 EVALUATING THE USEFULNESS OF SIDE INFORMATION FOR MINING TEXT DOCUMENTS 

##### A PROJECT REPORT

###### ***Submitted by***

##### DIVYA LAKSHMI K (951711205013)

**SANDHYA S (951711205046)**

**VAISHNAVI R (951711205058)**

***in partial fulfillment for the award of the degree***

***of***

**BACHELOR OF ENGINEERING**

*in*

# INFORMATION TECHNOLOGY

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

ANNA UNIVERSITY: CHENNAI 600 025

##### APRIL 2015

ANNA UNIVERSITY: CHENNAI 600 025

**BONAFIDE CERTIFICATE**

Certified that this project report **“EVALUATING THE USEFULNESS OF SIDE INFORMATION FOR MINING TEXT DOCUMENTS”** is the bonafide work of “**DIVYA LAKSHMI K (951711205013), SANDHYA S (951711205046), VAISHNAVI R (951711205058)”** who carried out the project work under my supervision.

**SIGNATURE** **SIGNATURE**

**Dr. T. REVATHI Mrs. S. SAROJA**

**SENIOR PROFESSOR ASSISTANT PROFESSOR**

**HEAD OF THE DEPARTMENT SUPERVISOR**

Department of Information Technology Department of Information Technology

Mepco Schlenk Engg. College, Sivakasi Mepco Schlenk Engg. College, Sivakasi

Virudhunagar Dt. – 626 005 Virudhunagar Dt. – 626 005

Submitted to the Viva-Voce examination held on **/ / .**

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**ACKNOWLEDGEMENT**

First and foremost, we thank the almighty for showering grace and blessings on us for making this project a great success.

We extend our sincere gratitude to our college management and our honourable Principal**. Dr. S ARIVAZHAGAN, M.E., Ph.D.,** for providing sufficient working environment such as systems and library facilities. We also thank him very much for providing us adequate lab facilities, which enable us to complete our project.

We would like to extend our heartfelt gratitude to **Dr.T.Revathi, M.E, PhD,** Senior Professorand Head, Department of Information Technology, Mepco Schlenk engineering college for her valuable guidance and timely encouragements. She put all her valuable experience and expertise in directing, suggesting and supporting us throughout the project to bring out of the best.

We would like to extend our gratitude to**Mrs. J. Angela Jennifa Sujana, M.Tech, (Ph.D)**Assistant Professor, Department of Information Technology, Mepco Schlenk Engineering College for being our Project Coordinator and directing us through our project.

We extend our sincere thanks to **Mrs.S.Saroja, M.E** Assistant Professor, Department of Information Technology, Mepco Schlenk Engineering College for her moral support and guidance throughout the project.

We extend our thanks to all staff members and lab technicians **Mr.S.T.Vairamuthu** and **Mr.R.Raj Kumar**for their moral support in completing our project.

**DEDICATED TO PARENTS, FACULTIES AND FRIENDS**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NUMBER** | **TITLE** | **PAGE NUMBER** |
|  | **ABSTRACT** | vii |
|  | **LIST OF FIGURES** | ix |
|  | **LIST OF ABBREVIATION** | x |
|  | **LIST OF SYMBOLS** | xi |
| 1. | **INTRODUCTION** | 1 |
|  | * 1. OBJECTIVE | 2 |
|  | 1.2.PURPOSE | 3 |
|  | 1.3.SCOPE | 3 |
|  | 1.4.REQUIREMENTS | 6 |
|  | 1.5.SUMMARY | 7 |
| 2. | **LITERATURE SURVEY** | 8 |
|  | 2.1.OVERVIEW | 8 |
|  | 2.2.CLUSTERING | 8 |
|  | 2.2.1.PARTITIONING CLUSTERING | 8 |
|  | 2.2.2.HIERARCHICAL CLUSTERING | 10 |
|  | 2.2.3.DENSITY BASED CLUSTERING | 12 |
|  | 2.3.CLASSIFICATION | 12 |
|  | 2.3.1.NAIVE BAYES CLASSIFIER | 13 |
|  | 2.3.2.DECISION TREE CLASSIFIER | 13 |
|  | 2.3.3.SVM CLASSIFIER | 14 |
|  | 2.4.SUMMARY | 14 |
| 3. | **SYSTEM STUDY** | 15 |
|  | 3.1.OVERVIEW | 16 |
|  | 3.2.PROPOSED SYSTEM | 16 |
|  | 3.3.SUMMARY | 20 |
| 4. | **SYSTEM DESIGN** | 21 |
|  | 4.1.OVERVIEW | 22 |
|  | 4.2.MODULES | 22 |
|  | 4.2.1.PRE-PROCESSING MODULE | 22 |
|  | 4.2.2.INITIAL CLUSTERING MODULE | 23 |
|  | 4.2.3.MINOR 1 MODULE | 24 |
|  | 4.2.4.MINOR 2 MODULE | 24 |
|  | 4.2.5.CLASSIFICATION MODULE | 25 |
|  | 4.3.SUMMARY | 26 |
| 5. | **SYSTEM IMPLEMENTATION** | 27 |
|  | 5.1.OVERVIEW | 28 |
|  | 5.2.CLUSTERING | 29 |
|  | 5.2.1.THE COATES ALGORITHM | 29 |
|  | 5.3.CLASSIFICATION | 29 |
|  | 5.3.1.THE COLT ALGORITHM | 33 |
|  | 5.4.SUMMARY | 33 |
| 6. | **RESULTS AND DISCUSSIONS** | 34 |
|  | 6.1.OVERVIEW | 35 |
|  | 6.2.OUTPUT | 36 |
|  | 6.3.PERFORMANCE ANALYSIS | 36 |
|  | 6.4.SUMMARY | 39 |
| 7. | **CONCLUSION** | 41 |
| 8. | **APPENDICES** | 42 |
|  | 8.1.HARDWARE REQUIREMENTS | 44 |
|  | 8.2.SOFTWARE REQUIREMENTS | 45 |
|  | 8.2.1.ARCHITECTURE OF HADOOP | 45 |
|  | 8.2.2.INTEGRATION OF HADOOP WITH ECLIPSE | 47 |
|  | 8.3.SOURCE CODE | 53 |
|  | 8.3.1.CLUSTERING | 53 |
|  | 8.3.2.CLASSIFICATION | 74 |
| 9. | **REFERENCES** | 87 |

**ABSTRACT**



ABSTRACT

Text mining has become great important in recent years. Text mining is the process of structuring the unstructured textual data. Clustering (aids in mining) refers to process of grouping text documents based on similarity (that is how far documents are relevant to each other). Text documents contains enormous amount of data that are useful for clustering. Text documents contain some side information and it may include history of the document, user behaviour, author of the document, citations and much non textual information. We can’t use all the side information in clustering process because it may either improve the quality or degrade the quality of the clusters. The importance of side information is difficult to analyze. Hence, effort must be taken to find the usefulness of side information. This project defines a principled way to maximize the utilization of side information in clustering text documents. This approach uses classical partitioning method with probabilistic model to perform clustering. This technique can be extended to classification. Our experimental results demonstrate that the proposed method outperforms all the existing methods in both clustering and classification process.

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE NUMBER** | **TITLE** | **PAGE NUMBER** |
| 1.1 | TEXT DOCUMENT WITH SIDE INFORMATION | 3 |
| 1.2 | CLUSTERING WITH SIDE INFORMATION | 5 |
| 1.3. | CLASSIFICATION PROCESS | 5 |
| 2.1 | TYPES OF CLUSTERING | 8 |
| 2.2. | TYPES OF CLASSIFICATION | 13 |
| 4.1. | PRE-PROCESSING MODULE | 23 |
| 4.2. | INITIAL CLUSTER MODULE | 24 |
| 4.3. | MINOR 1 MODULE | 25 |
| 4.4. | MINOR 2 MODULE | 26 |
| 4.5. | CLASSIFICATION | 27 |
| 6.1. | CLUSTERING OUTPUT | 37 |
| 6.2. | CLASSIFICATION OUTPUT | 38 |
| 6.3. | EFFECTIVENESS Vs NO. OF CLUSTERS | 40 |
| 6.4. | EFFECTIVENESS Vs DATA SIZE | 41 |
| 8.1. | TASK IN MAP REDUCE PROGRAM | 46 |
| 8.2. | CREATION OF NEW JAVA PROJECT | 48 |
| 8.3. | ADDITION OF HADOOP JARS TO ECLIPSE | 48 |
| 8.4. | CREATION OF JAVA CLASS IN ECLIPSE | 49 |
| 8.5. | CREATION OF CONFIGURATION FILES | 49 |
| 8.6. | EXPORT OF RESOURCES INTO A JAR FILE | 50 |
| 8.7. | CREATION OF JAR FILE | 50 |
| 8.8. | RUNNING OF DAEMON NODES | 51 |
| 8.9. | CREATION OF INPUT DIRECTORY | 51 |
| 8.10. | EXECUTION OF THE PROGRAM | 52 |

**LIST OF ABBREVIATIONS**

|  |  |  |
| --- | --- | --- |
| 1 | **COATES** | **CO**NTENT AND **A**UXILIARY ATTRIBUTE BASED **TE**XT CLU**S**TERING |
| 2 | **COLT** | **CO**NTENT AND AUXI**L**IARY ATTRIBUTE BASED **T**EXT CLASSIFICATION |
| 3 | **EM** | EXPECTATION **M**AXIMIZATION |
| 4 | **AGNES** | **AG**GLOMERATIVE**NES**TING |
| 5 | **DIANA** | **DI**VISIVE **ANA**LYSIS |
| 6 | **DBSCAN** | DENSITY-**B**ASED **S**PATIAL **C**LUSTERING OF **A**PPLICATIONS WITH **N**OISE |
| 7 | **SVM** | **S**UPPORT **V**ECTOR **M**ACHINE |

**LIST OF SYMBOLS**

**N** – Total number of documents.

**Ti…TN** – ‘N’ text documents.

**Xi** – set of auxiliary variables.

**k** – Total number of clusters formed.

**C1…Ck** – ‘k’ clusters.

**L1…Lk**– Centroids for k clusters.

**t** - Variable used for iteration.

**r** - Auxiliary attribute.

**Gr**– Gini-index for auxiliary attribute r.

**frj** – frequency of records in cluster j for which ‘r’ takes value 1.

**prj**– relative presence of attribute r in cluster j.

**qc(i,t)** – Cluster index for document Ti

**qa(i,t)** – Cluster index for document Ti which has highest posterior probability.

**l1…lN** – ‘N’ labels.

**Ri** – Set of attributes whose gini-index is greater than threshold.

**INTRODUCTION**



**CHAPTER 1**

**INTRODUCTION**

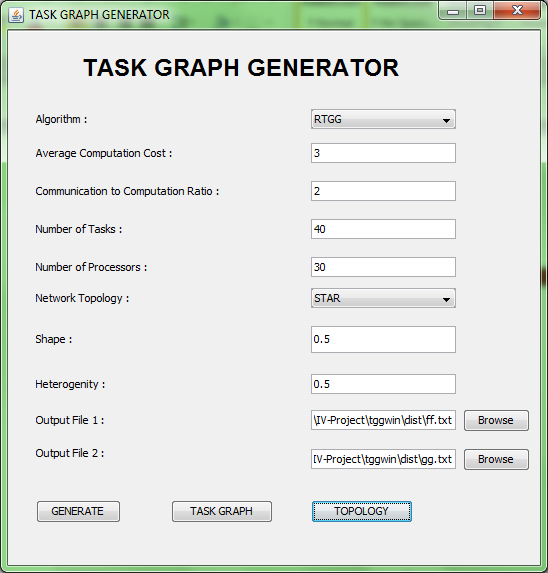
**1.1 OBJECTIVE**

**1.2 PURPOSE**

**1.3 SCOPE**

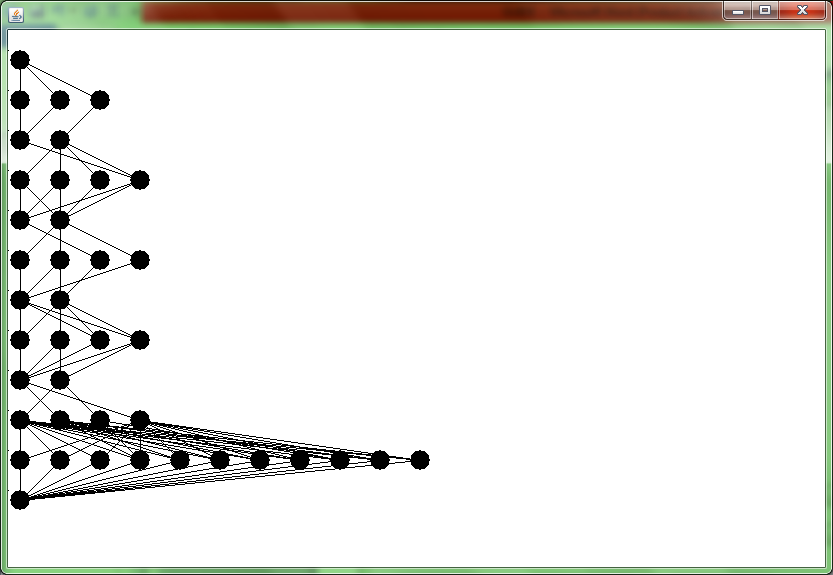
**1.4 REQUIREMENTS**

One of the main requirements is the TASK GRAPH GENERATOR (TGG) which generates Task Graph based on certain algorithms like RTGG, FFT, etc. On giving input as certain parameters (No. of tasks, No. of processors, Topology, etc.,), the TGG will give two output files.



**Fig. 1.1 Input window of Task Graph Generator**

The first file contains number of tasks and number of edges in its first and second lines respectively. It is followed by a tabular form of task and it’s adjacent with communication between them.



**Fig 1.2 Topology of the randomly generated graph**

**Fig. 1.3 Output File 1**

The second file contains the number of processors followed by a matrix like representation which shows the processing time of each task on each processor.

**Fig. 1.4 Output File 2**

**1.5 SUMMARY**

The rest of the document is organized as follows:

Section 2 briefly describes the related works and existing methods that have been performed for the proposed method.

Section 3 briefly explains the steps involved in both clustering and classification in form of text.

Section 4 specifies working modules of the method in pictorial representation.

Section 5 briefly describes the clustering and classification algorithm in form of step by step procedure.

Section 6 shows the results of both classification and clustering algorithms and compute evaluation metrics based on comparison between baseline techniques and our proposed method.

Section 7 contains conclusion and further enhancement.

**LITERATURE SURVEY**



**CHAPTER 2**

**LITERATURE SURVEY**

**2.1 DYNAMIC LEVEL SCHEDULING**

Dynamic-level scheduling is an effective compile-time scheduling technique which is used for inter processor communication overhead when mapping precedence-constrained, communicating tasks onto arbitrarily interconnected processor networks. Scheduling and routing are performed simultaneously to account for limited interconnections between processors, and communications are scheduled along with computations to eliminate shared-resource contention. It encompasses heterogeneous processing environments, and presents two techniques designed to enhance scheduling performance: forward/backward scheduling, and precedence constraint appendage.

2.2 LEVELIZED MIN TIME ALGORITHM

The LMT algorithm is a two phase scheduling algorithm. The first phase groups the tasks that can be executed in parallel using the level attribute. The second phase is a greedy method that assigns each task to the fastest available processor. A task at a lower level has higher priority for scheduling than a task at a higher level. Within the same level, the task with the highest average computation cost has the highest priority. If the number of tasks in a level is greater than the number of available processors, the fine-grain task, arc merged into a coarse-grain task until the number of tasks is equal to the number of processors. Then the tasks are sorted in reverse order (largest task first) based on average computation time. Beginning from the largest task, each task will be assigned to the processor that minimizes the sum of computation cost of the task and the communication costs with tasks in the previous layers and does not have any scheduled task at the same level.

2.3 HETEROGENEOUS EARLIEST-FINISH-TIME ALGORITHM

HEFT is a simple and best scheduling technique in static task scheduling in heterogeneous as well as homogeneous environment for limited number of processors. HEFT has two stages:

* Prioritization phase
* Processor selection stage.

HEFT calculates the priority using upward ranking (ranku). An application is traversed in upward direction and the rank of all nodes in a list with the help of mean communication and mean computation cost are found. Generated list is arranged in decreasing order of ranku. HEFT uses a Tie breaking policy for selecting the nodes, which node or successor selects whose rank value is highest. Upward rank of task ni is described as:

Rank(ni) = Wi+maxnjϵsucc(ni) (cij+ranku(nj))

where Wi is the mean computation cost, succ (ni) is the immediate child of node ni, ci,j is the mean communication cost of node (i,j). In case of two nodes have equal rank value selects randomly. In upward ranking, graph is traversed from exit node to entry node. Highest rank is same with exit node:

ranku(nexit) = W exit ranku(ni)

is total critical path from source node to exit node including communication and computation cost of tasks.

2.4 CRITICAL-PATH-ON-A-MACHINE

CPOP computes the rank value of each node by adding both the techniques ranku+ rankd. An application is traversed from entry node (ni) to exit node (nj) is called downward ranking and traverse exit node to entry node is called upward ranking. CPOP has two steps:

* Task prioritization phase
* Task allocation phase.

Task Prioritization Phase:

Tasks are prioritized according to their rank value (ranku+ rankd) with the help of communication and computation costs of DAG then set into a queue (decreasing order). CPOP used critical path (CP) of an application to find the longest path starting from entry node to exit node.

Task allocation Phase:

Tasks are selected according to higher rank value and selects for scheduling to best suitable processor which minimizes the execution time of task. CP nodes are scheduled on a processor which has less mean computation cost then other processors.

2.5 RELIABILITY AWARE SCHEDULING ALGORITHM

Reliability Aware Scheduling Algorithm effectively measures system reliability, based on an optimal reliability communication path search algorithm. Reliability priority rank (RRank) is used to estimate the task's priority by considering reliability overheads. Based on directed acyclic graph (DAG) a reliability-aware scheduling algorithm for precedence constrained tasks, which can achieve high quality of reliability for applications is proposed.

**SYSTEM STUDY**



**CHAPTER 3**

**SYSTEM STUDY**

**3.1 OVERVIEW**

This section discusses the overall flow of the proposed system. Proposed system describes a way to schedule tasks to processor with minimum make span. This is done by Heterogeneous Allotment Aware Scheduling which schedules tasks to processors based on RRank.

**3.2 PROPOSED SYSTEM**

The input to the system is a set of files which contains communication cost between adjacent tasks and processing time of tasks on processors. The proposed algorithm is a two-phase algorithm. The ﬁrst phase ﬁnds a favourable number of replication of tasks. The second phase applies a scheduling method that is based on the expected executed time and the communication time.

**3.2.1 Algorithm Phase I – Allotment Lower Bound**

The recovery of failures may delay completion. Accordingly, this work uses a replication method to reduce the delay. This replication method executes many the same task on many processors. Two strategies, Strategy I and Strategy II, are considered. In Strategy I, no replication of a task can be stopped. In Strategy II, once a replication of a task is completed, the other replications of the task are immediately terminated. Restated, only one replication of a task is completed. The execution of a task is assume to have to be restarted on a failed processor after the processor has been recovered.

The tasks can be mapped as malleable tasks. Accordingly, this work modiﬁes a linear program in [5], [19] to solve the allotment problem. In Strategy II, if *f1(vi , j + 1) <= < f1(vi , j)*for someαi>= 1, then the expected processing time Tp(vi, ) and the expected work *W(vi, )*  strictly decreasing in 1 <<αi+ 1.

An optimal solution for allotment of task *vi* is between *ai* and *m*. Let *AL(i)* be the lower bound on the number of allotted processors for task *vi*. The algorithm of allotment lower bound (Algorithm 1) determines the value of *AL(i)*.

**3.2.2 Algorithm Phase II – Allotment Aware Scheduling**

**3.3. SUMMARY**

This chapter describes the entire workflow of our system. Inputs, outputs of each and every phase are described in detail. The overall functions of the system are studied.

**SYSTEM DESIGN**



**CHAPTER 4**

**SYSTEM DESIGN**

**4.1 OVERVIEW**

This section discuss about the design of our system. The modules of the system are described in detail. The various steps involved in each module are specified.

**4.2 MODULES**

There are 5 modules in our system. The module describes the overall workflow of our system. There are two phases involved in the algorithm,

* Initialization phase
* Main phase

These two phases are accomplishes by 5 modules to design the overall system.

Major modules include,

* Pre-processing module
* Initial Cluster module
* Minor 1 module
* Minor 2 module
* Classification module

The initialization phase includes pre-processing, initial cluster modules. The main phase comprises of minor 1 and minor 2 modules.

**4.2.1 Pre-processing module**

The input to the pre-processing module is a collection of text documents and their auxiliary attributes. Each text document has its own side information and its content. The input text document must be first pre-processed before clustering process starts.

Text Documents and Auxiliary attributes

Pre processed Documents

Stemming

Frequency of the document

Projection

**Fig 4.1:** Pre-processing Module

The steps involved in Fig 4.1 are,

* The words in the document are stemmed to its root word.
* Lovins stemmer algorithm is used to stem the words in the document.
* The be verbs, punctuations are removed from the document in the stemming stage.
* The frequency of each word in the documents is computed.
* In the projection stage, the highest frequency word in each document in found.
* For each highest frequency value, 1+log dt is computed.
* This value is the output of the preprocessing stage.

**4.2.2 Initial cluster module**

The input to this module is the 1+log dt value of all the documents. Buckshot algorithm is used to form the initial cluster. Distance is calculated between the values. Clustering is done based on the minimum distance between the values. The number of clusters formed is based on the number of input text documents. The number of clusters is the root of the text document.

Projection value of each document

Initial Clusters

Find root of no. of document (k)

Update cluster centroid

Calculate the distance between all the documents

Assign remaining documents to cluster

Group the documents with minimum distance

Calculate cluster centroid

YES

NO

Less than k

**Fig 4.2:** initial cluster module

The procedure followed in fig 4.2 are,

* The input is projection value of all the documents.
* Calculate the value of number of clusters.
* Let the number of clusters be k.
* The value of k is root of the number of text documents.
* Calculate the distance between each and every text documents.
* Group the documents with the minimum distance.
* Repeat the steps till it reaches k.
* Calculate the cluster centroid.
* Assign the remaining documents to cluster with the closest centroid value.
* Update the cluster centroid.
* Initial number of clusters is the output of this module.

**4.2.3 Minor 1 module**

The input to this module is an initial set of clusters. Calculate the similarity between the documents. Similarity is found by calculating the cosine similarity value. Reconstruct the cluster based on the cosine similarity value.

Initial Clusters

Calculate the cosine similarity

Find the cluster with highest cosine similarity value

Assign the document to that cluster

Update the cluster

**Fig 4.3:** Minor 1 module

The steps in fig 4.3 are,

* The input is the initial cluster obtained from the initialization phase.
* Calculate the cosine similarity between the documents and all the clusters.
* Find the cluster with the highest cosine similarity value.
* Assign the document to that cluster which has the highest cosine similarity value.
* Update the cluster centroid.
* The output of this module is the reconstructed cluster.

**4.2.4 Minor 2 module**

The cluster formed in the previous module is the input to this module. Side information is used here. The major aim of this module is to find the informative auxiliary attributes and to generate clusters with the help of these auxiliary attribute. To find the informative auxiliary attribute the concept of gini index is used.

Cluster from minor 1

Calculate Gini index for an auxiliary attribute

Update the cluster

Assign the document to that cluster

Calculate threshold value

If gini index>threshold

YES

Calculate posterior probability for the auxiliary attribute

Find the cluster with the highest probability value

NO

Auxiliary attribute is informative

**Fig.4.4:** Minor 2 module

The following steps are followed in fig 4.4,

* The input is the set of clusters that is formed in the minor 1 module.
* Calculate the Gini index for each auxiliary attribute.
* Calculate a threshold value.
* Check whether the gini index is greater than the threshold value.
* If yes, the auxiliary attribute is informative and can be used for clustering else the auxiliary attribute is omitted.
* Calculate the posterior probability for the informative auxiliary attributes.
* Find the cluster with the highest posterior probability value.
* Assign the document to the cluster with highest posterior probability value.
* Update the cluster centroids.

**4.2.5 Classification module**

Classification is the process of assigning labels to the documents. The clusters formed as the result of the clustering process is the input to classification. The most appropriate label is selected for each document and it is assigned to it.

Clusters

Find r closest clusters by calculating cosine similarity

Find r closest clusters by calculating posterior probability

Find the labels of the documents present in 2r clusters

Find the majority label

Assign the label to the document

**Fig.4.5:** Classification module

The steps involved in fig 4.5 are,

* The input is the final clusters from the clustering process.
* The top closest r clusters are selected by calculating the cosine similarity value.
* The top closest r clusters are selected by calculating the posterior probability value.
* Collect all the labels of the documents that is present in the cluster.
* Find the majority label available.
* Assign that label to the text document.

**4.3 SUMMARY**

This section discussed the entire design of our system. The modules used and their functionalities are described in detail.

**SYSTEM IMPLEMENTATION**



**CHAPTER 5**

**SYSTEM IMPLEMENTATION**

**5.1 OVERVIEW**

This chapter discussed about the implementation of the system. The system is implemented with the use of Hadoop 2.6.0 integrated with eclipse. The two main algorithm used in this system are the COATES (COntent and Auxiliary attribute based TExt cluStering algorithm) for clustering the text documents and the COLT (COntent and auxiliary attribute-based Text classification algorithm) for giving labels to the documents.

**5.2 CLUSTERING**

The input for our clustering algorithm is number of text documents denoted by Ti...TN and the auxiliary attributes Xifrom which informative auxiliary attributes are extracted and clustering is done based on that attributes. We describe the COATES algorithm for clustering as follows.

**5.2.1 The COATES Algorithm:**

This algorithm is used for clustering text documents based on side information. It is abbreviated as COntent and Auxiliary attribute based TExtcluStering algorithm. It is necessary to perform Buckshot algorithm to get k clusters which becomes input to the COATES algorithm.

This is explained in Initialization phase and the COATES algorithm is explained in Main phase.

**Initialization phase:**

This phase forms the initial clusters for COATES algorithm.

**Step 1: Stemming**

In this process all the words in the document are stemmed to its root word and also all the 'be' form verbs are removed. For this step, we have used Lovins Stemmer algorithm devised by Julie Beth Lovins, which is more suitable for our project than all other algorithms.

**Step 2: Pre-processing**

In this process, frequency of occurrence of each stemmed word in a document is identified. Each document can have many numbers of words and the similar words are counted and stored in a linked list.

**Step 3: Projection**

Projection finds the highest frequency in each document denote it as dt for every document. Then, calculate the value of 1+log dt.Each document has one projected value.

**Step 4: Clustering**

Based on the projected value for each document, the documents are clustered. Since there are N documents, there must be √N clusters based on Buckshot algorithm. Form NxN matrix and find the distance between a projected value with all other document's projected value. A matrix will be formed in which the smallest **√**N values are taken and group those two documents to form a cluster. At the end of this process, √N clusters will be available consisting of two documents each. Then, calculate the centroids for each cluster. The remaining documents are put into the cluster in such a way that, calculate the distance between the projected value of each document with centroids of all clusters. To which cluster, the distance is small, place the document in that cluster. Again calculate the cluster centroid after adding document to it. Repeat this step until all documents are placed in cluster. Update cluster centroids.

With this formation of initial clusters, the Initialization phase gets completed.

**Main Phase**

The clustering based on text content of the document and the side information is done in this main phase.

**Step 5: First Minor Iteration**

In this step, the clusters are again refined based on the content of the text document. Consider, there are say, five clusters which contains three documents each. Take the first document Ti and take the first cluster Cj. Find the similar words in a cluster (i.e.,) the words present in all the documents of a particular cluster and count their number of occurrences and place it in an array. Check whether the same word is present in the document Ti and count the occurrences and store it in the array. Calculate the Cosine Similarity between the document and the cluster using the formula

Cosine Similarity = (Eqn. 5.1)

Where,

* k is the index of the common words taken for calculating Cosine Similarity.
* Ti specifies the document
* cs is the cluster index
* t is the total number of common words taken for consideration.

Repeat this step with all the clusters. Place the document Ti in the cluster for which the document Ti has the highest similarity. The cluster is redefined. Repeat the same step for all documents and place the documents in the modified cluster.

**Step 6: Second Minor Iteration**

The clusters are again refined based on the side attributes. The auxiliary attributes associated with the document Ti is mentioned as Xi. Xi is the set which contains only the auxiliary attributes associated with the document Ti. For every Xi in Ti, find the relative presence of the auxiliary attribute in the cluster.

Prj = (Eqn. 5. 2)

Where,

* Prj is the relative presence of auxiliary attribute r in cluster j
* frj is the number of documents in cluster j containing the auxiliary attribute r

Relative presence =

Gini index is calculated for each auxiliary attribute using the formula

(Eqn.5.3)

where,

* Gr is the gini index of the auxiliary attribute r.

Gini index is just squaring and adding the relative presence of the auxiliary attribute r in all clusters. The value of gini index must lies between 1/k and 1. Clustering is based only on the auxiliary attributes whose gini index is greater than a particular threshold ɣ. The threshold ɣ is calculated as 1.5 standard deviations below the mean value of the gini index in that particular iteration. Identify the auxiliary attributes whose gini index is greater than the particular threshold ɣ for the document and put it in a set Ri. The same process is repeated for all the documents and Ri...RN exists for Ti...TN.

In order to calculate Posterior probability, we need to find Pa(Ri) and Pa(Ri|tiCj). The first probability is evaluated as,

Pa(Ri) = (Eqn.5.4)

which is,

* Ri is the set of auxiliary attributes whose gini index are greater than the threshold value for the document Ti
* r is the attribute in the set Ri

The second probability is evaluated as,

= (Eqn. 5.5)

Which is

The third probability Pa (TiCj) is evaluated as,

Pa (TiCj)=

The posterior probability for every document with each cluster is calculated as,

= (Eqn.5.6)

Then, the normalized posterior probabilities of each document with every cluster is calculated as,

(Eqn.5.7)

where,

* = Posterior probability of document Ti with cluster Cj
* = Sum of posterior probability of document Ti with all clusters.

Place the document Ti in the cluster Cj whose posterior probability is highest among all. Repeat the same process for all the documents Ti...TN. Hence, the cluster is refined.

Repeat the steps – First Minor iteration and Second Minor iteration until the changes between the currently obtained output and the output obtained during the previous iteration is less than 1%. Refined clusters are obtained after the end of this process.

**5.3 CLASSIFICATION**

The clusters formed after this clustering algorithm is given as input to the classification algorithm. In this algorithm, all the documents are assigned with labels for easy identification based on the COLT algorithm.

**5.3.1 COLT Algorithm:**

The abbreviation of this algorithm is COntent and auxiLiary attribute-based Text classification algorithm. This algorithm uses supervised clustering approach which clusters the given documents into k clusters and assigns names to all the documents. The steps followed are,

**Step 1: Feature Selection**

In this process, gini index is calculated for every auxiliary attribute with the given labels and select the attributes whose gini index is greater than the threshold value. The threshold value is calculated as standard deviations of the gini index subtracted from the average value of the gini index. This process is called feature selection and the attributes selected are used for further classification.

**Step 2: Initialization**

In Initialization phase, the process till initial clustering is followed which will be then given to COATES algorithm. The processes of Stemming, Pre-processing, Projection and Initial clustering is carried out to find out number of clusters will obtain for the given number of input documents.

**Step 3: Cluster Training Model Construction**

In this phase, the main phase of COATES algorithm is followed with a change in that. It is instead of using all the auxiliary attributes in the second minor iteration, the attributes which are selected from the step 1 are only chosen. At the end of this step, supervised clusters are formed each having some documents.

**Step 4: Classification**

In this phase, all the documents are labelled with a name. It can be done as follows. Find the cosine similarity of each document with all the clusters and take the clusters whose cosine similarity is nearer to the threshold value. The threshold value is identified as the cosine similarity value of the particular document with the cluster in which it is placed. Consider the number of clusters taken as r. Find the posterior probability of each document with all the clusters and take r closest clusters. Now, we have 2r (r (Cosine similarity) + r (Posterior probability)) number of clusters. Take the documents present in these clusters. Calculate the number of occurrences of each label in the taken documents. Assign the label which has maximum occurrence as the label for the document. Repeat the step for all the documents.

**5.4 SUMMARY**

Thus, the two main algorithms are explained in this chapter. All the formulae used, the cosine similarity, calculating gini index and posterior probability are explained clearly in this chapter.

**RESULTS AND DISCUSSIONS**



**CHAPTER 6**

**RESULTS AND DISCUSSIONS**

**6.1. OVERVIEW:**

In this paper, we designed an algorithm to perform text mining with use of side information. Text documents contain large amount of information that are used in order to perform clustering process. In order to design clustering process, we performed iterative partitioning technique with probability estimation process which calculates importance of different forms of side information. This technique is used for both clustering and classification algorithms. The effectiveness and efficiency of our approach is illustrated by results based on real data sets. This results show that the quality of clustering and classification can be greatly enhanced with the use of side information while maintaining high level of efficiency.

**6.2. OUTPUT**

The output to our algorithm is the list of clusters and the document present in each cluster. We have given an id to each and every document which is given as the input and the cluster formed. Our output is in format,

Cluster number Document number

The document number is unique. Each document occurs in only one cluster. A cluster can have any number of documents. The number of clusters depends on the number of input documents given as inputs. The clusters generated are fair enough.

This output from the clustering is given as an input to the classification algorithm. The final clusters are given as input to the classification process. The output of classification process is the class labels for each text document. Each document is assigned with an appropriate class label. The output is of the format,

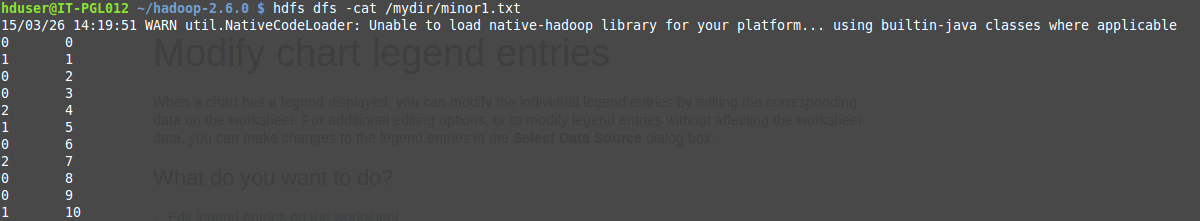
Document number Class label

The labels indicate the domain associated with the given document. The document can be classified based on their labels assigned to them.

**CLUSTERING OUTPUT**

Cluster number

Document number



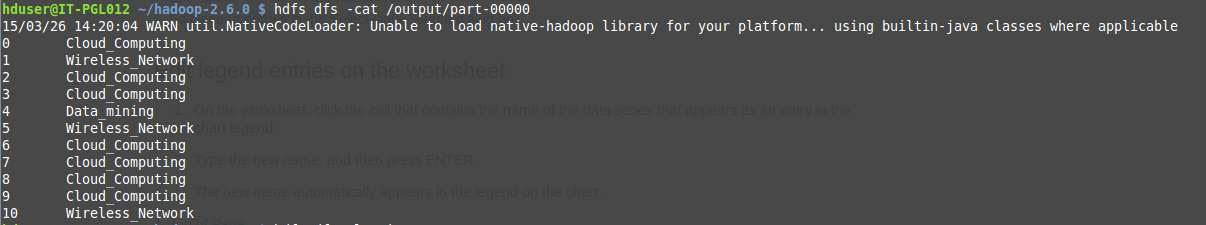
**Fig. 6.1:** clustering output

Fig 6.1 shows the output of clustering. The first column specifies the cluster number and the second column specifies the document number.

**CLASSIFICATION OUTPUT**

Document number

Class Label



**Fig. 6.2:** Classification output

Fig.6.2 shows the output of classification algorithm. The first column indicates the document number and the second column shows the appropriate class label for each text document.

**6.3 PERFORMANCE ANALYSIS**

In this section, we compare our clustering methods against a baseline technique on real data sets. We refer to our clustering approach as COATES (Content and Auxiliary attribute based TExtcluStering). As the baseline, we adopt K-Means approach to text. This approach is widely known to form clusters in an effective way. Thus we can compare our methods with baseline which are chosen in such a way that we can evaluate advantage of our approach over pure text mining.

The aim of evaluation metrics is to show that our approach is superior to natural clustering techniques. For each cluster, we compute cluster purity, which is defined as fraction of the documents in cluster which corresponds to its dominant class label. Dominant class label is nothing but a label which occurs maximum number of times in cluster.

Let number of documents in k clusters be denoted by n1…nk.

Dominant cluster label can be represented by l1…lk.

Let number of documents with cluster label li be denoted by ci.

Then overall cluster purity P is defined by

P = (Eqn. 6.1)

The cluster purity always lies between 0 and 1. Perfect clustering will provide cluster purity of almost 1, whereas poor clustering will provide very low values of cluster purity. For efficiency, we tested the execution time of our method with respect to baseline technique.

The effectiveness results for baseline technique and COATES algorithm with increasing number of clusters for CORA dataset is illustrated as follows.

Consider an example,

|  |  |  |
| --- | --- | --- |
| Document number | Cluster number | Class label |
| 0 | 0 | Cloud\_computing |
| 1 | 1 | Wireless\_network |
| 2 | 0 | Cloud\_computing |
| 3 | 0 | Cloud\_computing |
| 4 | 2 | Data\_mining |
| 5 | 1 | Wireless\_network |
| 6 | 0 | Cloud\_computing |
| 7 | 2 | Cloud\_computing |
| 8 | 0 | Cloud\_computing |
| 9 | 0 | Cloud\_computing |
| 10 | 1 | Wireless\_network |

Here total number of text documents is 11. Total Clusters obtained after clustering process is 3.Documents are assigned with class labels such as data mining, cloud computing, wireless networks.

Purity Calculation

n0 = 6, l0 = Cloud\_computing, c0 = 6

n1 = 3, l1 = Wireless\_network, c1 = 3

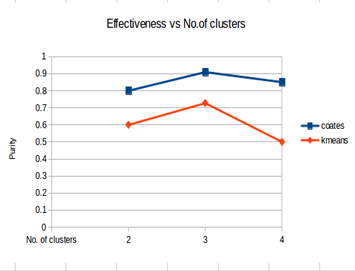
n2 = 2, l2 = Data\_mining, c2 = 1

P = c0+c1+c2/n0+n1+n2

=10/11

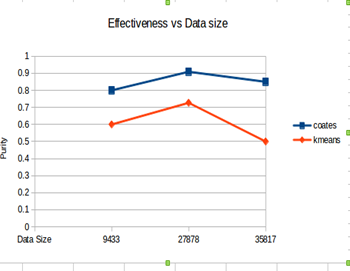
=0.9

The number of clusters is marked on X-axis whereas cluster purity is marked on Y-axis. The purity of clusters will slightly increase when number of clusters increases on data sets. This is because large number of clusters results in finer granularity of partitioning.



**Fig.6.3:** Effectiveness Vs No. of clusters

We also tested effectiveness of method with increasing data size. X-axis illustrates size of data and Y-axis illustrates purity of clusters. We find improvement in quality for large data sets. This is because for small data sets, sparse side information is less informative. In such cases, most auxiliary attributes appear very infrequently and so it is difficult to form clusters. As the size of data sets increases, it will enhance the quality of clusters.



**Fig.6.4:** Effectiveness Vs Data size

**6.4 SUMMARY**

This chapter discussed about the outputs obtained as the result of this algorithm. The output obtained from our algorithm is compared with the existing system. We have analysed our system and the existing system and found that our cluster quality is better than the existing systems. We computed the cluster quality with respect to the number of clusters and input data size.

**CONCLUSION**



**CHAPTER 7**

**CONCLUSION**

In this paper, an algorithm to perform text mining with use of side information is designed. For this, clustering helps in grouping documents based on text content and side attributes. In our proposed method, initially clusters are formed based on frequency of words in each text documents. Then it can be refined by finding cosine similarity between the documents. Finally, the resulting clusters are formed with the use of high informative side attributes. This approach performs iterative partitioning technique with probability estimation process which computes importance of different forms of side information. This approach is used in both clustering and classification algorithms. In classification, each cluster is assigned with meaningful class labels. The effectiveness of our approach is illustrated by results based on real data sets. This results show that the quality of clustering and classification can be greatly enhanced with the use of side information while maintaining high level of efficiency.

**FURTHER ENHANCEMENT**

Proposed project provides the output as clusters which can be formed by mining more informative text data from the documents and clusters are formed by grouping documents based on frequency, cosine similarity, and probability estimation process for computing the usefulness of side information.

In the project, author can be taken as side attributes. Based on that, documents can be clustered. But it is not mandatory to use only one side information for clustering process. For further enhancement, any numbers of side information can be used. This will improve quality of clusters and produce more effective results.

**APPENDIX**

**CHAPTER 8**



**APPENDIX**

**8.1. HARDWARE REQUIREMENTS**

* Operating System - Linux Mint
* Hard Disk - 121 GB
* RAM - 2 GB

**8.2. SOFTWARE REQUIREMENTS**

* Hadoop-2.6.0
* Eclipse IDE 3.8

**8.2.1. Architecture of Hadoop:**

The Hadoop is a framework that allows processing large amount of data in a distributed environment. It is designed to improve scalability. It offers large computation and storage facilities. It consists a rich set of libraries for detecting exceptions and failures in the application layer. Hadoop can be single node or multimode cluster. We have used single node cluster in our project. A Hadoop project includes the following modules,

* **Hadoop Common**: The common utilities that support the other Hadoop modules
* **Hadoop Distributed File System (HDFS)**: A distributed file system that provides high-throughput access to application data.
* **Hadoop YARN**: A framework for job scheduling and cluster resource management.
* **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets.

Hadoop uses master/slave architecture for distributed storage and computation. The entire task in Hadoop is executed in a form of map and reduce program. Execution of a task in Hadoop is known as job. Before the Hadoop framework executes the task, the user has to specify the following details,

* The location of the input and output files in the distributed file system
* The input and output formats
* The classes containing the map and reduce functions

|  |
| --- |
| partsofMR.png |
| Parts of a MapReduce job |

**Fig.8.1:** Different Task in Map Reduce program

The various task involved to execute a map reduce program is as follows,

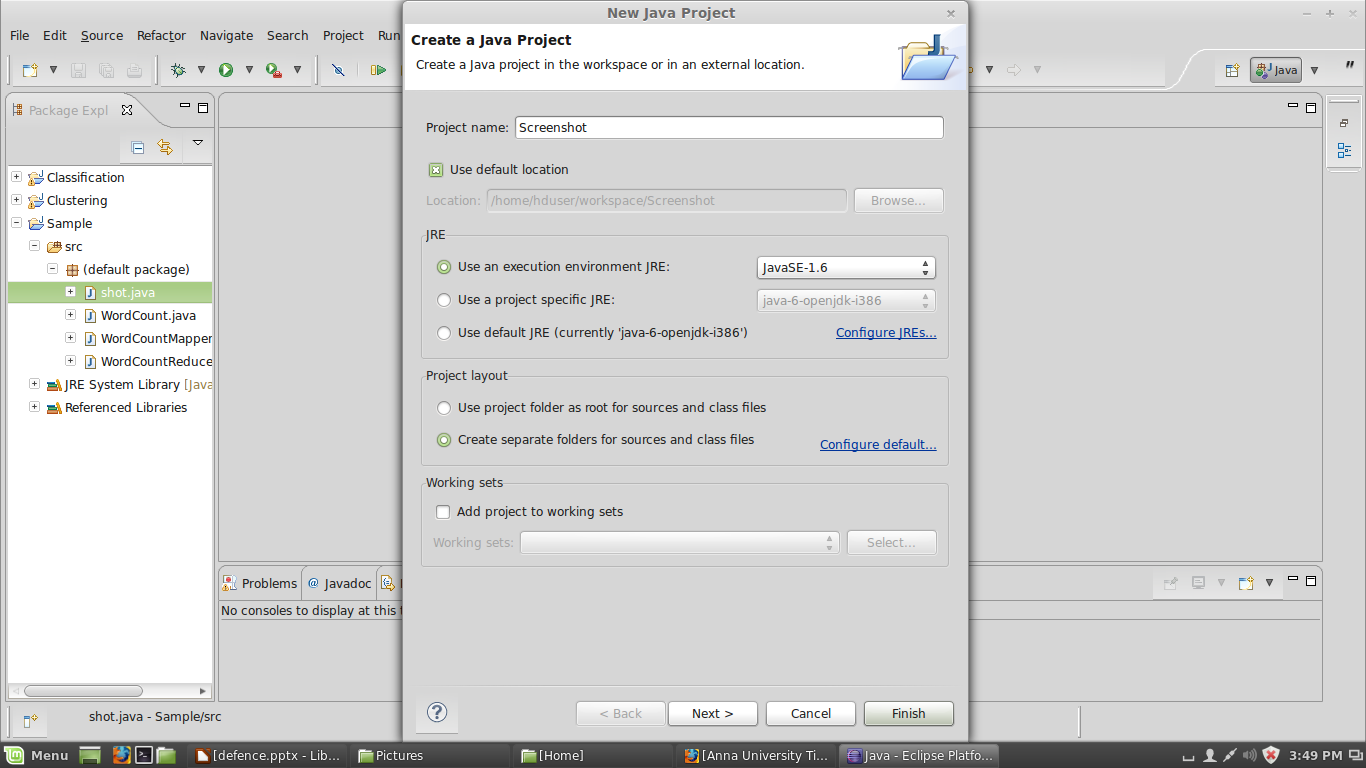
* The user specifies the job configuration by setting different parameters specific to the job.
* The user also specifies the number of reducer tasks and the reduce function.
* The user also has to specify the format of the input, and the locations of the input.
* The Hadoop framework uses this information to split of the input into several pieces.
* Each input piece is fed into a user-defined map function.
* The map tasks process the input data and emit intermediate data.
* The output of the map phase is sorted and a default or custom partitioning may be applied on the intermediate data.
* The reduce function processes the data in each partition and merges the intermediate values or performs a user-specified function.
* The user is expected to specify the types of the output key and the output value of the map and reduce functions.
* The output of the reduce function is collected to the output files on the disk by the Hadoop framework.

**8.2.2. Integration of Hadoop with Eclipse**

There are several steps involved in integrating Hadoop with Eclipse IDE. Java code is used to write the map reduce program. These map and reduce program are written in the Eclipse IDE. These programs are then executed in the Hadoop Environment. The various steps are,

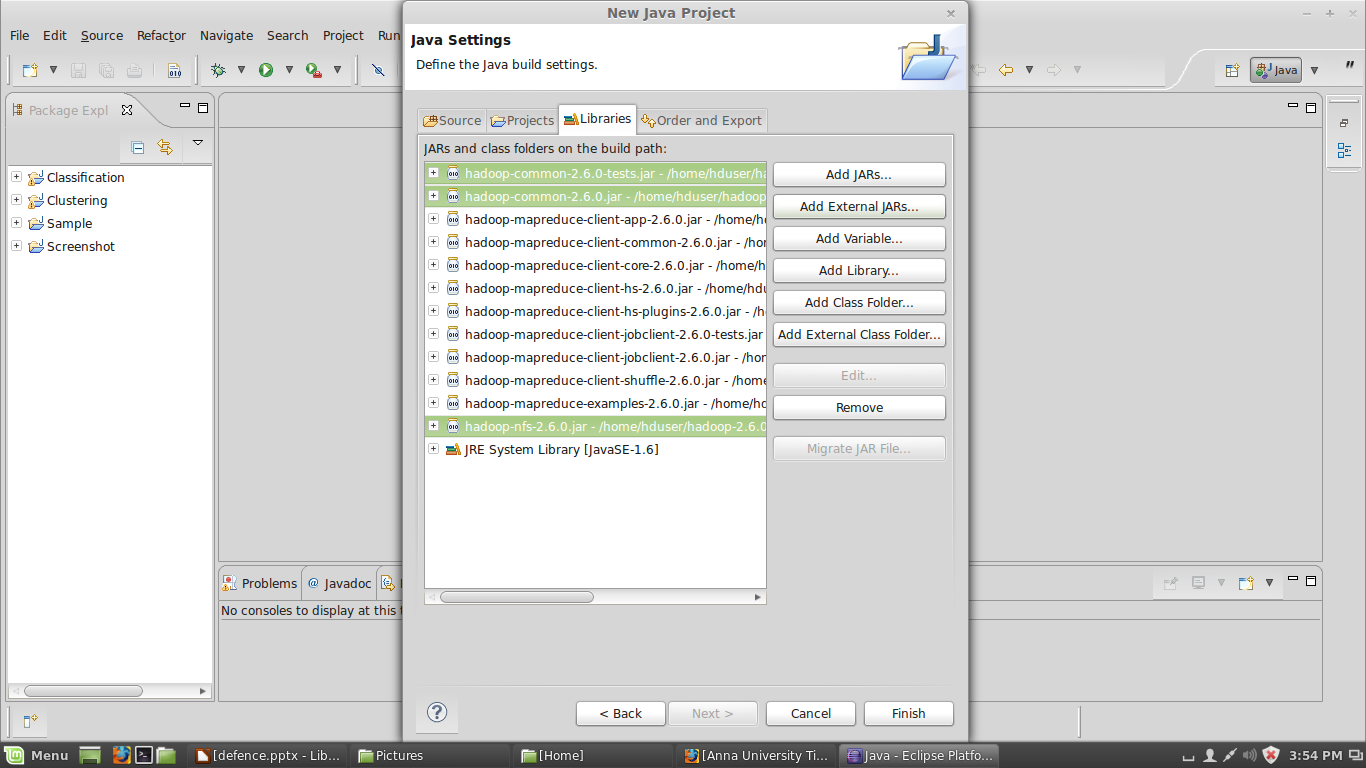
* Create a new java project.
* Add all the external Hadoop jar files.
* Write the Map reduce program
* Create a configuration file to execute the program
* Export the program to a jar file.
* In Hadoop, run all the daemon nodes.
* Create an input directory and store the input in it.
* Run the jar file and the output file is created.

**Step 1 :** Create a new java project. Select **file -> new ->java project**. A wizard is opened. Specify the name of our project. Click next>



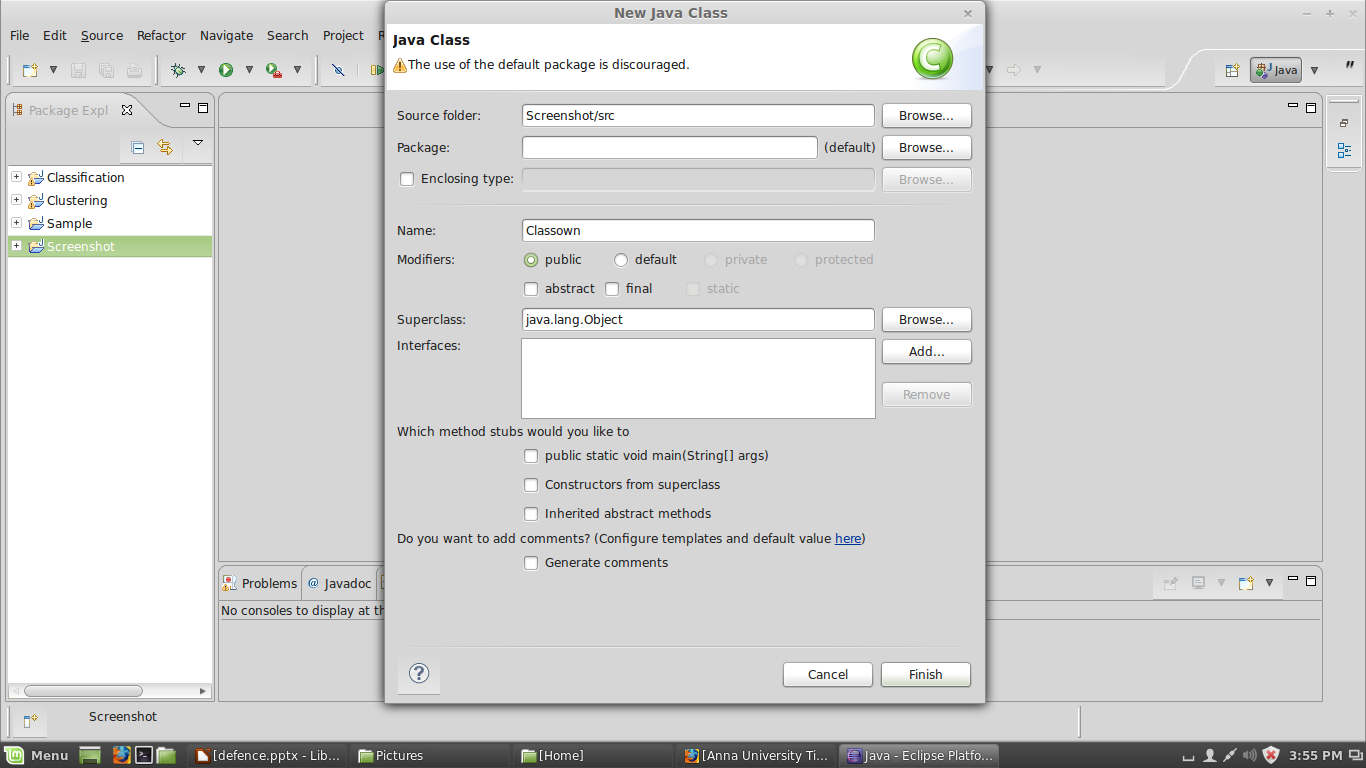
**Fig 8.2:** Creation of new Java project in Eclipse

**Step 2 :**In the next window , select the **Libraries tab.** Click **Add External JARs** and include all the jar files required to write a map reduce program. Once all the jar files are included select Finish.



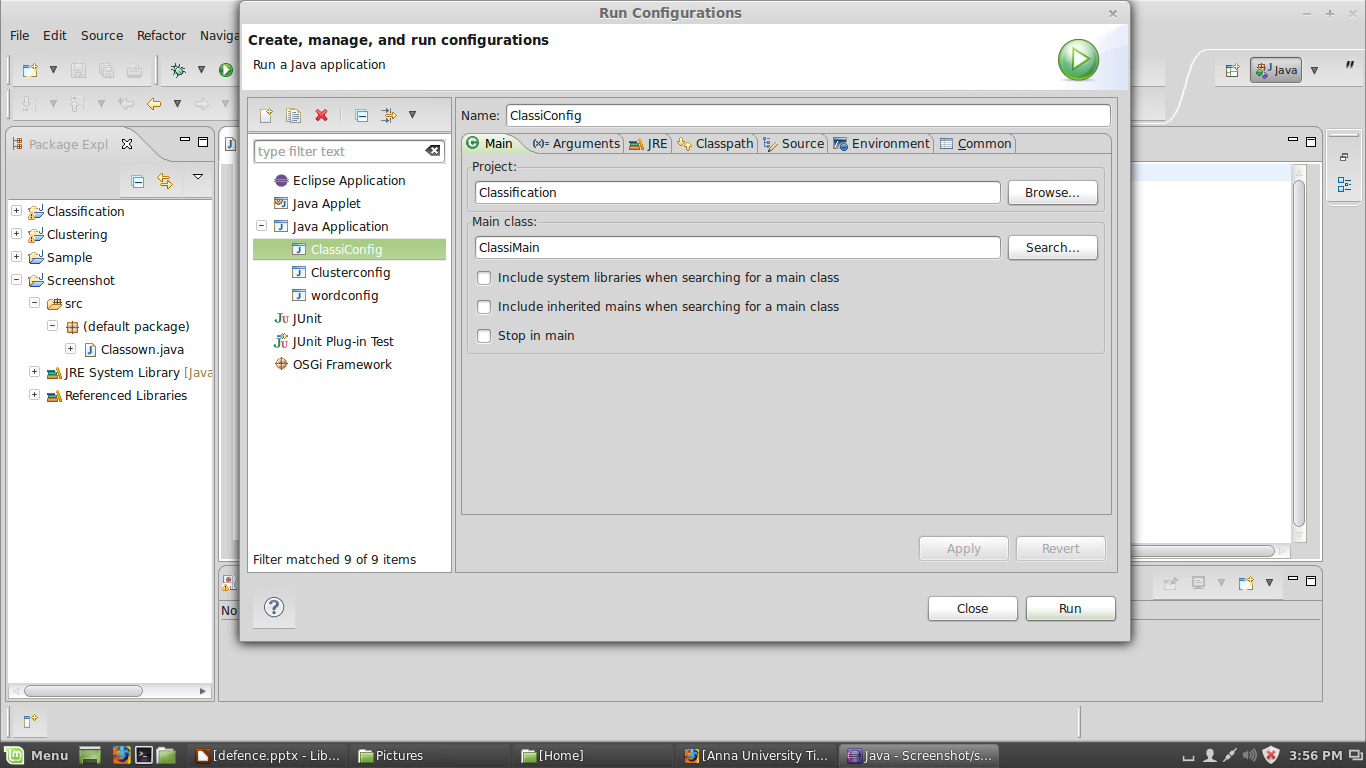
**Fig 8.3:** Addition of external Hadoop Jars to Eclipse.

**Step 3:** Once the project is created. Create a new class for the project. Right click on the project, select **new->java class**. Specify the class name. Click Finish.



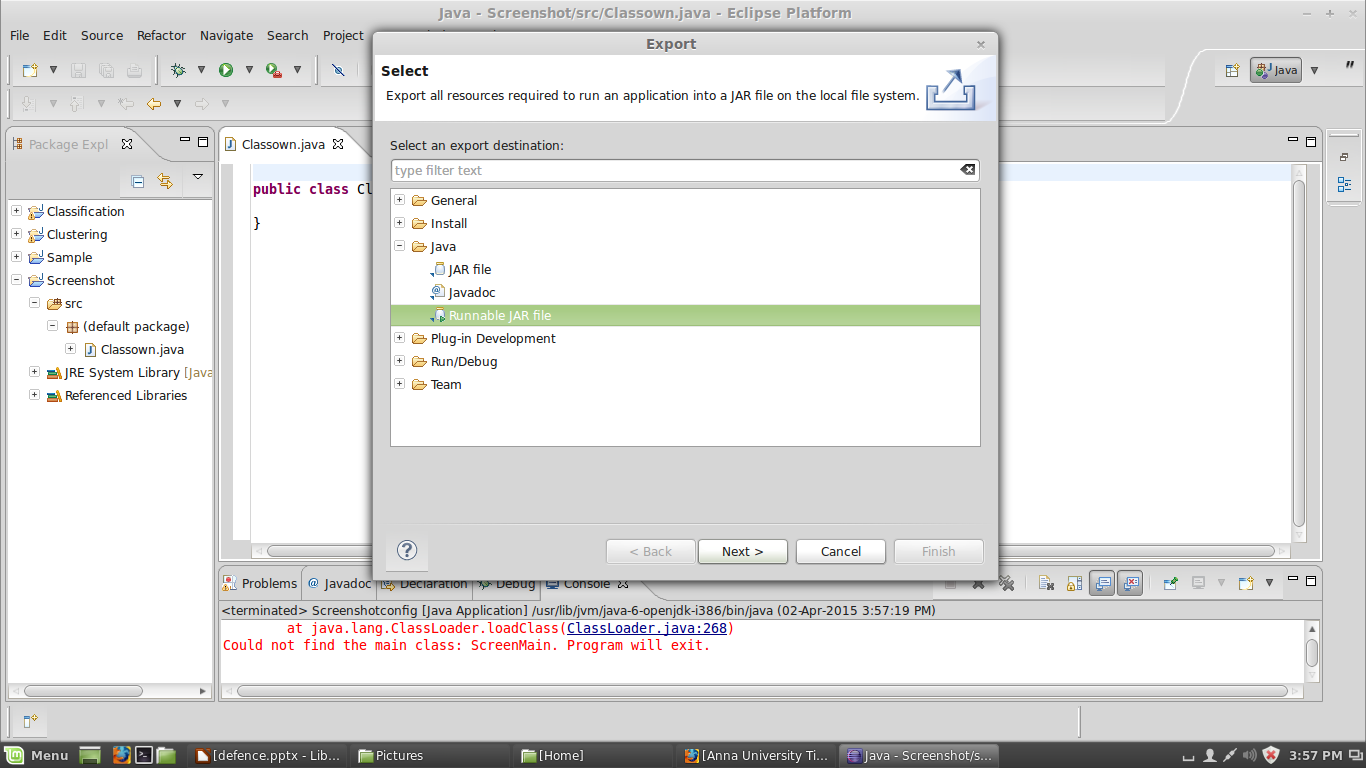
**Fig 8.4:** Creation of java class in Eclipse

**Step 4 :** Once the class is created, include the map and reduce programs. Then click **Tools -> run Configuration**. Create a configuration file by specifying the project and class name. Run the project.



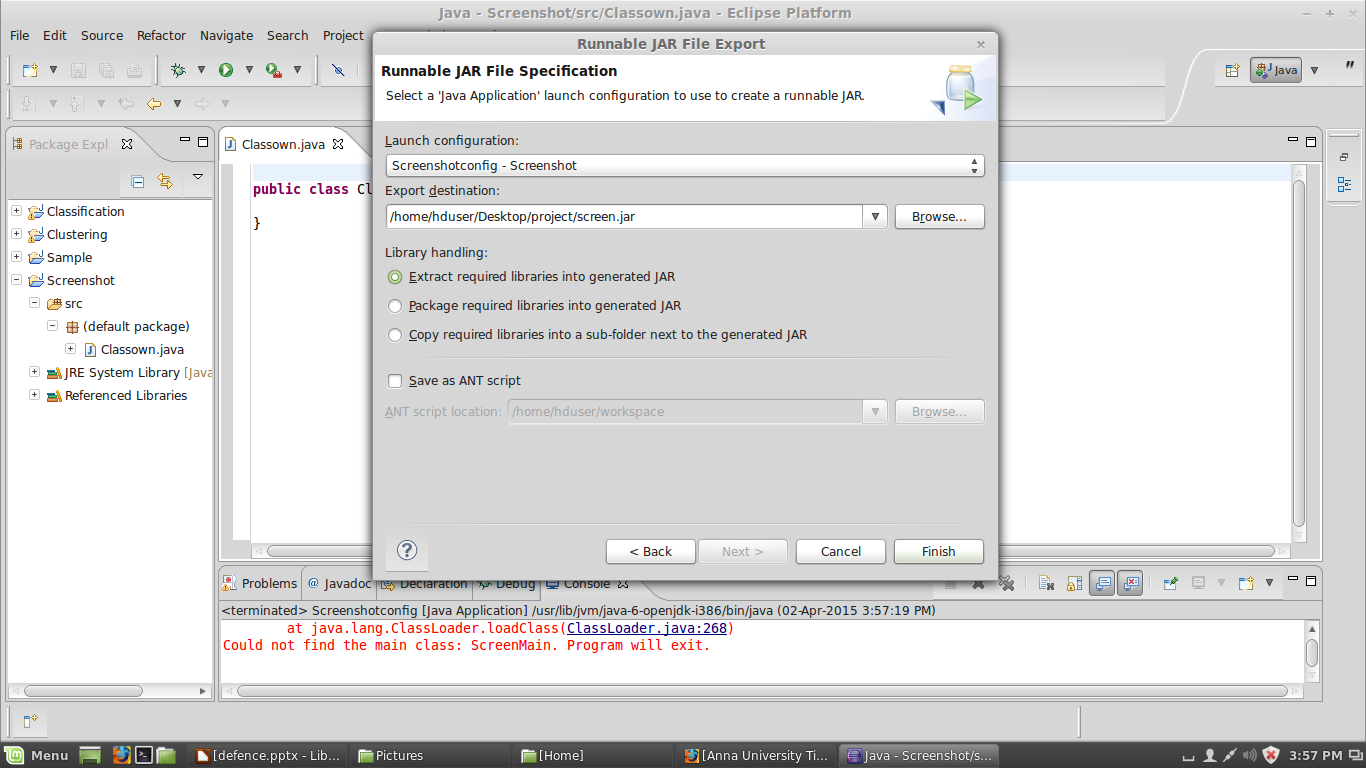
**Fig 8.5:** Creation of configuration files.

**Step 5**: Export the project to a jar file. Click **File -> Export**. A wizards gets open. Select **Runnable jar file** and click Next.

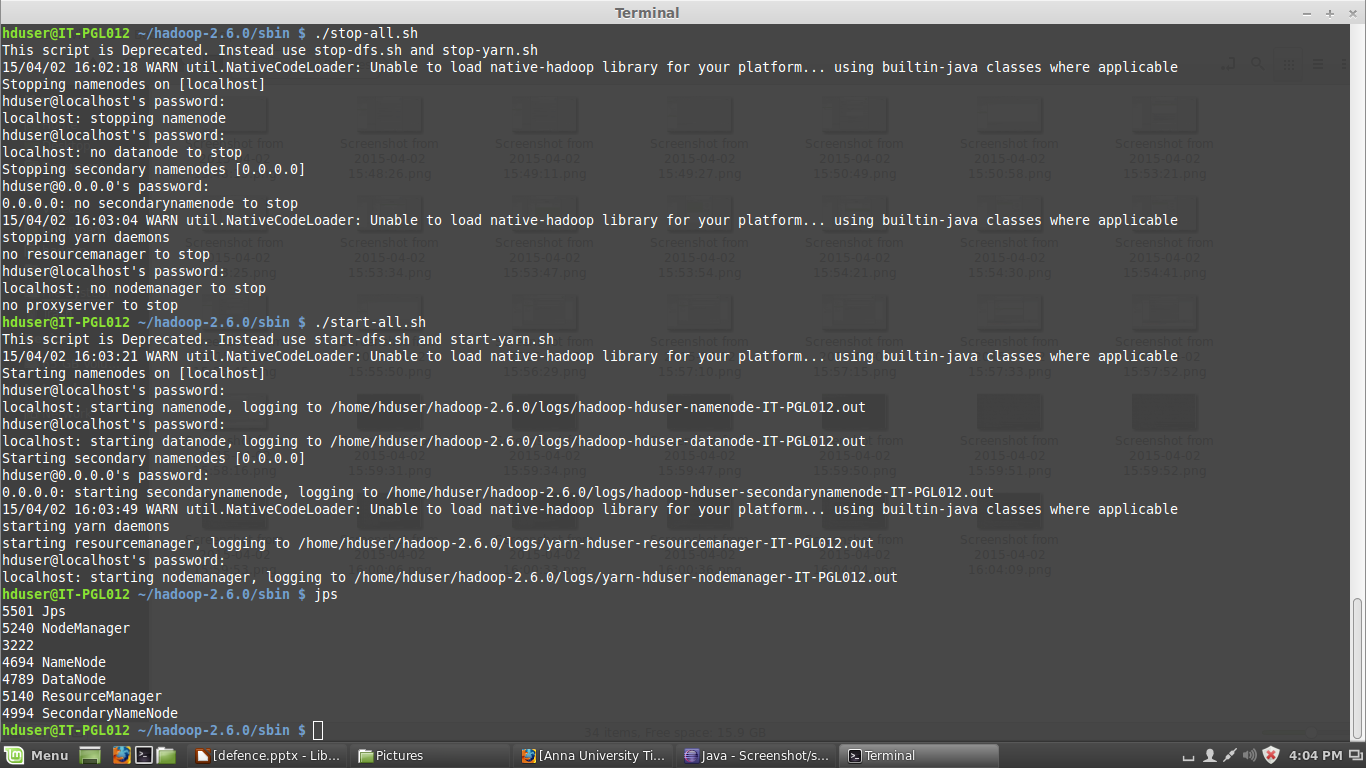


**Fig 8.6:** Export of resources into a jar file

**Step 6 :** In the Runnable Jar File Export wizard , select the configuration file created for our project. Specify a desired location to create the jar file. Click Finish.

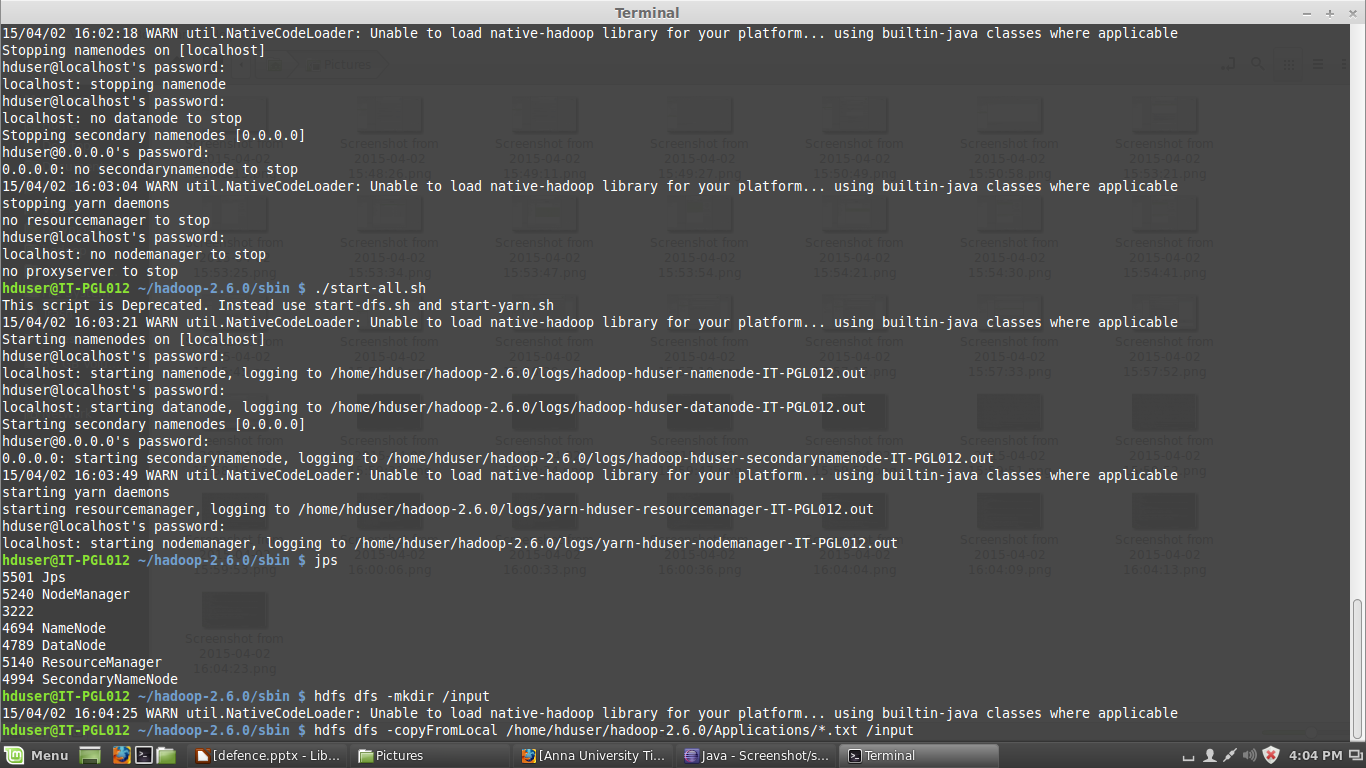
**Fig 8.7:** Creation of Jar file

**Step 7:** In the Hadoop environment, first format the name node. The command used is **hadoop namenode –format**. Then start all the daemon nodes. The command to start all the nodes is **./start-all.sh**.



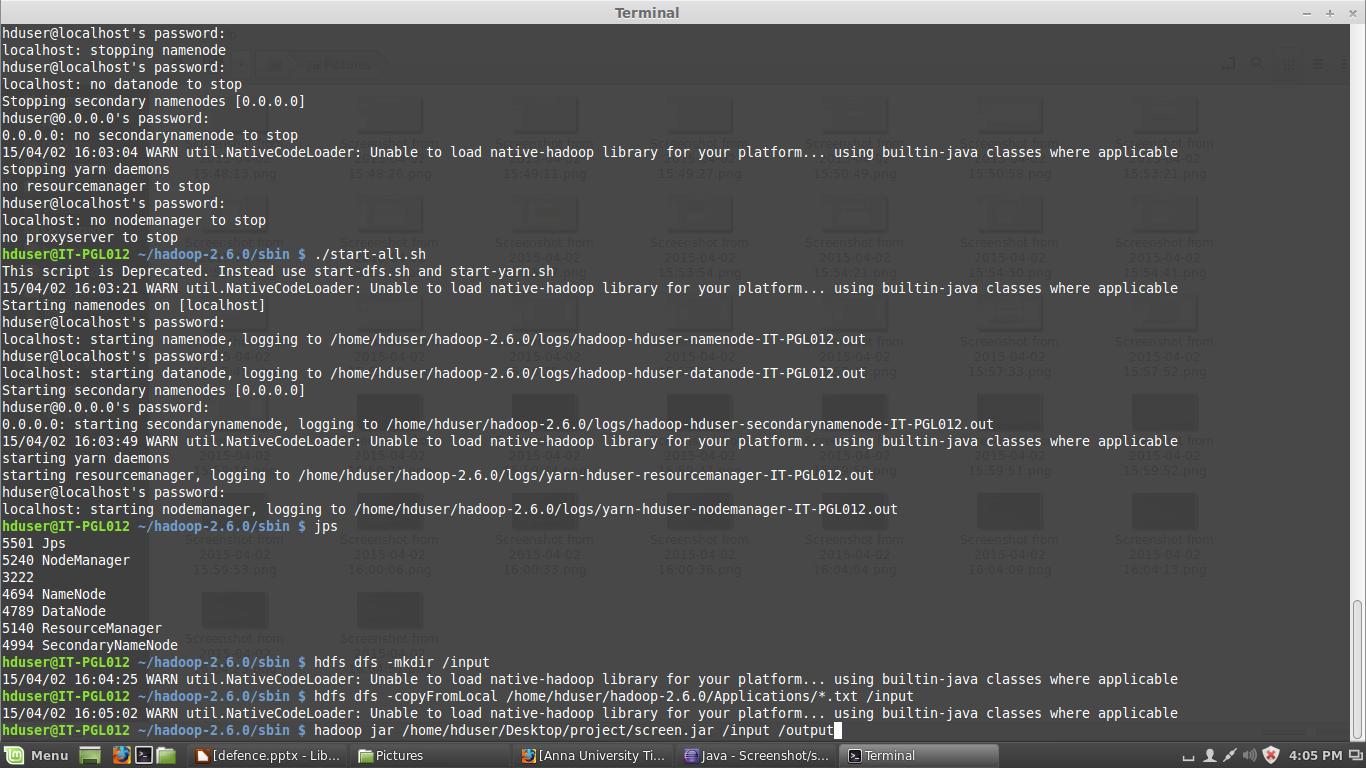
**Fig 8.8:** Running of Daemon nodes.

**Step 8 :** Create a input directory in hdfs. Copy the files from the local file system to the hdfs file system.



**Fig 8.9:** Creation of input directory.

**Step 9:** Run the jar file. Specify the location of the jar file, input directory and the output directory. The output directory must not be created manually. We have to just specify the name , it will be created automatically at the time of successful execution.



**Fig 8.10:** Execution of the program

**8.3. SOURCE CODE**

**8.3.1. Clustering**

**Main**

import java.io.BufferedReader;

import java.io.File;

import java.io.InputStreamReader;

import java.util.HashMap;

import java.util.LinkedList;

import java.util.StringTokenizer;

import javax.naming.spi.DirectoryManager;

import org.apache.hadoop.fs.FileStatus;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.FileUtil;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.conf.\*;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapred.\*;

import org.apache.hadoop.util.\*;

public class ClusterMain extends Configured implements Tool{

static int t=0;

public int run(String[] args) throws Exception

{

JobConf conf = new JobConf(getConf(), ClusterMain.class);

conf.setJobName("Clustering");

final String OUT\_PATH="intermediate\_output";

final String OUT\_PATH1="intermediate\_output1";

final String OUT\_PATH2="intermediate\_output2";

final String OUT\_PATH3="intermediate\_output3";

final String OUT\_PATH4="intermediate\_output4";

final String OUT\_PATH5="intermediate\_output5";

LinkedList l1,l;

l=new LinkedList();

l1=new LinkedList();

int j=0;double percent=1.0;

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(StemmingMapper.class);

conf.setReducerClass(StemmingReducer.class);

Path inp = new Path(args[0]);

Path out = new Path(OUT\_PATH);

FileInputFormat.addInputPath(conf, inp);

FileOutputFormat.setOutputPath(conf, out);

JobClient.runJob(conf);

JobConf conf1 = new JobConf(getConf(), ClusterMain.class);

conf1.setJobName("Clustering");

conf1.setOutputKeyClass(Text.class);

conf1.setOutputValueClass(IntWritable.class);

conf1.setMapperClass(FreqMapper.class);

conf1.setReducerClass(FreqReducer.class);

Path inp1 = new Path(OUT\_PATH);

Path out1 = new Path(OUT\_PATH1);

FileInputFormat.addInputPath(conf1, inp1);

FileOutputFormat.setOutputPath(conf1, out1);

JobClient.runJob(conf1);

JobConf conf2 = new JobConf(getConf(), ClusterMain.class);

conf2.setJobName("Clustering");

conf2.setOutputKeyClass(IntWritable.class);

conf2.setOutputValueClass(DoubleWritable.class);

conf2.setMapperClass(ProjectMapper.class);

conf2.setReducerClass(ProjectReducer.class);

Path inp2 = new Path(OUT\_PATH1);

Path out2 = new Path(OUT\_PATH2);

FileInputFormat.addInputPath(conf2, inp2);

FileOutputFormat.setOutputPath(conf2, out2);

JobClient.runJob(conf2);

JobConf conf3 = new JobConf(getConf(), ClusterMain.class);

conf3.setJobName("Clustering");

conf3.setOutputKeyClass(IntWritable.class);

conf3.setOutputValueClass(IntWritable.class);

conf3.setMapperClass(clusterMapper.class);

conf3.setReducerClass(clusterReducer.class);

Path inp3 = new Path(OUT\_PATH2);

Path out3 = new Path(OUT\_PATH3);

FileInputFormat.addInputPath(conf3, inp3);

FileOutputFormat.setOutputPath(conf3, out3);

JobClient.runJob(conf3);

while((t<2)||(percent>1))

{

FileSystem f=FileSystem.get(new Configuration());

Path p1=new Path("hdfs://localhost:54310/mydir/myfile.txt");

if(f.exists(p1))

f.delete(new Path( "hdfs://localhost:54310/mydir/myfile.txt"),true);

JobConf conf6 = new JobConf(getConf(), ClusterMain.class);

conf6.setJobName("Clustering");

conf6.setOutputKeyClass(IntWritable.class);

conf6.setOutputValueClass(IntWritable.class);

conf6.setMapperClass(MinorMapper.class);

conf6.setReducerClass(MinorReducer.class);

Path inp6;

if(t==0)

inp6 = new Path(OUT\_PATH3);

else

inp6=new Path(args[1]);

Path out6 = new Path(OUT\_PATH4);

FileInputFormat.addInputPath(conf6, inp6);

FileOutputFormat.setOutputPath(conf6, out6);

JobClient.runJob(conf6);

Path p=new Path("hdfs://localhost:54310/mydir/minor1.txt");

if(f.exists(p))

f.delete(new Path( "hdfs://localhost:54310/mydir/minor1.txt"),true);

Path p2=new Path("hdfs://localhost:54310/mydir/relp.txt");

if(f.exists(p2))

f.delete(new Path( "hdfs://localhost:54310/mydir/relp.txt"),true);

JobConf conf4 = new JobConf(getConf(), ClusterMain.class);

conf4.setJobName("Clustering");

conf4.setOutputKeyClass(IntWritable.class);

conf4.setOutputValueClass(IntWritable.class);

conf4.setMapperClass(Minor1Mapper.class);

conf4.setReducerClass(Minor1Reducer.class);

Path inp4 = new Path(OUT\_PATH4);

Path out4 = new Path(OUT\_PATH5);

FileInputFormat.addInputPath(conf4, inp4);

FileOutputFormat.setOutputPath(conf4, out4);

JobClient.runJob(conf4);

JobConf conf5 = new JobConf(getConf(), ClusterMain.class);

conf5.setJobName("Clustering");

conf5.setOutputKeyClass(IntWritable.class);

conf5.setOutputValueClass(Text.class);

conf5.setMapperClass(Minor2Mapper.class);

conf5.setReducerClass(Minor2Reducer.class);

if(f.exists(new Path("hdfs://localhost:54310/output/part-00000")))

f.delete(new Path("hdfs://localhost:54310/output"),true);

Path out5;

Path inp5 = new Path(OUT\_PATH5);

out5 = new Path(args[1]);

FileInputFormat.addInputPath(conf5, inp5);

FileOutputFormat.setOutputPath(conf5, out5);

JobClient.runJob(conf5);

if(t==0)

{

Path p3=new Path("hdfs://localhost:54310/output/part-00000");

BufferedReader br1=new BufferedReader(new InputStreamReader(f.open(p3)));

l=new LinkedList();

String line;

line=br1.readLine();

while(line!=null)

{

StringTokenizer token=new StringTokenizer(line);

l.add(token.nextToken());

line=br1.readLine();

}

}

else

{

Path p4=new Path("hdfs://localhost:54310/output/part-00000");

BufferedReader br2=new BufferedReader(new InputStreamReader(f.open(p4)));

l1=new LinkedList();

String line1;

line1=br2.readLine();

while(line1!=null)

{

StringTokenizer token1=new StringTokenizer(line1);

l1.add(token1.nextToken());

line1=br2.readLine();

}

}

j=0;

if(t!=0)

{

for(int i=0;i<l.size();i++)

{

if(!l.get(i).equals(l1.get(i)))

j++;

}

percent=(j/l.size())\*100;

l=new LinkedList(l1);

}

f.delete(new Path("hdfs://localhost:54310/user/hduser/intermediate\_output4"),true);

f.delete(new Path("hdfs://localhost:54310/user/hduser/intermediate\_output5"),true);

Minor1Mapper.f=0;

Minor2Mapper.f=0;

MinorMapper.f=0;

t++;

}

return 0;

}

public static void main(String[] args) throws Exception{

int res = ToolRunner.run(new Configuration(), new ClusterMain(),args);

System.exit(res);

}

}

**Stemmer**

For Stemming Lovins algorithm is used [12].

**Frequency Mapper**

import java.io.\*;

import java.util.StringTokenizer;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapred.\*;

public class FreqMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable>{

private static IntWritable one;

private Text word = new Text();

public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException

{

String line =value.toString();

StringTokenizer tokenizer = new StringTokenizer(line);

word.set(tokenizer.nextToken());

one=new IntWritable(Integer.parseInt(tokenizer.nextToken()));

}

}

**Frequency Reducer**

import java.io.IOException;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapred.\*;

public class FreqReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable>{

public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException

{

LinkedList a=new LinkedList();

LinkedList ch=new LinkedList();

int i=0;

while(values.hasNext())

{

a.add(values.next().get());

i++;

}

int j,k,l=0,m,flag,count=0;

for(j=0;j<i;j++)

{

count=0;

flag=0;

for(m=0;m<l;m++)

{

if((Integer)ch.get(m)==(Integer)a.get(j))

flag=1;

}

if(flag==0)

{

for(k=j;k<i;k++)

{

if((Integer)a.get(j)==(Integer)a.get(k))

count++;

}

ch.add((Integer)a.get(j));

l++;

String str=Integer.toString((Integer)a.get(j));

Text key1=new Text(str);

output.collect(key1, new IntWritable(count));

}

}

}

}

**Projection Mapper**

import java.io.\*;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapred.\*;

public class ProjectMapper extends MapReduceBase implements Mapper<LongWritable, Text, IntWritable, DoubleWritable>{

public void map(LongWritable key, Text value, OutputCollector<IntWritable, DoubleWritable> output, Reporter reporter) throws IOException

{

String line =value.toString();

StringTokenizer tokenizer = new StringTokenizer(line);

IntWritable one=new IntWritable(Integer.parseInt(tokenizer.nextToken()));

DoubleWritable two=new DoubleWritable(Double.parseDouble(tokenizer.nextToken()));

output.collect(one, two);

}

}

Projection Reducer

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapred.\*;

public class ProjectReducer extends MapReduceBase implements Reducer<IntWritable, DoubleWritable, IntWritable, DoubleWritable>{

public void reduce(IntWritable key, Iterator<DoubleWritable> values, OutputCollector<IntWritable, DoubleWritable> output, Reporter reporter) throws IOException

{

LinkedList a=new LinkedList();

int i=0;

while(values.hasNext())

{

a.add(values.next().get());

i++;

}

double max=(Double)a.get(0);

for(int j=0;j<i;j++)

{

if(max<(Double)a.get(j))

{

max=(Double)a.get(j);

}

}

double d=1+Math.log10(max);

output.collect(key, new DoubleWritable(d));

}

}

**Cluster Mapper**

import java.io.\*;

import java.lang.\*;

import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileStatus;

public class clusterMapper extends MapReduceBase implements Mapper<LongWritable, Text, IntWritable, IntWritable>{

static int f=0;

double[][] a;

public int[] min() {

double min = a[0][0];

int[] txt=new int[2];

for (int col = 0; col < a.length; col++) {

for (int row = 0; row < a[col].length; row++) {

if (min > a[col][row]) {

min = a[col][row];

txt[0]=col;

txt[1]=row;

}

}

}

return txt;

}

public void edistance (double[] freq,int n)

{

int i,j;

a=new double[n][n];

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

if(i==j)

a[i][j]=1000;

else

a[i][j]=Math.sqrt((freq[i]\*freq[i])-(freq[j]\*freq[j]));

}

}

}

public void map(LongWritable key, Text value, OutputCollector<IntWritable, IntWritable> output, Reporter reporter) throws IOException

{

f++;

if(f==1)

{

try

{

FileSystem fs = FileSystem.get(new Configuration());

Path p=new Path("hdfs://localhost:54310/user/hduser/intermediate\_output2/part-00000");

int j1=0;

double[] matrix=new double[1000];

BufferedReader br=new BufferedReader(new InputStreamReader(fs.open(p)));

String line;

line=br.readLine();

while (line != null)

{

StringTokenizer tokenizer=new StringTokenizer(line);

tokenizer.nextToken();

matrix[j1]=Double.parseDouble(tokenizer.nextToken());

j1++;

line=br.readLine();

}

double k,sum,sum1;

int[][] doc=new int[1000][1000];

double[] cent=new double[100000];

double[][] ed1=new double[1000][1000];

int[] clus = new int[5000];

int u=0;

List l=new LinkedList();

int[] avai=new int[100000];

int av=0;

int[] mat1=new int[2];

int i=0,j,flag=0;

HashMap h=new HashMap();

k=Math.floor(Math.sqrt(j1));

int[] matrix1=new int[j1];

edistance(matrix,j1);

mat1=min();

l.add(mat1[0]);

l.add(mat1[1]);

avai[av++]=mat1[0];

avai[av++]=mat1[1];

h.put(i,l);

i++;

while(i<k)

{

flag=0;

a[mat1[0]][mat1[1]]=1000;

mat1=min();

for(int s=0;s<av;s++)

{

if(avai[s]==mat1[0]||avai[s]==mat1[1])

flag=1;

}

if(flag==0)

{

avai[av++]=mat1[0];

avai[av++]=mat1[1];

l=new LinkedList();

l.add(mat1[0]);

l.add(mat1[1]);

h.put(i,l);

i++;

}

}

int size;

for(i=0;i<k;i++)

{

sum=0;

l=(LinkedList)h.get(i);

size=l.size();

for(j=0;j<size;j++)

{

sum=sum+matrix[(Integer)l.get(j)];

clus[u]=(Integer)l.get(j);

doc[j][u]=(Integer)l.get(j);

u++;

}

cent[i]=sum/size;

}

int rem=0;

for(i=0;i<j1;i++)

{

j=0;

while(j<2\*k)

{

if(i==clus[j])

break;

else

j++;

}

if(j>=2\*k)

{

matrix1[rem]=i;

rem++;

}

}

for(int co=0;co<rem;co++)

{

int inc=0;

while(inc<k)

{

ed1[co][inc]=Math.sqrt(Math.abs((matrix[matrix1[co]]\*matrix[matrix1[co]])-(cent[inc]\*cent[inc])));

inc++;

}

double mini = ed1[co][0];

int clu=0,row;

int[] txt=new int[2];

for (row = 0; row < inc; row++)

{

if (mini > ed1[co][row])

{

mini = ed1[co][row];

clu=row;

}

}

l=(LinkedList)h.get(clu);

l.add(matrix1[co]);

h.put(clu, l);

l=(LinkedList)h.get(clu);

sum1=0;

for(int sam=0;sam<l.size();sam++)

{

sum1=sum1+matrix[(Integer)l.get(sam)];

}

cent[clu]=sum1/l.size();

}

for(i=0;i<k;i++)

{

l=(LinkedList)h.get(i);

for(j=0;j<l.size();j++)

{

output.collect(new IntWritable(i),new IntWritable((Integer)l.get(j)));

}

}

}

catch(Exception e){}

}

}

}

**Cluster Reducer**

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.\*;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapred.\*;

public class clusterReducer extends MapReduceBase implements Reducer<IntWritable,IntWritable, IntWritable, IntWritable>{

public void reduce(IntWritable key, Iterator<IntWritable> values, OutputCollector<IntWritable, IntWritable> output, Reporter reporter) throws IOException

{

while(values.hasNext())

{

output.collect(key,values.next());

}

}

}

**Minor Mapper**

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapred.\*;

public class MinorMapper extends MapReduceBase implements Mapper<LongWritable, Text, IntWritable, IntWritable>{

static int f=0;

public void map(LongWritable key, Text value, OutputCollector<IntWritable, IntWritable> output, Reporter reporter) throws IOException

{

f++;

if(f==1)

{

String w;

int[][] doc=new int[100][100];

int i=0,j=0,u,v,t,r,k,s;

LinkedList docnum=new LinkedList();

LinkedList word=new LinkedList();

LinkedList l1,l2;

FileSystem fs2 = FileSystem.get(new Configuration());

Path p2=new Path("hdfs://localhost:54310/user/hduser/intermediate\_output3/part-00000");

BufferedReader br2=new BufferedReader(new InputStreamReader(fs2.open(p2)));

String li=br2.readLine();

int val,clus=0,tx;

LinkedList l3=new LinkedList();

LinkedList l5=new LinkedList();

while(li!=null)

{

StringTokenizer token=new StringTokenizer(li);

val=Integer.parseInt(token.nextToken());

if(!l3.contains(val))

clus++;

l3.add(val);

tx=Integer.parseInt(token.nextToken());

l5.add(tx);

li=br2.readLine();

}

int q=0;

int[] co=new int[clus];

for(i=0;i<clus;i++)

co[i]=0;

for(k=0;k<clus;k++)

{

q=0;

for(s=0;s<l3.size();s++)

{

if((Integer)l3.get(s)==k)

{

doc[k][q]=(Integer)l5.get(s);

co[k]++;

q++;

}

}

}

FileSystem fs = FileSystem.get(new Configuration());

Path p=new Path("hdfs://localhost:54310/user/hduser/intermediate\_output/part-00000");

BufferedReader br=new BufferedReader(new InputStreamReader(fs.open(p)));

FileSystem fs3 = FileSystem.get(new Configuration());

Path p3=new Path("hdfs://localhost:54310/mydir/myfile.txt");

if(!fs3.exists(p3))

{

fs3.createNewFile(p3);

}

FSDataOutputStream fileOutputStream = fs3.append(p3);

BufferedWriter br3=new BufferedWriter(new OutputStreamWriter(fileOutputStream));

String line;

LinkedList l=new LinkedList();

HashMap h=new HashMap();

line=br.readLine();

while (line != null)

{

StringTokenizer tokenizer=new StringTokenizer(line);

word.add(tokenizer.nextToken());

docnum.add(Integer.parseInt(tokenizer.nextToken()));

j++;

line=br.readLine();

}

for(u=0;u<l5.size();u++)

{

l=new LinkedList();

for(v=0;v<j;v++)

{

if((Integer)l5.get(u)==(Integer)docnum.get(v))

{

l.add((String)word.get(v));

}

}

h.put((Integer)l5.get(u),l);

}

int[][] count=new int[10][10000];

for(i=0;i<clus;i++)

{

l=(LinkedList)h.get(doc[i][0]);

for(u=1;u<co[i];u++)

{

l1=(LinkedList)h.get(doc[i][u]);

l2=new LinkedList();

for(t=0;t<l.size();t++)

{

for(r=0;r<l1.size();r++)

{

if(l.get(t).equals(l1.get(r)))

{

l2.add(l.get(t));

}

}

}

l=new LinkedList(l2);

}

int[] count1=new int[l.size()];

w=(String)word.get(0);

for(s=0;s<l.size();s++)

{

for(int n=0;n<j;n++)

{

w=(String)word.get(n);

if(w.equals((String)l.get(s)))

{

for(u=0;u<co[i];u++)

{

if((Integer)docnum.get(n)==doc[i][u])

count[i][s]++;

}

}

}

}

for(int td=0;td<l5.size();td++)

{

for(int ini=0;ini<l.size();ini++)

count1[ini]=0;

for(s=0;s<l.size();s++)

{

for(int n=0;n<j;n++)

{

w=(String)word.get(n);

if(w.equals((String)l.get(s))&&(Integer)docnum.get(n)==td)

{

count1[s]++;

}

}

}

int sum=0,product=1;

int sqr=0,sqr1=0;

double deno=0;

for(int sa=0;sa<l.size();sa++)

{

product=count[i][sa]\*count1[sa];

sum=sum+product;

sqr=sqr+(count[i][sa]\*count[i][sa]);

sqr1=sqr1+(count1[sa]\*count1[sa]);

}

deno=Math.sqrt((double)sqr\*(double)sqr1);

double cosine,zero=0;

cosine=sum/deno;

output.collect(new IntWritable(2),new IntWritable(1));

}

}

br3.close();

}

}

}

**Minor Reducer**

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapred.\*;

public class MinorReducer extends MapReduceBase implements Reducer<IntWritable,IntWritable, IntWritable, IntWritable>{

public void reduce(IntWritable key, Iterator<IntWritable> values, OutputCollector<IntWritable, IntWritable> output, Reporter reporter) throws IOException

{

while(values.hasNext())

{

output.collect(key,values.next());

}

}

}

**Minor 1 Mapper**

import java.io.\*;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapred.\*;

public class Minor1Mapper extends MapReduceBase implements Mapper<LongWritable, Text, IntWritable, IntWritable>{

static int f=0;

public void map(LongWritable key, Text value, OutputCollector<IntWritable, IntWritable> output, Reporter reporter) throws IOException

{

int i,j,clus,fl=0;

double max,c;

f++;

if(f==1)

{

FileSystem fs = FileSystem.get(new Configuration());

Path p=new Path("hdfs://localhost:54310/mydir/myfile.txt");

BufferedReader br=new BufferedReader(new InputStreamReader(fs.open(p)));

String line=br.readLine();

LinkedList l=new LinkedList();

LinkedList l1=new LinkedList();

LinkedList l2=new LinkedList();

while(line!=null)

{

StringTokenizer tokenizer=new StringTokenizer(line);

l.add(tokenizer.nextToken());

l1.add(tokenizer.nextToken());

l2.add(tokenizer.nextToken());

line=br.readLine();

fl++;

}

FileSystem fs1 = FileSystem.get(new Configuration());

FileStatus[] status = fs1.listStatus(new Path("hdfs://localhost:54310/input"));

FileSystem fs2 = FileSystem.get(new Configuration());

Path p2=new Path("hdfs://localhost:54310/mydir/minor1.txt");

if(!fs2.exists(p2))

{

fs2.createNewFile(p2);

}

FSDataOutputStream fileOutputStream = fs2.append(p2);

BufferedWriter br2=new BufferedWriter(new OutputStreamWriter(fileOutputStream));

for(i=0;i<status.length;i++)

{

max=Double.parseDouble(l2.get(i).toString());

clus=Integer.parseInt(l.get(i).toString());

for(j=i;j<l1.size();j++)

{

if(i==(Integer.parseInt(l1.get(j).toString())))

{

c=Double.parseDouble(l2.get(j).toString());

if(c>max)

{

max=c;

clus=Integer.parseInt(l.get(j).toString());

}

}

}

br2.append(clus+"\t"+i+"\n");

output.collect(new IntWritable(clus),new IntWritable(i));

}

br2.close();

}

}

}

**Minor 1 Reducer**

import java.io.BufferedReader;

public class Minor1Reducer extends MapReduceBase implements Reducer<IntWritable,IntWritable, IntWritable, IntWritable>{

public void reduce(IntWritable key, Iterator<IntWritable> values, OutputCollector<IntWritable, IntWritable> output, Reporter reporter) throws IOException

{

FileSystem fs = FileSystem.get(new Configuration());

FileStatus[] status = fs.listStatus(new Path("hdfs://localhost:54310/input"));

String line,head,name;

HashMap h=new HashMap();

LinkedList l;

LinkedList txt=new LinkedList();

LinkedList l1=new LinkedList();

int k,j=0,i;

char[] c=new char[5000];

char[] c1=new char[5000];

FileSystem fs2 = FileSystem.get(new Configuration());

Path p2=new Path("hdfs://localhost:54310/mydir/relp.txt");

if(!fs2.exists(p2))

{

fs2.createNewFile(p2);

}

FSDataOutputStream fileOutputStream = fs2.append(p2);

BufferedWriter br2=new BufferedWriter(new OutputStreamWriter(fileOutputStream));

for (i=0;i<status.length;i++)

{

l=new LinkedList();

BufferedReader br=new BufferedReader(new InputStreamReader(fs.open(status[i].getPath())));

line=br.readLine();

while(line!=null)

{

j=0;

StringTokenizer tokenizer=new StringTokenizer(line);

head=tokenizer.nextToken();

if(head.equals("Author:"))

{

while(tokenizer.hasMoreTokens())

{

c=tokenizer.nextToken().toCharArray();

for(k=0;k<c.length;k++)

{

if(c[k]==',')

{

name=new String(c1,0,j);

j=0;

if(!l1.contains(name))

{

l1.add(name);

}

l.add(name);

}

else

c1[j++]=c[k];

}

if(j!=0)

c1[j++]=' ';

if(!tokenizer.hasMoreTokens())

{

name=new String(c1,0,j-1);

if(!l1.contains(name))

{

l1.add(name);

}

l.add(name);

}

}

}

line=br.readLine();

}

h.put(i, l);

}

while(values.hasNext())

{

txt.add(values.next().get());

}

String check;

LinkedList l2;

double rp;

double num=0,den=0;

for(i=0;i<l1.size();i++)

{

check=(String)l1.get(i);

for(j=0;j<txt.size();j++)

{

l2=new LinkedList();

l2=(LinkedList)h.get((Integer)txt.get(j));

if(l2.contains(check))

{

num++;

}

}

for(j=0;j<status.length;j++)

{

l2=new LinkedList();

l2=(LinkedList)h.get(j);

if(l2.contains(check))

den++;

}

rp=num/den;

br2.append(check+"\t"+key+"\t"+rp+"\n");

output.collect(key,new IntWritable(i));

num=0;den=0;

}

br2.close();

}

}

Minor 2 Mapper

import java.io.\*;

public class Minor2Mapper extends MapReduceBase implements Mapper<LongWritable, Text, IntWritable, Text>{

static int f=0;

public void map(LongWritable key, Text value, OutputCollector<IntWritable, Text> output, Reporter reporter) throws IOException

{

f++;

if(f==1)

{

int fl=0;

FileSystem fs1 = FileSystem.get(new Configuration());

Path p=new Path("hdfs://localhost:54310/mydir/relp.txt");

BufferedReader br=new BufferedReader(new InputStreamReader(fs1.open(p)));

String line=br.readLine();

String ch,nt=null;

LinkedList l=new LinkedList();

LinkedList l1=new LinkedList();

LinkedList l2=new LinkedList();

while(line!=null)

{

StringTokenizer tokenizer=new StringTokenizer(line);

ch=tokenizer.nextToken();

while(tokenizer.hasMoreTokens())

{

nt=tokenizer.nextToken();

if(nt.matches("[a-zA-Z.]+"))

{

ch=ch.concat(" "+nt);

}

else

break;

}

l.add(ch);

l1.add(nt);

l2.add(Double.parseDouble(tokenizer.nextToken()));

line=br.readLine();

fl++;

}

int d,i,j,m;

String name;

for(d=1;d<fl;d++)

{

if(l.get(0).equals((String)l.get(d)))

break;

}

double[] sum=new double[d];

double avg=0,var=0,sd,thresh;

for(i=0;i<d;i++)

{

name=(String)l.get(i);

sum[i]=0;

for(j=i;j<fl;j++)

{

if(l.get(i).equals(l.get(j)))

sum[i]=sum[i]+((Double)l2.get(j)\*(Double)l2.get(j));

}

avg=avg+sum[i];

}

avg=avg/d;

for(i=0;i<d;i++)

{

var=var+(sum[i]-avg)\*(sum[i]-avg);

}

sd=Math.sqrt(var/d);

thresh=avg-(1.5\*sd);

LinkedList l3=new LinkedList();

for(m=0;m<d;m++)

{

if(sum[m]>=thresh)

{

l3.add(l.get(m));

}

}

for(m=0;m<l3.size();m++)

{

for(r=0;r<l4.size();r++)

{

if(l3.get(m).equals((String)l4.get(r)))

{

output.collect(new IntWritable(i),new Text(l3.get(m).toString()));

}

}

}

}

}

}

}

**Minor 2 Reducer**

import java.io.BufferedReader;

import java.io.IOException;

public class Minor2Reducer extends MapReduceBase implements Reducer<IntWritable, Text, IntWritable, Text>{

public void reduce(IntWritable key, Iterator<Text> values, OutputCollector<IntWritable, Text> output, Reporter reporter) throws IOException

{

LinkedList l=new LinkedList();

LinkedList l2;

LinkedList l3=new LinkedList();

LinkedList l5=new LinkedList();

double[] count;

double pro=1;

int k1=0;

while(values.hasNext())

{

l.add(values.next().toString());

k1++;

}

FileSystem fs = FileSystem.get(new Configuration());

FileStatus[] status = fs.listStatus(new Path("hdfs://localhost:54310/input"));

String line1,head,name1;

HashMap h=new HashMap();

LinkedList l4;

LinkedList txt=new LinkedList();

int k,r,i,j,n,clus=0,s;

j=0;

FileSystem fs1 = FileSystem.get(new Configuration());

Path p=new Path("hdfs://localhost:54310/mydir/minor1.txt");

BufferedReader br=new BufferedReader(new InputStreamReader(fs.open(p)));

String line=br.readLine();

int val;

while(line!=null)

{

StringTokenizer token=new StringTokenizer(line);

val=Integer.parseInt(token.nextToken());

if(!l3.contains(val))

clus++;

l3.add(val);

l5.add(Integer.parseInt(token.nextToken()));

line=br.readLine();

}

char[] c=new char[500];

char[] c1=new char[500];

double z=0,nor=0;

double oc;

double[] arr=new double[clus];

double[] pos=new double[clus];

count=new double[l.size()];

for(s=0;s<clus;s++)

{

arr[s]=1;

}

for(j=0;j<l.size();j++)

{

count[j]=0;

double ruf=Math.floor(Math.sqrt(status.length));

double[] count1=new double[(int)ruf];

for(i=0;i<status.length;i++)

{

l2=(LinkedList)h.get(i);

if(l2.contains(l.get(j)))

{

count[j]++;

count1[(Integer)l3.get(i)]++;

}

}

for(s=0;s<clus;s++)

{

arr[s]=arr[s]\*(count1[s]/count[j]);

}

pro=pro\*(1/count[j]);

}

for(i=0;i<clus;i++)

{

oc=Collections.frequency(l3, i);

pos[i]=(oc/status.length)\*(arr[i]/pro);

nor=nor+pos[i];

}

for(i=0;i<clus;i++)

{

pos[i]=pos[i]/nor;

}

double max=pos[0];int cl=0;

for(i=0;i<clus;i++)

{

if(max<pos[i])

{

cl=i;

}

}

int sample=key.get();

String s1=Integer.toString(sample);

Text t=new Text(s1);

output.collect(new IntWritable(cl),t);

}

}

**8.3.2. Classification**

**Feature Mapper**

import java.io.\*;

import java.util.HashMap;

import org.apache.hadoop.mapred.\*;

public class FeatureMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable>{

static int f=0;

static HashMap h;

static LinkedList lal1;

public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException

{

f++;

if(f==1)

{

FileSystem fs = FileSystem.get(new Configuration());

Path p=new Path("hdfs://localhost:54310/cdir/classlabels.txt");

BufferedReader br=new BufferedReader(new InputStreamReader(fs.open(p)));

String line=br.readLine();

String file,line1;

String label;

FileSystem fs1 = FileSystem.get(new Configuration());

FileStatus[] status = fs1.listStatus(new Path("hdfs://localhost:54310/input"));

h=new HashMap();

LinkedList lal,lal2;

lal1=new LinkedList();

int i,j=0;

char[] tmp;

int k=0;

char[] t=new char[1000];

while(line!=null)

{

lal=new LinkedList();

StringTokenizer token=new StringTokenizer(line);

file=token.nextToken();

label=token.nextToken();

for(i=0;i<status.length;i++)

{

BufferedReader br1=new BufferedReader(new InputStreamReader(fs1.open(status[i].getPath())));

line1=br1.readLine();

while(line1!=null)

{

StringTokenizer to=new StringTokenizer(line1);

if(to.nextToken().equals("URL:"))

{

if(file.equals(to.nextToken()))

k=i;

}

line1=br1.readLine();

}

}

tmp=label.toCharArray();

for(i=0;i<label.length();i++)

{

if(tmp[i]=='/')

{

String s=new String(t,0,j);

lal.add(s);

if(!lal1.contains(s))

lal1.add(s);

j=0;

}

else

{

t[j]=tmp[i];

j++;

}

}

h.put(k,lal);

line=br.readLine();

}

HashMap h1=new HashMap();

for(i=0;i<lal1.size();i++)

{

lal2=new LinkedList();

for(j=0;j<status.length;j++)

{

lal=(LinkedList)h.get(j);

if(lal.contains(lal1.get(i)))

{

lal2.add(j);

}

}

h1.put(lal1.get(i),lal2);

}

String check;

LinkedList l3;

double num=0; double avg=0;

double den=0,rp=0;

double[] gini=new double[l1.size()];

for(i=0;i<l1.size();i++)

gini[i]=0;

for(i=0;i<l1.size();i++)

{

den=0;

l2=new LinkedList();

check=(String)l1.get(i);

for(j=0;j<status.length;j++)

{

l=(LinkedList)h2.get(j);

if(l.contains(check))

{

l2.add(j);

den++;

}

}

for(j=0;j<lal1.size();j++)

{

num=0;

l3=(LinkedList)h1.get(lal1.get(j));

for(int m=0;m<l2.size();m++)

{

if(l3.contains((l2.get(m))))

{

num++;

}

}

rp=num/den;

gini[i]=gini[i]+(rp\*rp);

}

avg=avg+gini[i];

}

avg=avg/l1.size();

double var=0,sd;

for(i=0;i<l1.size();i++)

{

var=var+((gini[i]-avg)\*(gini[i]-avg));

}

sd=Math.sqrt(var/l1.size());

double thres=avg-sd;

FileSystem fs2 = FileSystem.get(new Configuration());

Path p2=new Path("hdfs://localhost:54310/cdir/valid.txt");

if(!fs2.exists(p2))

fs2.createNewFile(p2);

BufferedWriter br2=new BufferedWriter(new OutputStreamWriter(fs2.append(p2)));

for(i=0;i<l1.size();i++)

{

if(gini[i]>=thres)

{

br2.write((String)l1.get(i)+"\n");

output.collect(new Text((String)l1.get(i)),new IntWritable(1));

}

}

br2.close();

}

}

}

**Feature Reducer**

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapred.\*;

public class FeatureReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable>{

public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException

{

while(values.hasNext())

{

output.collect(key, values.next());

}

}

}

**Classi Mapper**

import java.io.\*;

import java.util.\*;

import org.apache.hadoop.mapred.\*;

public class ClassiMapper extends MapReduceBase implements Mapper<LongWritable, Text, IntWritable, Text> {

static int f=0;

public void map(LongWritable key, Text value, OutputCollector<IntWritable, Text> output, Reporter reporter) throws IOException

{

f++;

if(f==1)

{

String w;

int[][] doc=new int[100][100];

int i=0,j=0,u,v,t,r,k,s;

LinkedList docnum=new LinkedList();

LinkedList word=new LinkedList();

LinkedList l1,l2,l6;

FileSystem fs2 = FileSystem.get(new Configuration());

Path p2=new Path("hdfs://localhost:54310/user/hduser/intermediateoutput6/part-00000");

BufferedReader br2=new BufferedReader(new InputStreamReader(fs2.open(p2)));

String li=br2.readLine();

int val,clus=0,tx;

LinkedList l3=new LinkedList();

LinkedList l5=new LinkedList();

while(li!=null)

{

StringTokenizer token=new StringTokenizer(li);

val=Integer.parseInt(token.nextToken());

if(!l3.contains(val))

clus++;

l3.add(val);

tx=Integer.parseInt(token.nextToken());

l5.add(tx);

li=br2.readLine();

}

int q=0;

int[] co=new int[clus];

for(i=0;i<clus;i++)

co[i]=0;

for(k=0;k<clus;k++)

{

q=0;

for(s=0;s<l3.size();s++)

{

if((Integer)l3.get(s)==k)

{

doc[k][q]=(Integer)l5.get(s);

co[k]++;

q++;

}

}

}

FileSystem fs = FileSystem.get(new Configuration());

Path p=new Path("hdfs://localhost:54310/user/hduser/intermediate\_output/part-00000");

BufferedReader br=new BufferedReader(new InputStreamReader(fs.open(p)));

String line,aline;

LinkedList l=new LinkedList();

HashMap h=new HashMap();

line=br.readLine();

while (line != null)

{

StringTokenizer tokenizer=new StringTokenizer(line);

word.add(tokenizer.nextToken());

docnum.add(Integer.parseInt(tokenizer.nextToken()));

j++;

line=br.readLine();

}

FileSystem fs1 = FileSystem.get(new Configuration());

Path p1=new Path("hdfs://localhost:54310/mydir/minor1.txt");

BufferedReader br1=new BufferedReader(new InputStreamReader(fs1.open(p1)));

String line1;

FileSystem fs3 = FileSystem.get(new Configuration());

Path p3=new Path("hdfs://localhost:54310/mydir/myfile.txt");

if(!fs3.exists(p3))

{

fs3.createNewFile(p3);

}

FSDataOutputStream fileOutputStream = fs3.append(p3);

BufferedWriter br3=new BufferedWriter(new OutputStreamWriter(fileOutputStream));

FileSystem fs4 = FileSystem.get(new Configuration());

Path p4=new Path("hdfs://localhost:54310/mydir/author.txt");

BufferedReader br5;

for(u=0;u<l5.size();u++)

{

l=new LinkedList();

for(v=0;v<j;v++)

{

if((Integer)l5.get(u)==(Integer)docnum.get(v))

{

l.add((String)word.get(v));

}

}

h.put((Integer)l5.get(u),l);

}

int[][] count=new int[10][10000];

for(i=0;i<clus;i++)

{

l=(LinkedList)h.get(doc[i][0]);

for(u=1;u<co[i];u++)

{

l1=(LinkedList)h.get(doc[i][u]);

l2=new LinkedList();

for(t=0;t<l.size();t++)

{

for(r=0;r<l1.size();r++)

{

if(l.get(t).equals(l1.get(r)))

{

l2.add(l.get(t));

}

}

}

l=new LinkedList(l2);

}

int[] count1=new int[l.size()];

w=(String)word.get(0);

for(s=0;s<l.size();s++)

{

for(int n=0;n<j;n++)

{

w=(String)word.get(n);

if(w.equals((String)l.get(s)))

{

for(u=0;u<co[i];u++)

{

if((Integer)docnum.get(n)==doc[i][u])

count[i][s]++;

}

}

}

}

for(int td=0;td<l5.size();td++)

{

for(int ini=0;ini<l.size();ini++)

count1[ini]=0;

for(s=0;s<l.size();s++)

{

for(int n=0;n<j;n++)

{

w=(String)word.get(n);

if(w.equals((String)l.get(s))&&(Integer)docnum.get(n)==td)

{

count1[s]++;

}

}

}

int sum=0,product=1;

int sqr=0,sqr1=0;

double deno=0;

for(int sa=0;sa<l.size();sa++)

{

product=count[i][sa]\*count1[sa];

sum=sum+product;

sqr=sqr+(count[i][sa]\*count[i][sa]);

sqr1=sqr1+(count1[sa]\*count1[sa]);

}

deno=Math.sqrt((double)sqr\*(double)sqr1);

double cosine,zero=0;

cosine=sum/deno;

if(deno!=0)

br3.append(i+"\t"+td+"\t"+cosine+"\n");

else

br3.append(i+"\t"+td+"\t"+zero+"\n");

}

}

br3.close();

BufferedReader br4;

String ch,nt;

StringTokenizer token,token1,token2;

int[] R=new int[clus];

int[] Rcount=new int[l5.size()];

int[] cl=new int[clus];

int clus\_num,clnum,z,rm;

double val1=0;

double[] cos=new double[clus];

for(i=0;i<l5.size();i++)

{

k=0;z=0;

line=br1.readLine();

token=new StringTokenizer(line);

clus\_num=Integer.parseInt(token.nextToken());

br4=new BufferedReader(new InputStreamReader(fs3.open(p3)));

for(j=0;j<clus\*l5.size();j++)

{

li=br4.readLine();

token1=new StringTokenizer(li);

clnum=Integer.parseInt(token1.nextToken());

if(i==Integer.parseInt(token1.nextToken()))

{

if(clnum==clus\_num)

{

R[k]=clnum;

k++;

val1=Double.parseDouble(token1.nextToken());

}

else

{

cos[z]=Double.parseDouble(token1.nextToken());

cl[z]=clnum;

z++;

}

}

}

for(int op=0;op<z;op++)

{

if((cos[op]>val1-0.2)&&(cos[op]<val1+0.2))

{

R[k]=cl[op];

k++;

}

}

l6=new LinkedList();

Rcount[i]=k;

for(int op=0;op<k;op++)

{

String sample=String.valueOf(R[op]);

output.collect(new IntWritable(i), new Text(sample));

}

br4.close();

}

br1.close();

}

}

}

**Classi Reducer**

import java.io.BufferedReader;

public class ClassiReducer extends MapReduceBase implements Reducer<IntWritable,Text, IntWritable, Text>{

public void reduce(IntWritable key, Iterator<Text> values, OutputCollector<IntWritable, Text> output, Reporter reporter) throws IOException

{

LinkedList clus1=new LinkedList();

while(values.hasNext())

{

String va=values.next().toString();

int z1=Integer.parseInt(va);

clus1.add(z1);

}

FileSystem fs4 = FileSystem.get(new Configuration());

Path p4=new Path("hdfs://localhost:54310/mydir/author.txt");

BufferedReader br5;

br5=new BufferedReader(new InputStreamReader(fs4.open(p4)));

String aline=br5.readLine();

StringTokenizer token2;

int rm,i=key.get();

String ch,nt;

LinkedList l=new LinkedList();

while(aline!=null)

{

token2=new StringTokenizer(aline);

rm=Integer.parseInt(token2.nextToken());

if(rm==i)

{

ch=token2.nextToken();

while(token2.hasMoreTokens())

{

nt=token2.nextToken();

ch=ch.concat(" "+nt);

}

l.add(ch);

}

aline=br5.readLine();

}

br5.close();

LinkedList l2;

LinkedList l3=new LinkedList();

LinkedList l5=new LinkedList();

double[] count;

double pro=1;

int k1=0;

FileSystem fs = FileSystem.get(new Configuration());

FileStatus[] status = fs.listStatus(new Path("hdfs://localhost:54310/input"));

String line1,head,name1;

HashMap h=new HashMap();

LinkedList l4,l6=new LinkedList();

LinkedList txt=new LinkedList();

int k,r,j,n,clus=0,s;

j=0;

FileSystem fs1 = FileSystem.get(new Configuration());

Path p=new Path("hdfs://localhost:54310/user/hduser/intermediate\_output6/part-00000");

BufferedReader br=new BufferedReader(new InputStreamReader(fs.Open(p)));

String line=br.readLine();

int val;

while(line!=null)

{

StringTokenizer token=new StringTokenizer(line);

val=Integer.parseInt(token.nextToken());

if(!l3.contains(val))

clus++;

l3.add(val);

l5.add(Integer.parseInt(token.nextToken()));

line=br.readLine();

}

double z=0,nor=0;

double oc;

double[] arr=new double[clus];

double[] pos=new double[clus];

int[] sort=new int[clus];

count=new double[l.size()];

for(s=0;s<clus;s++)

{

arr[s]=1;

}

for(j=0;j<l.size();j++)

{

count[j]=0;

double ruf=Math.floor(Math.sqrt(status.length));

double[] count1=new double[(int)ruf];

for(i=0;i<status.length;i++)

{

l2=(LinkedList)h.get(i);

if(l2.contains(l.get(j)))

{

count[j]++;

count1[(Integer)l3.get(i)]++;

}

}

for(s=0;s<clus;s++)

{

arr[s]=arr[s]\*(count1[s]/count[j]);

}

pro=pro\*(1/count[j]);

}

for(i=0;i<clus;i++)

{

oc=Collections.frequency(l3, i);

pos[i]=(oc/status.length)\*(arr[i]/pro);

nor=nor+pos[i];

}

for(i=0;i<clus;i++)

{

pos[i]=pos[i]/nor;

sort[i]=i;

}

double temp, large;

int o=0,cc,d;

for(i=0;i<clus;i++)

{

for(j=0;j<clus;j++)

{

if(pos[i]>pos[j])

{

temp=pos[j];

d=sort[j];

pos[j]=pos[i];

sort[j]=sort[i];

pos[i]=temp;

sort[i]=d;

}

}

}

d=clus1.size();

LinkedList doc,label,lab;

label=new LinkedList();

for(i=0;i<d;i++)

clus1.add(sort[i]);

for(i=0;i<clus1.size();i++)

{

doc=new LinkedList();

for(j=0;j<l5.size();j++)

{

if(clus1.get(i).equals(l3.get(j)))

doc.add(l5.get(j));

}

for(j=0;j<doc.size();j++)

{

lab=(LinkedList)FeatureMapper.h.get((Integer)doc.get(j));

label.addAll(lab);

}

}

int si=FeatureMapper.lal1.size();

String la;

int[] oc1=new int[si];

for(i=0;i<si;i++)

{

la=FeatureMapper.lal1.get(i).toString();

oc1[i]=Collections.frequency(label, la);

}

int large1=oc1[0],d1=0;

for(i=0;i<si;i++)

{

if(oc1[i]>large1)

{

large1=oc1[i];

d1=i;

}

}

l6.add(FeatureMapper.lal1.get(d1));

for(i=0;i<si;i++)

{

if((large1==oc1[i])&&(!l6.contains(FeatureMapper.lal1.get(i))))

l6.add(FeatureMapper.lal1.get(i));

}

String line2;

int[] cw=new int[l6.size()];

for(i=0;i<l6.size();i++)

cw[i]=0;

if(l6.size()!=1)

{

LinkedList l7=(LinkedList)FeatureMapper.h.get(key.get());

for(i=0;i<l6.size();i++)

{

if(l7.contains(l6.get(i)))

{

d1=FeatureMapper.lal1.indexOf(l6.get(i));

break;

}

}

}

output.collect(key,new Text(FeatureMapper.lal1.get(d1).toString()));

}

}

**REFERENCES**



**CHAPTER 8**

**REFERENCES**

[1] Tapas Kanungo,David M.Mount,Nathan S.Netanyahu,Christine D.Piatko,Ruth Silverman and Abgela Y.Wu, "An Efficient k-Means Clustering Algorithm : Analysis and Implementation,IEEE Transactions on Pattern Analysis and Machine Intelligence ,Vol 24,No.7 July 2002.

[2] Raghuvira Pratap A,K Suvarna Vani,J Rama Devi and Dr. K Nageswara Rao,"An Efficient Density based Improved K-Medoids Clustering Algorithm, International Journal of Advanced Computer Science and Applications,Vol.2 ,No.6 ,2011.

[3] T.Liu,S.Liu,Z.Chen and W.Y.Ma,"An evaluation of feature selection for text clustering,"in Proc.ICML Conf,Washington,DC,USA,2003.

[4] K.Sasirekha,P.Baby,"Agglomerative Hierarchical Clustering Algorithm-A Review",International Journal of Scientific and Research Publications,Volume 3,Issue 3,March 2013.

[5]Inderjit S.Dhillon,Subramanyam Mallela,Rahul Kumar,"A Divisive Information-theoretic Feature Clustering Algorithm for Text Classification",Journal of Machine Learning Research,Vol 3 ,2003.

[6] Martin Ester,Hans-Peter Kriegel,Jorg Sander,Xiaowei Xu,"A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", KDD-96 Proceedings.

[7]S.L.Ting,W.H.Ip,Albert H.C.Tsang,"Is Navie Bayes a Good Classifier for

Document Classification?",International Journal of Software Engineering and Its Applications,Vol.5,no.3,July 2011.

[8] S.Rasoul Safavian and David Landgrebe,"A Survey of Decision Tree Classifier Methodology".

[9] Durgesh K Srivastava,Lekha Bhambhu,"Data Classification Using Support Vector Machine",Journal of Theoretical and Applied Information Technology.

[10]H.Schutze and C.Silverstein,"Projections for efficient document clustering," in Proc.ACM SIGIR Conf,New York,NY,USA,1997.

[11]G.Salton,"An introduction to Modern Information Retrieval",London, UK,

McGraw Hill,1983.

[12]http://www.cs.waikato.ac.nz/~eibe/stemmers/index.html