Volume 5, Issue 5, May 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Study of Classification Algorithms using MapReduce Framework

[1]NV Maha Lakshmi, [2]L Kanya Kumari, [3]A Rama Satish
[1,2]Assistant Professor, CSE Department, DVR & Dr HS MIC College of Technology, Kanchikacherla, India
[3] Associate Professor, CSE Department, DVR & Dr HS MIC College of Technology, Kanchikacherla, India

***Abstract*— Today there is not only a need for efficient data mining techniques to process large volume of data but also a need for a means to meet the computational requirements to process such huge volume of data. As an important task of data mining, classification plays an important role in information retrieval, web searching, CRM, etc. Most of the present classification techniques are serial, which become impractical for large data set.** *The computing resource is underutilized and the executing time is not waitable. In this paper we explore the parallel implementation methods of several classification algorithms based on a distributed computing environment running Apache Hadoop that uses the MapReduce paradigm to process high volume data. We also compare classification algorithms such as K-Nearest Neighbors, Naive Bayes model and Decision tree based on traditional models and MapReduce models.*

***Keywords*— k-Nearest Neighbor, Naïve Bayes, Hadoop, Mapreduce, Classification,** *decision tree.*

I. INTRODUCTION

Big data is defined as large amount of data which requires new technologies and architectures so that it becomes possible to extract value from it by capturing and analysis process. Due to such large size of data it becomes very difficult to perform effective analysis using the existing traditional techniques. Data mining algorithms are divided in four classes, including association rule learning, Clustering, Classification and Regression [1]. Classification algorithm deals with associating an unknown structure to a well known structure which is an important data mining problem[6]. It now, many classification algorithms have been proposed for big data. Many of them have limitation and weakness. Such as: low performance in large dataset, poor run-time performance when the training set is large, high Computation cost. To cover these limitations, many researchers using classification algorithm based on MapReduce. The MapReduce model was developed by Google to run data-intensive applications on a distributed infrastructure like commodity cluster. Rest of the paper is organized as follows. Section II presents the MapReduce. Section III will describe the classification algorithm based on MapReduce. Finally, the overall conclusions of this study are presented in section IV.

II. MAPREDUCE FRAMEWORK

MapReduce is a programming model used in clusters that have numerous nodes and use considerable computing resources to manage large amounts of data in parallel. MapReduce is proposed by Google in 2004. In the MapReduce model, an application to be executed is called a "job". A job can be divided into two parts: "map tasks" and "reduce tasks", in which the map-tasks run the map function and the reduce-tasks run the reduce function. Map function processes input data assigned by the master and produce many intermediate _key, value_ pairs. Based on _key, value_ pairs that are generated by map function processes, the reduce function then merges, sorts, and finally returns the result.

The MapReduce model mainly entails applying the idea of divide and conquer. It distributes a large amount of data to many nodes to perform parallel processing, which reduces the execution time and improves the performance. At runtime, input data are divided into many of the same sized data blocks; these blocks are then assigned to nodes that perform the same map function in parallel. After the map function is performed, the generated output is an intermediate datum composed of several _key, value_ pairs. The nodes that perform the reduce function obtain these intermediate data, and finally generate the output data. Fig. 1 shows the MapReduce flow chart.

The advantage of MapReduce is that it is easy to use. By using this model, many parallel computing details are hidden. The system automatically assigns nodes that differ from the mapper and reducer for computing. When programming, a programmer does not need to spend extra time on data and program division. Therefore, a programmer

does not need to have a deep understanding of parallel computing. A programmer must simply focus on the normal function processing rather than the parallel processing. This can simplify the application development process substantially and shorten the development time[2].

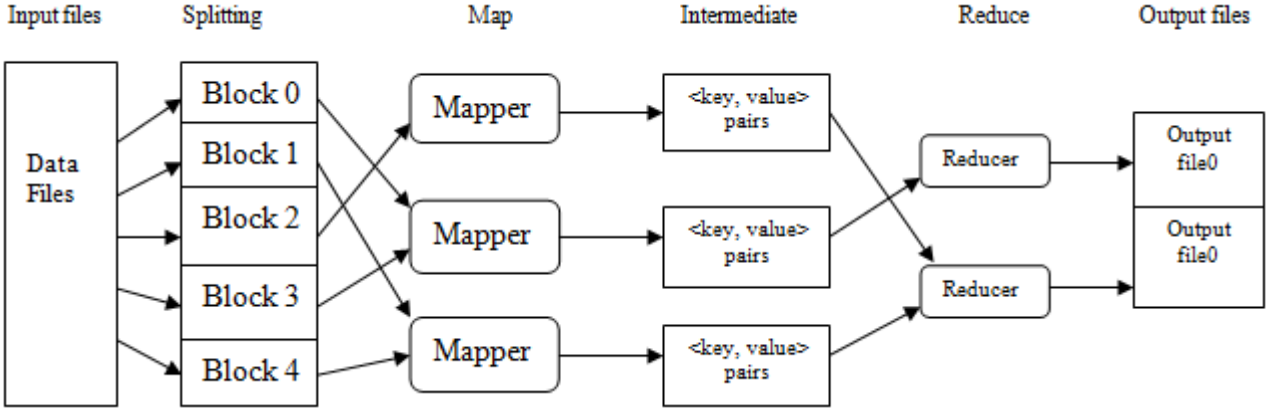

Fig.1.The overview of the MapReduce model.

III. CLASSIFICATION ALGORITHMS BASED ON MAPREDUCE

In this section we briefly introduce some of the Classification Algorithm.

A. K Nearest Neighbor

K Nearest Neighbor (KNN) is one of the most widely used classification Algorithm in data mining, which based on learning by analogy, that is by comparing a given test tuple with training tuples which are similar to it[3]. KNN classification determines the decision boundary locally that was developed from the need to perform discriminate analysis when reliable parametric estimates of probability densities are unknown or difficult to determine. If k = 1, then the object is simply assigned to the class of its nearest Neighbor. In practical applications, k is in units or tens rather than in hundreds or thousands [3]. K is a user defined constant, and an unlabeled vector or test point is classified by assigning the label which is most frequent among the k training samples nearest to that query point. In k-NN classification, the output is a class membership. Many researches are proposed based on the centralized paradigm where the KNN join is performed on a single, centralized server. Parallelization KNN algorithm improves the classification efficiency [3].

Prior to implementing the K-Nearest Neighbor technique lets see how to handle the input and output for the implementation. Since we are using the MapReduce paradigm we must make sure that the input is form of a <key,value> pair. The Map routine performs the function of calculating the distance of each data point with the classes and lists it out. The Reduce routine then chooses the first 'k' Neighbors in increasing order of distances and conducts a majority vote. After which it sets the data point's label as the label of the class with the majority vote count. Now, we need to organize the input and output directories. To do this let us name the directory that holds the data vectors as vectors and the training and testing data as trainFile and testFile. Having organized our input/output directories and training and testing data ready, we can apply the k-Nearest Neighbor technique in a distributed environment by following the algorithms discussed below to design the Map and the Reduce functions for the k-Nearest Neighbor Technique.

Algorithm Mapper design for KNN

Procedure KNN MapDesign
Create list to maintain data points in the testing data-set
testList=new testList
Load file containing testing data-set
Load testFile
Update list with data points from file
testList≤ testFile
open file containing training data set
open trainFile
Load training data points one at a time and compute distance with every testing data point
Distance(trainData,testData)
Write the distance of test data points from all the training data points with their respective class labels in ascending order of distances
testFile≤testData(dist,label)
call reducer
end procedure

Algorithm Reducer design for KNN

Procedure KNN ReduceDesign
Load the value of 'k'
Load testFile
OPEN testFile
Load test data points one at a time
Read testDataPoint
Initialize counters for all class labels
SET counters to ZERO
Look through top 'k' distance for the respective test data point and increment the corresponding class label counter
For i=0 to k
$COUNTER_i$ ++
Assign the class label with the highest count for the testDataPoint in question
TestDataPoint=classLabel(COUNTRE max)
Update output file with classified test data point
outFile =outFile+testDataPoint
end procedure

Algorithm Implementing KNN Function

Procedure KNN FUNCTION
Read the value of 'k'
SET 'k'
Set paths for training and testing data directories
SET trainFile
SET testFile
Create new JOB
SET MAPPER to map class defined
SET REDUCER to reduce class define
Set paths for putput directory
SUBMIT JOB
end procedure

B. DECISION TREE C4.5

C4.5 is a standard algorithm for inducing classification rules in the form of decision tree. The default criteria of choosing splitting attributes in C4.5 is Information gain ratio. Instead of using information gain as that in ID3, information gain ratio avoids the bias of selecting attributes with many values[10].

Let C denote the number of classes, and *p(S,j)* is the proportion of instances in *S* that are assigned to j^{th} class[5]. Therefore, the entropy of attribute *S* is calculated as:

$$Entropy(S) = -\sum_{j=1}^{C} p(S,j)\log p(S,j)$$

Accordingly the information gain by a training dataset *T* is defined as:

$$Gain(S,T) = Entropy(S) - \sum_{v \in Values(T_S)} \frac{|T_{S,v}|}{|T_S|} Entropy(Sv)$$

Where $Values(T_S)$ is the set of values of *S* in*T*, T_s is the subset of *T* induced by *S*, and $T_{s,v}$ is the subset of *T* in which attribute *S* has a value of *v*.

Therefore, the information gain ratio of attribute *S* is defined as:

$$GainRatio(S,T) = \frac{Gain(S,T)}{SplitInfo(S,T)}$$

Where $SplitInfo(S,T)$ is calculated as:

$$SplitInfo(S,T) = -\sum_{v \in Values(T_S)} \frac{|T_{S,v}|}{T_S} log \frac{|T_{S,v}|}{T_S}$$

Now we study mapreduce technique for generating decision tree.We first introduce few data structures which is used in implementation:

1. ***attribute table:*** It contains the basic information of attribute a, including the row identifier of instance *row_id*, values of attribute *values(a)* and class labels of instances *c*[5].
2. ***count table:*** It computes the count of instances with specific class labels if split by attribute *a*. Two fields are included in this table: class label *c* and count *cnt*[5].
3. ***hash table:*** It stores the link information between tree nodes *node_id* and *row_id*, as well as the link between parent node *node_id* and it's branches *subnode_ids*[5].

Using the above three data structures, the process is composed of four steps: data preparation, selection, update, and tree growing.

Data Preparation:

a. Convert the traditional data into the supportable format with respect to MapReduce algorithm. So data is converted and populates attribute table and count table.
b. MAP_ATTRIBUTE procedure transforms the instance record into attribute table with attribute a_j (j=1,2,…,M) as the key and row_id, class label *c* as values.
c. REDUCE_ATTRIBUTE computes the number of instances with specific class labels if split by attribute a_j, which forms the count table. At the beginning of the process hash table set to NULL.

Algorithm **Data Preparation**

procedure MAP_ATTRIBUTE(*row_id*,(a_1,a_2,…,a_M,*c*))
    emit(a_j,(*row_id*,*c*))
end procedure
procedure REDUCE_ATTRIBUTE(a_j,(*row_id*,*c*))
    emit(a_j,(*c*,*cnt*))
end procedure

Selection:

a. The first step is to select the best attribute a_{best}.
b. REDUCE_POPULATION procedure takes the number of instances for each attribute-value pair to aggregate the total size of records for given attribute a_j.
c. MAP_COMPUTATION procedure computes the information and split information of a_j.
d. REDUCE_COMPUTATION computes the information gain ratio and the maximum value of *GainRatio*(a_j) will be selected as a splitting attribute a_{best}.

Algorithm **Selection**

procedure REDUCE_POPULATION((*a*$_j$,(*c*,*cnt*))
    emit (*a*$_j$,*all*)
end procedure
procedure MAP_COMPUTATION((*a*$_j$,(*c*,*cnt*,*all*)))
    compute *Entropy*(*a*$_j$)
    compute $Info(a_j)=\frac{cnt}{all}Entropy(a_j)$
    compute $SplitInfo(a_j)=-\frac{cnt}{all}log\frac{cnt}{all}$
    emit(*a*$_j$,(*Info*(*a*$_j$),*SplitInfo*(*a*$_j$))
end procedure
procedure REDUCE_COMPUTATION(*a*$_j$,(*Info*(*a*$_j$),*SplitInfo*(*a*$_j$)))
    emit(*a*$_j$,*GainRatio*(*a*$_j$))
end procedure

Update:

MAP_UPDATE_COUNT reads a record from attribute table with key-value equals to a_{best}, and emits the count of class labels. MAP_HASH assigns *node_id*based on a hash value of $a_{best}$to make sure that records with same values are split into the same partition.

Algorithm **Update Tables**

procedure MAP_UPDATE_COUNT((*a*$_{best}$,(*row_id*,*c*)))
    emit(*a*$_{best}$,(*c*,*cnt'*))
end procedure
procedure MAP_HASH(*a*$_{best}$,*row_id*))
    compute *node_id*=*hash*(*a*$_{best}$)
    emit(*row_id*,*node_id*)
end procedure

Tree Growing:

In previous step we generate nodes. Now we need to extend the decision tree by building linkage between nodes. We create a *node_id* for generated nodes and we compare with existing old values. If available, we group with the existing nodes, otherwise we create new sub nodes.

Algorithm **Tree Growing**

procedure MAP(*(a_{best},row_id)*)
    compute *node_id=hash(a_{best})*
    if*node_id*is same with the old value **then**
        emit(*row_id,node_id*)
    end if
    add a new subnode
    emit(*row_id,node_id,subnode_id*)
end procedure

Among the above four steps, sequence of MapReduce operations, data preparation is a onetime task and the remaining are repetitive. The terminal condition is all *node_id* become leaf nodes, and then a decision tree is built.

C. Naive Bayes

The Naïve Bayes classifier assumes that the presence of a particular feature of a class is unrelated to the presence of any other features on a given the class variable. This assumption is called class conditional independence. But it appears to work well in practice even when the independence assumption is not valid. It classifies data in two steps [4]:

Training step: Using the training samples, the method estimates the parameters of a probability distribution, assuming features are conditionally independent in the given class.

Prediction step: For an unlabeled test sample, the method computes the posterior probability of that sample belonging to each class. The method then classifies the test sample according the largest posterior probability.To demonstrate the concept of Naïve Bayes Classification, consider the knowledge of statistics. Let Y be the classification attribute and X{x1,x2,…,xk} be the vector valued array of input attributes, the classification problem simplifies to estimating the conditional probability $P(Y \mid X)$ from a set of training patterns. $P(Y \mid X)$ is the posterior probability, and $P(Y)$ is the prior probability.Suppose that there are m classes, Y1, Y2, …, Ym. Given a tuple X, the classifier will predict that X belongs to the class having the highest posterior probability[9]. The Naïve Bayes classifier predicts that tuple X belongs to the class Yi if and only if

$$P(Y_i/X) \geq P(Y_j/X)$$

The Bayes rule states that this probability can be expressed as the formulation

$$P(Y_i/X)=P(X/Y_i).P(Y_i)/P(X)$$

As $P(X)$ is constant for all classes, only $P(X/Y_i).P(Y_i)$ need be maximized. The prior probabilities are estimated by the probability of Yi in the training set. In order to reduce computation in evaluating $P(X/Y_i)$ the Naïve Bayes assumption of class conditional independence is made. So

$$P(X/Y_i)=\Pi P(x_k/Y_i) \ (k=1 \text{ to } n)$$

Research on Parallel Classification Algorithms for Large-scale Data And we easily estimate the probabilities $P(X1 \mid Y\,i)$, $P(X_2 \mid Y\,i)$,… from the training tuples. The predicted class label is the class Yi for which $P(X/Y_i).P(Y_i)$ is the maximum. The implementation of the parallel Naïve Bayes's MapReduce model is divided into training and prediction stages.

Training stage:

The distributed computing of Hadoop is divided into two phases which are called Map and Reduce. First, the InputFormat which is belonged to the Hadoop framework loads the input data into small data blocks known as data fragmentation, and the size of each data fragmentation is 5M, and the length of all of them is equal, and each split is divided into records[4]. Each map processes a single split, and the map task passes the split to the getRecordReader() method on InputFormat to gain a RecordReader for that split. The RecordReader is iterators of the records. Then the map task uses a RecordReader to generate record key-value pairs, which passes to the map function.Secondly, the map function statistics the categories and properties of the input data, including the values of categories and properties. The attributes and categories of the input records are separated by a comma, and the final attribute is the property of classification.Finally, the reduce function aggregates the number of each attribute and category value, which results in the form of (category, Index1:count1, Index2:count2, Index3:count3, … , Indexn:countn), and then output the training model. Its implementation is described as follows.

Algorithm Produce training
Input: the data set of training
Output: the statistical key-value pair of each category value and each attributes value

map(Object *key*, Text *values*, OutputCollector<Text,IntWritable> *output*, Reporter *reporter*){
1) *itr:* =new StringTokenizer(*values*.toString(),",");

2) *count:* =*itr*.countTokens();
3) FOR *each attribute* in *values* Do
4) EmitIntermediate(*w*,"1");
}
Reduce(Text *_keys*, Iterator<IntWritable> *values*, OutputCollector<Text,IntWritable> *output*,
Reporter *reporter*){
1) *sum*: =0;
2) FOR each *v* in *values* DO BEGIN
3) *sum:* +=ParseInt(*v*);
4) END
5) output.collect(*_keys*,*sum*);
}

Prediction stage:

Predicting the data record with the output of the training model. The implementation of the algorithm is stated as follows: first, use the statistical values of attribute values and category values to train the unlabeled record. In addition, use the distributed cache to improve the efficiency of the algorithm in the process of the algorithm implementation. Its implementation is described as follows.

Algorithm Produce Testing
Input: the data set of training
Output: the statistical key-value pair of each category value and each attributes value

map(Object *key*,Text *values*,OutputCollectot<Text,DoubleWritable> *output*,Reporter *reporter*){
1) *modeltype:* =new ModelType();
2) *categories*: =*modeltype*.getCategorys();
3) *data: =values*.toString().split(",");
4) *pcts:* =new double[];
5) *count:* =0;
6) *pctsResutl:* =0;
7) FOR *the value of the attribute* not NULL DO BEGIN
8) *category: =categories[0]*;
9) END
10) FOR *each value of the attribute* DO BEGIN
11) FOR *each value of the category* DO BEGIN
12) *pct*: =Counter(*attribute*,*category*)/Counter(*category*)
13) *result: =result*pct*;
14) END
15) *pcts[count++]=formula 2*;
16) END
17) *pctsResult: =the max of the pcts*
18) Emit(*category*,*pctsResult*

IV. CONCLUSION

According to our study on above classification algorithms without parallelism, we can conclude that decision tree algorithm has less error rate and it is easy to implement compared to KNN and Naive Bayes. KNN has less accuracy compared to other two algorithms and they are equal. The efficiency of KNN and Naive Bayes can be improved by increasing number of data sets and increasing number of attributes, respectively. With respect to speed, Navie Bayes algorithm is faster followed by decision tree and KNN. However traditional algorithms face difficulty with huge data. Using with MapReduce we can process large databases, improves performance, reduce the training time and prediction time and cost of communication can be reduced. Scalability is not an issue when we choose MapReduce technique as a solution for any data set size.

REFERENCES

[1] Qing He, Fuzhen Zhuang, Jincheng Li and Zhongzhi Shi, "Parallel Implementation of Classification Algorithms Based on MapReduce", Lecture Notes in Computer Science, vol. 6401, pp. 655-662, 2010.

[2] Chia-WeiLeea, Kuang-YuHsieha, Sun-YuanHsieha,b,∗ , Hung-ChangHsiaoa , " A Dynamic Data Placement Strategy for Hadoop in Heterogeneous Environments" Big Data Research (2014) 14–22 ELSEVIER.

[3] Prajesh P Anchalia , Koushik Roy, "The k-Nearest Neighbor Algorithm Using MapReduce Paradigm**"** 2014 Fifth International Conference on Intelligent Systems, Modelling and Simulation.

[4] Lijuan Zhou, Hui Wang, Wenbo Wang **Research on Parallel Classification Algorithms for Large-scale Data,** Journal of Convergence Information Technology(JCIT), Volume 7, Number 21, Nov 2012, doi : 10.4156/jcit.vol7.issue21.41

[5] Wei Dai and Wei Ji, " A MapReduce Implementation of C4.5 Decision Tree Algorithm, International Journal of Database Theory and Application Vol. 7, No.1 (2014),pp.49-60

[6] Seyed Reza Pakize, Abolfazl Gandomi, Comparative Study of Classification Algorithms Based on MapReduce Model, International Journal of Innovative Research in Advanced Engineering (IJIRAE) Volume 1 Issue 7 (August 2014)

[7] DR. A. N. Nandakumar[1], Nandita Yambem[2] A Survey on Data Mining Algorithms on Apache Hadoop Platform, International Journal of Emerging Technology and Advanced Engineering, Volume 4, Issue 1, January 2014)

[8] Apache Hadoop. http://hadoop.apache.org/

[9] Jiawei Han and Micheline Kamber, "Data Mining Concepts and Techniques" Second Edition

[10] J.R. Quinlan," C 4.5: Programs for Machine Learning", Morgan Kaufmann

