Abstract **The k-Nearest Neighbor classifier is one of the most well known methods in data mining because of its effectiveness and simplicity. Due to its way of working, the application of this classifier may be restricted to problems with a certain number of examples, especially, when the runtime matters. However, the classification of large amounts of data is becoming a necessary task in a great number of real-world applications. This topic is known as big data classification, in which standard data mining techniques normally fail to tackle such volume of data. In this contribution we propose a Map Reduce-based approach for k-Nearest neighbor classification.**

Keywords: Big data, Data mining, Similarity-based machine learning, Sparsification, Supervised normalized cut.

I. INTRODUCTION

The k-Nearest Neighbor algorithm (k-NN) is considered one of the ten most influential data mining algorithms. The classification of big data is becoming an essential task in a wide variety of fields such as biomedicine, social media, marketing, etc. The recent advances in data gathering in many of these fields has resulted in an inexorable increment of the data that we have to manage. The volume, diversity and complexity that bring big data may hinder the analysis and knowledge extraction processes. The MapReduce framework highlights as a simple and robust programming paradigm to tackle large-scale datasets within a cluster of nodes. MapReduce is a very popular parallel programming paradigm that was developed to process and/or generate big datasets that do not fit into a physical memory. Characterized by its transparency for programmers, this Framework enables the processing of huge amounts of data on top of a computer cluster regardless the underlying hardware or software. This is based on functional programming and works in two main steps: the map phase and the reduce phase. In a MapReduce program, all map and reduce operations run in parallel. First of all, all map functions are independently run. Meanwhile, reduce operations wait until the map phase has finished. Then, they process different keys concurrently and independently. Note that inputs and outputs of a MapReduce job are stored in an associated distributed file system that is accessible from any computer of the used cluster.

Several leading machine learning techniques for classification and clustering such as the K-nearest neighbor algorithm, Support Vector Machines (SVMs). SEVERAL leading machine learning techniques for classification and clustering such as the K-nearest neighbor algorithm, variants of supervised normalized cut or support vector machines with Gaussian RBF kernels use as input pairwise similarities. The application of similarity-based algorithms to large-scale data sets is challenging because the number of similarities grows quadratic ally as a function of the number of objects in the data set. Several scarification approaches known to date, have been applied to reduce the number of non-zero entries in the similarity matrix with minimal effect on specific matrix properties. These approaches, however, have to generate the full set of pairwise similarities in advance and thus take at least quadratic time. In this paper, we propose a novel methodology called sparse computation that overcomes the computational burden of computing all pairwise comparisons between the data points by generating only the relevant similarities. Hence, not only is the resulting matrix sparse but also the computation itself is linear in the number of resulting non-zero entries. The relevant similarities are identified by projecting the data points onto a low-dimensional space in which the concept of grid neighborhoods is employed to devise groups of objects with potentially high similarity. Once the relevant pairs of objects have been identified, their similarity is computed in the original space. This differentiates the method from known grid-based clustering algorithms that use the grid neighborhoods to identify the clusters. With our approach, objects can belong to the same grid neighborhood while ending up in different clusters, or conversely, belong to different neighborhoods but still get clustered jointly. The grid dimensionality and grid resolution are the parameters that control the density of the generated similarity matrix. A key aspect of sparse computation is the efficient projection of the data onto a low-dimensional space. Well-known methods such as Principal Component Analysis (PCA) or Multidimensional Scaling (MDS) require excessive running times for large and high-dimensional data sets and are thus not practical for large-scale applications. We suggest generating a low-dimensional space using an algorithm referred to here as approximate-PCA.

II. LITERATURE SURVEY

Sparse computation that generates a sparse similarity matrix which contains only relevant similarities without performing first all pairwise comparisons. Sparse computation that overcomes the computational burden of computing all pairwise comparisons between the data points by generating only the relevant similarities.

The classification algorithms presented in the previous section, require as input pairwise similarities between the objects in the data set. The number of pairwise similarities grows quadratic ally in the size of the data set, which poses a Challenge in terms of scalability. This challenge is shared also by a vast spectrum of clustering approaches, including Greedy agglomerative clustering and expectation maximization algorithms. A great deal of research work has been conducted on scarifying dense matrices. Such efforts consider input graphs or matrices that are dense and apply scarifying algorithms that aim to preserve various matrix properties. Arora et al. describe a simple random-sampling based procedure that generates a sparse matrix whose eigenvectors are close to the eigenvectors of the original matrix. The algorithm considers all non-zero entries of the original matrix and uses the Chernoff-Hoeffding bounds to set some of the entries to zero. The running time of this algorithm is at least proportional to the number of non-zero entries in the input matrix. Spielman and Teng present a graph sparsification algorithm that produces a subgraph of the original, whose Laplacian quadratic form is approximately the same as that of the original graph. Their algorithm has a complexity that is close to being linear in the number of non-zero entries in the original Laplacian. Jhurani recently proposed an algorithm that transforms the original matrix into a sparse matrix with minimal changes to the singular values and the singular vectors corresponding to the near null-space of the original matrix. All these sparsification approaches are based on evaluating all entries of the complete similarity matrix and determining for each entry whether or not to round it to zero. The reading of the entries of the dense similarity matrix alone requires (n2) running time for a data set of n objects. For this reason, these algorithms are not practical for large-scale data sets. By contrast, our approach determines in advance which entries of the similarity matrix are relevant and evaluates only those. Another strategy that aims to reduce the computational burden of computing all pairwise similarities is proposed. They suggest to use initially an approximate similarity measure to subdivide the objects into overlapping subsets. The exact similarities are then only computed between objects that belong to the same subset. This strategy reduces the running time significantly when the computation of the exact similarity measure is expensive, e.g., when the number of attributes is large. In their paper, McCallum et al. study the problem of reference matching in the context of bibliographic citations of research papers. The problem consists of grouping citations that reference the same paper. The approximate distance measure is based on the number of words two citations have in common, which can be computed efficiently using an inverted index. The complexity of this approach is, however, still (n2) because the approximate similarity measure must be computed for all pairs of objects.

III. KNN

In pattern recognition, the *k*-nearest neighbors algorithm (*k*-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the *k* closest training examples in the feature space. The output depends on whether *k*-NN is used for classification or regression:

In *k-NN classification*, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k* is a positive integer, typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.

In *k-NN regression*, the output is the property value for the object. This value is the average of the values of its *k* nearest neighbors.

IV. SYSTEM ARCHITECTURE

In this section, we will discuss working of the proposed system for this platform:

Data pre-processing is done after loading the dataset in this the dataset is normalized. The extension of input file is .csv(Comma separated value).The csv input file divided or splits into number of parts. After that takes a series of key-value Pairs and processes each one of them to generate zero or more key-value pairs and the key-value pairs generated by the mapper are known as intermediate keys. After that here is a combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets

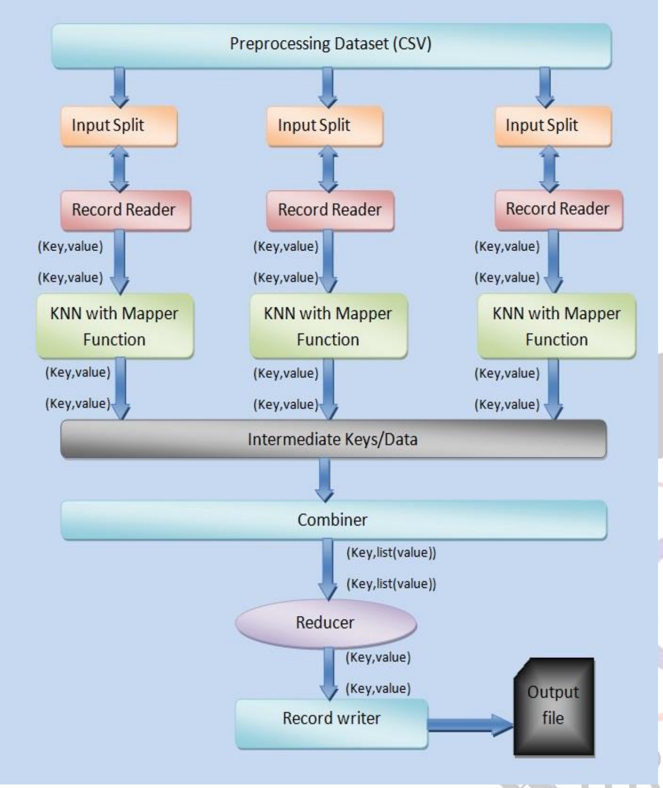


Fig. 1 System Architecture

It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper And the Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing. Once the execution is over, it gives zero or more key-value pairs to the final step. Last one is the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

V. COMPARISM OF SYSTEM

A. Existing System

The classification algorithms presented in the previous section, require as input pairwise similarities between the objects in the data set. The number of pairwise similarities grows quadratic ally in the size of the data set, which poses a challenge in terms of scalability. This challenge is shared also by a vast spectrum of clustering approaches, including greedy agglomerative clustering and expectation-maximization algorithms.

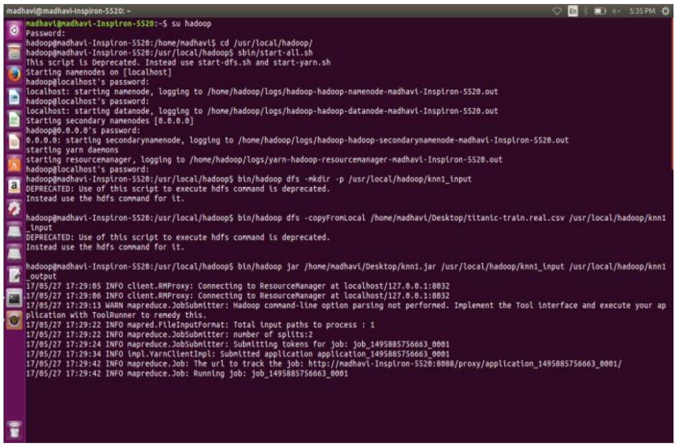
A great deal of research work has been conducted on scarifying dense matrices. Such efforts consider input graphs or matrices that are dense and apply scarifying algorithms that aim to preserve various matrix properties.

B. Proposed System

To improve the performance of similarity based algorithm for large scale data set in data mining. This system generates only the relevant similarities without performing all pairwise comparisons between objects in the data set using K Nearest Neighbor algorithm (KNN).

VI. RESULT ANALYSIS

A.

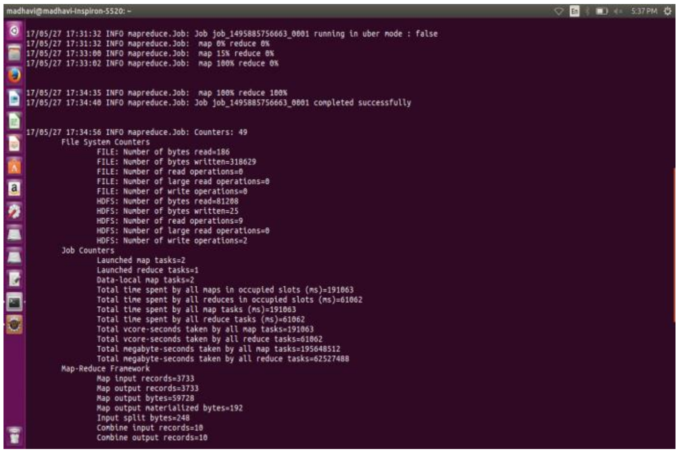


```
madhavi@madhavi-Inspiron-5520:~$ su hadoop
Password:
hadoop@madhavi-Inspiron-5520:/home/madhavi$ cd /usr/local/hadoop/
hadoop@madhavi-Inspiron-5520:/usr/local/hadoop$ sbin/start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [localhost]
hadoop@localhost's password:
localhost: starting namenode, logging to /home/hadoop/logs/hadoop-hadoop-namenode-madhavi-Inspiron-5520.out
hadoop@localhost's password:
localhost: starting datanode, logging to /home/hadoop/logs/hadoop-hadoop-datanode-madhavi-Inspiron-5520.out
Starting secondary namenodes [0.0.0.0]
hadoop@0.0.0.0's password:
0.0.0.0: starting secondarynamenode, logging to /home/hadoop/logs/hadoop-hadoop-secondarynamenode-madhavi-Inspiron-5520.out
starting yarn daemons
starting resourcemanager, logging to /home/hadoop/logs/yarn-hadoop-resourcemanager-madhavi-Inspiron-5520.out
hadoop@localhost's password:
hadoop@madhavi-Inspiron-5520:/usr/local/hadoop$ bin/hadoop dfs -mkdir -p /usr/local/hadoop/knn1_input
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

hadoop@madhavi-Inspiron-5520:/usr/local/hadoop$ bin/hadoop dfs -copyFromLocal /home/madhavi/Desktop/titanic-train.real.csv /usr/local/hadoop/knn1
_input
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

hadoop@madhavi-Inspiron-5520:/usr/local/hadoop$ bin/hadoop jar /home/madhavi/Desktop/knn1.jar /usr/local/hadoop/knn1_input /usr/local/hadoop/knn1
_output
17/05/27 17:29:05 INFO client.RMProxy: Connecting to ResourceManager at localhost/127.0.0.1:8032
17/05/27 17:29:06 INFO client.RMProxy: Connecting to ResourceManager at localhost/127.0.0.1:8032
17/05/27 17:29:13 WARN mapreduce.JobSubmitter: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your ap
plication with ToolRunner to remedy this.
17/05/27 17:29:22 INFO mapred.FileInputFormat: Total input paths to process : 1
17/05/27 17:29:22 INFO mapreduce.JobSubmitter: number of splits:2
17/05/27 17:29:24 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1495885756663_0001
17/05/27 17:29:34 INFO impl.YarnClientImpl: Submitted application application_1495885756663_0001
17/05/27 17:29:42 INFO mapreduce.Job: The url to track the job: http://madhavi-Inspiron-5520:8088/proxy/application_1495885756663_0001/
17/05/27 17:29:42 INFO mapreduce.Job: Running job: job_1495885756663_0001
```

Fig. 2 Start All Daemone

To start all daemons, write command sbin/start-all.sh
To see all daemons start or not, write command jps

B. Mapper And Reducer Processing

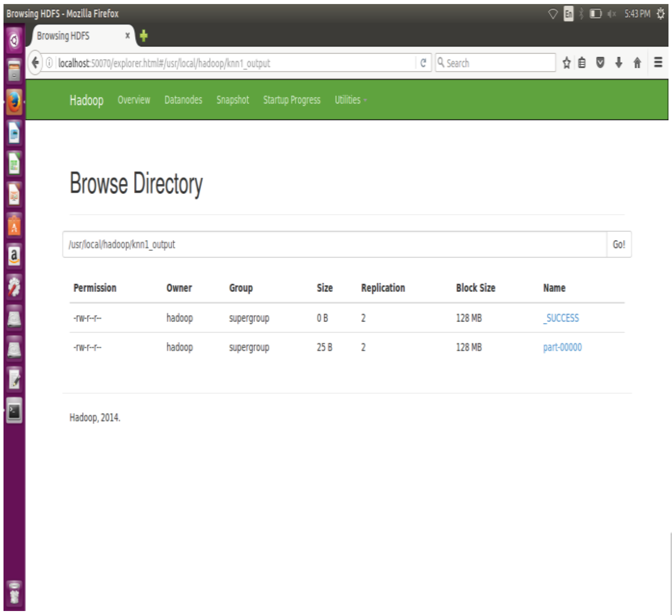


```
17/05/27 17:31:32 INFO mapreduce.Job: Job job_1495885756663_0001 running in uber mode : false
17/05/27 17:31:32 INFO mapreduce.Job:  map 0% reduce 0%
17/05/27 17:33:00 INFO mapreduce.Job:  map 15% reduce 0%
17/05/27 17:33:02 INFO mapreduce.Job:  map 100% reduce 0%


17/05/27 17:34:35 INFO mapreduce.Job:  map 100% reduce 100%
17/05/27 17:34:40 INFO mapreduce.Job: Job job_1495885756663_0001 completed successfully


17/05/27 17:34:56 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=186
                FILE: Number of bytes written=318629
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=81208
                HDFS: Number of bytes written=25
                HDFS: Number of read operations=9
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=2
                Launched reduce tasks=1
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=191063
                Total time spent by all reduces in occupied slots (ms)=61062
                Total time spent by all map tasks (ms)=191063
                Total time spent by all reduce tasks (ms)=61062
                Total vcore-seconds taken by all map tasks=191063
                Total vcore-seconds taken by all reduce tasks=61062
                Total megabyte-seconds taken by all map tasks=195648512
                Total megabyte-seconds taken by all reduce tasks=62527488
        Map-Reduce Framework
                Map input records=3733
                Map output records=3733
                Map output bytes=59728
                Map output materialized bytes=192
                Input split bytes=248
                Combine input records=10
                Combine output records=10
```

Fig. 3 Mapper and Reducer Processing

To check above screenshot shows the mapper and reducer function

C. *Browse Directory at Local Host*


Browsing HDFS - Mozilla Firefox
Browsing HDFS
localhost:50070/explorer.html#/usr/local/hadoop/knn1_output
Search
Hadoop
Overview
Datanodes
Snapshot
Startup Progress
Utilities
5:43 PM

Browse Directory

/usr/local/hadoop/knn1_output Go!

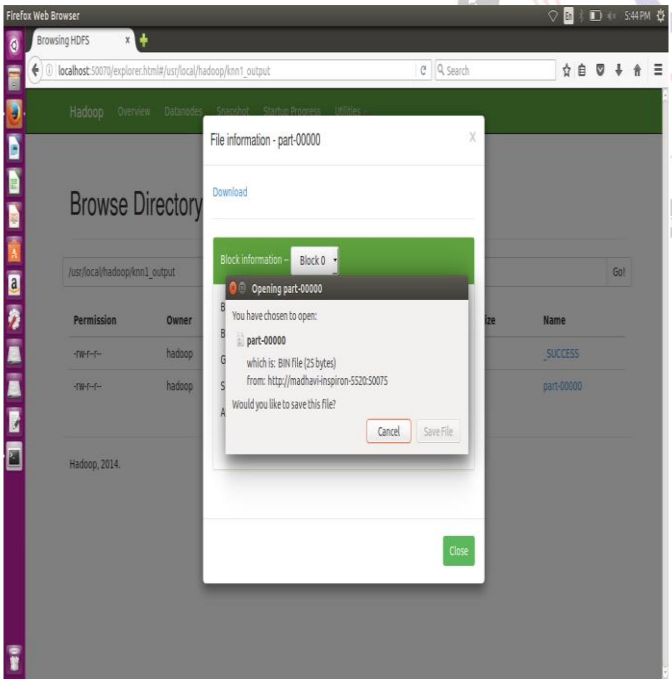
Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	0 B	2	128 MB	_SUCCESS
-rw-r--r--	hadoop	supergroup	25 B	2	128 MB	part-00000

Hadoop, 2014.

Fig. 4 Browse Directory at Local Host

The browse system consist two files, first is success and other is part 00000

D. *Download output file*



File information - part-00000

Download

Opening part-00000

You have chosen to open:

part-00000

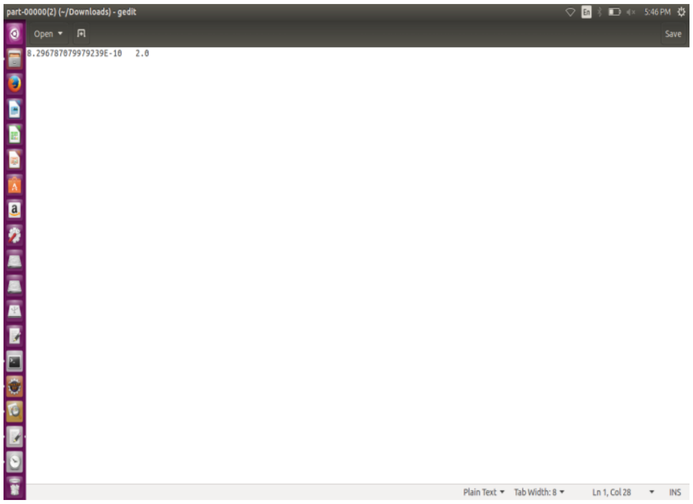
which is: BIN File (25 bytes)
from: http://madhavi-inspiron-5520:50075

Would you like to save this file?

Fig. 5 Download output file

Here we can download the part-00000 output file

E. *KNN Output*



```
8.296787879979239E-10   2.0
```

Fig. 6 KNN Output

This shows the output of ken algorithm

VII. CONCLUSION

In this, we have proposed a MapReduce approach to enable the k-Nearest neighbor technique to deal with large-scale datasets. Without a parallelization, the application of the k-NN algorithm would be limited to small or medium data, especially when low runtimes are a need. The proposed scheme is an exact parallelization of the k-NN model, so that, the precision remains the same and the efficiency has been largely improved.

ACKNOWLEDGMENTS

It gives us great pleasure in presenting the “K-Nearest Neighbor Approach Based On Map Reduce for Big Data Classification” I would like to take this opportunity to thank my internal guide Prof. S. B. Wagh for giving me all the help and guidance I needed. I am really grateful to them for their kind support. Their Valuable suggestion were very helpful

REFERENCES

[1] Dorit S. Hochbaum, Philipp Baumann,“Sparse Computation for Large-Scale Data Mining“IEEE Transactions on Big Data.

[2] T.M. Cover and P.E. Hart, “Nearest neighbor pattern classification,” IEEE Trans. on Information Theory, vol. 13, pp. 21–27, 1967.

[3] D.S. Hochbaum, C.-N. Hsu, and Y.T. Yang, “Ranking of multidimensional drug profiling data by fractional-adjusted bi-partitional scores,” Bioinformatics, vol. 28, pp. i106–i114, 2012.

[4] Y.T. Yang, B. Fishbain, D.S. Hochbaum, E.B. Norman, and E. Swanberg, “The supervised normalized cut method for detecting, classifying, and identifying special nuclear materials,” INFORMS Journal on Computing, 2013.

[5] D.S. Hochbaum, C. Lu, and E. Bertelli, “Evaluating performance of image segmentation criteria and techniques,” EURO Journal on Computational Optimization, vol. 1, pp. 155–180, 2013.

[6] B. Sch¨olkopf and A.J. Smola, Learning with kernels: support vector machines, regularization, optimization, and beyond. Cambridge MA: MIT Press, 2001.

[7] P. Baumann, D.S. Hochbaum, and Y.T. Yang, “A comparative study of leading machine learning techniques and two new algorithms,” 2015, Submitted 2015.

AUTHORS PROFILE

1 st Author Name :- Madhavi Bhamare
Qualification :- Diploma in Computer Engg from Mumbai University, B.E Appear of computer Engineering department from Late G. N. College Of Engineering Savitribai Phule Pune University.

2 nd Author Name:- Suvidha Patil
Qualification:- Diploma in Computer Engg from Government Polytechnic Nasik, B.E Appear of computer Engineering department from Late G. N. College Of Engineering Savitribai Phule Pune University.

3 rd st Author Name :- Kalpita Kuwar
Qualification :- Diploma in Computer Engg from Mumbai University, B.E Appear of computer Engineering department from Late G. N. College Of Engineering Savitribai Phule Pune University.

4 th Author Name : Sampresha Shinde
Qualification :- Diploma in Computer Engg from Mumbai University, B.E Appear of computer Engineering department from Late G. N. College Of Engineering Savitribai Phule Pune University.