International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

One Class Clustering Tree for Implementing Many to Many Data Linkage

Ravi R[1], Michael G[2]

[1]PG Scholar, Department of Computer Science and Engineering, Bharath University, Selaiyur, Chennai - 600 073, India

[2]Assistant Professor, Department of Computer Science and Engineering, Bharath University, Selaiyur, Chennai - 600 073, India

Abstract: ***One-to-many and many-to-many data linkage are important in data mining. In earlier works data linkage is performed among entities of the same type. To link between matching entities of different types in larger datasets a new one-to-many and many to many data linkage method with MapReduce is proposed that links between entities of same and different natures. The proposed method is based on a one-class clustering tree (OCCT) that characterizes the entities that should be linked together. With development of the information technology, the scale of data is increasing quickly. The massive data poses a great challenge for data processing and classification. In order to classify the data, there were several algorithm proposed to efficiently cluster the data. This project deals with scalable random forest algorithm for classifying the advertisement benchmark datasets.***

Keywords: Data mining, Hadoop, MapReduce, Clustering Tree

1. Introduction

Data linkage is one of the important task in data mining. It is of two kinds: one-to –one, one-to-many and many to many. One-to-one data linkage associates one entity with a matching entity in another data set. One-to-many data linkage associates an entity with a group of matching entities from another data set. Many-to-many data linkage associates group of entities from one data set with group of matching entities from another data set. In this paper many-to-many data linkage is implemented with MapReduce framework.

The massive data poses a great challenge for data processing and classification. In order to classify the data, there were several algorithm proposed to efficiently cluster the data. One among that is the random forest algorithm, which is used for the feature subset selection. The feature selection involves identifying a subset of the most useful features that produces compatible results as the original entire set of features. It is achieved by classifying the given data. The efficiency is calculated based on the time required to find a subset of features, the effectiveness is related to the quality of the subset of features. The existing system deals with fast clustering based feature selection algorithm, which is proven to be powerful, but when the size of the dataset increases rapidly, the current algorithm is found to be less efficient as the clustering of datasets takes quiet more number of time.

Hence the new method of implementation is proposed in this project to efficiently cluster the data and persist on the back-end database accordingly to reduce the time. It is achieved by scalable random forest algorithm. The Scalable random forest is implemented using Map Reduce Programming (An implementation of Big Data) to efficiently cluster the data.

It works on two phases, the first step deals with the gathering the datasets and persisting on the datastore and the second step deals with the clustering and classification of data. This process is completely implemented using Google App Engine's hadoop platform, which is a widely used open-source implementation of Google's distributed file system using MapReduce framework for scalable distributed computing or cloud computing. Hadoop MapReduce is a software framework for distributed processing of large data sets. It is a sub-project of the Apache Hadoop project. The framework takes care of scheduling tasks, monitoring them and re-executing failed tasks. The primary objective of MapReduce is to split the input data set into independent chunks that are processed in a completely parallel manner.

The Hadoop MapReduce framework sorts the outputs of the maps, which are then input to the reduce tasks. Both the input and the output of the job are stored in a file system. The proposed method with MapReduce links between the entities using a One-Class Clustering Tree. A clustering tree is a tree contains a cluster in each of the leaves instead of a single classification. Each cluster is generalized by a set of rules that is stored in the appropriate leaf. One-class clustering tree is implemented with the help of MapReduce in parallel. Clustering tree assigns each linkage task to mapper class. Master node in the mapper assigns the work to slaves. Slaves in turn completes the work and returns the result to master node. The reducer class combines the intermediate results from mapper class to provide a complete clustering tree. As clustering tree is implemented in a parallel and distributed environment it reduces the execution time. The objectives of the paper is threefold:

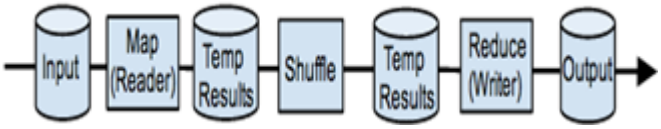
1) To implement one-to-many and many–to-many data linkage
2) To make data linkage for larger datasets more efficient
3) To execute data linkage in a parallel and distributed environment using MapReduce.

2. Related Work

2.1 Record linkage using Map Reduce

A MapReduce job has three stages: map, shuffle, and reduce. Each stage in the sequence must complete before the next one can run. Intermediate data is stored temporarily between the stages.

The data flow for a MapReduce job looks like this:



2.2 Map

The MapReduce library includes a Mapper class that performs the map stage. The map stage uses an input reader that delivers data one record at a time. The library also contains a collection of Input classes that implement readers for common types of data. You can also create your own reader, if needed.

The map stage uses a map() function that you must implement. When the map stage runs, it repeatedly calls the reader to get one input record at a time and applies the map() function to the record.

The implementation of the map() function depends on the kind of job you are running. When used in a Map job, the map () function emits output values. When used in a map reduce job, the map() function emits key-value pairs for the shuffle stage. When emitting pairs for a MapReduce job, the keys do not have to be unique. The same key can appear in many pairs.

2.3 Shuffle

The shuffle stage first groups all the pairs with the same key together and then outputs a single list of values for each key: If the same key-value pair occurs more than once, the associated value will appear multiple times in the shuffle output for that key. Also note that the list of values is not sorted. The shuffle stage uses a Google Cloud Storage bucket, either the default bucket or one that you can specify in your setup code.

2.4 Reduce

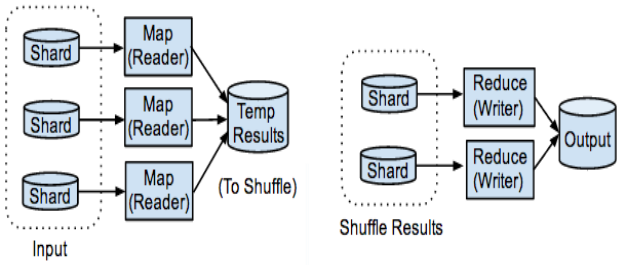
The MapReduce library includes a Reducer class that performs the reduce stage. The reduce stage uses a reduce() function that you must implement. When this stage executes, the reduce() function is called for each unique key in the shuffled intermediate data set. The reduce function takes a key and the list of values associated with that key and emits a new value based on the input.

The reduce output is passed to the output writer. The MapReduce library includes a collection of Output classes that implement writers for common types of output targets.

2.5 Sharding: Parallel Processing

Sharding divides the input of a stage into multiple data sets (*shards*) that are processed in parallel. This can significantly improve the time it takes to run a stage. When running a MapReduce job, all the shards in a stage must finish before the next stage can run. When a map stage runs, each shard is handled by a separate instance of the Mapper class, with its own input reader. Similarly, for a reduce stage, each shard is handled by a separate instance of the Reducer class with its own output writer. The shuffle stage also shards its input, but without using any user-specified classes.

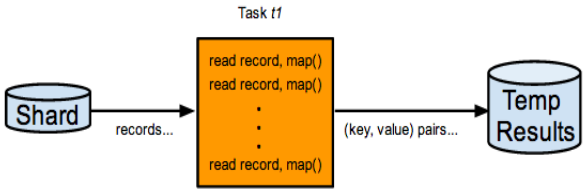
The number of shards used in each stage can be different. The implementation of the input and output classes determines the number of map and reduce shards respectively. The diagram below shows the map stage handling its input in three shards, and the reduce stage using two shards.



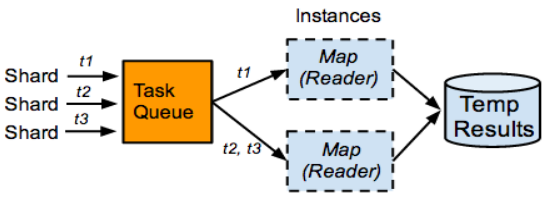
2.6 Slicing: Distributed Scheduling and Fault Tolerance

The data in a shard is processed sequentially. The job executes a consecutive series of tasks using an App Engine task queue, one task at a time per shard. When a task runs, it reads data from the associated shard and calls the appropriate function (map, shuffle, or reduce) as many times as possible in a configurable time interval.

The data processed in a task is called a *slice*. The amount of data consumed in a slice can vary, depending on how quickly the function processes its input. When a slice is completed, another task is enqueued for the next slice on the shard. The process repeats until all data in the shard has been processed. The diagram below shows a task in the map stage consuming a slice of a shard with repeated read/map calls:



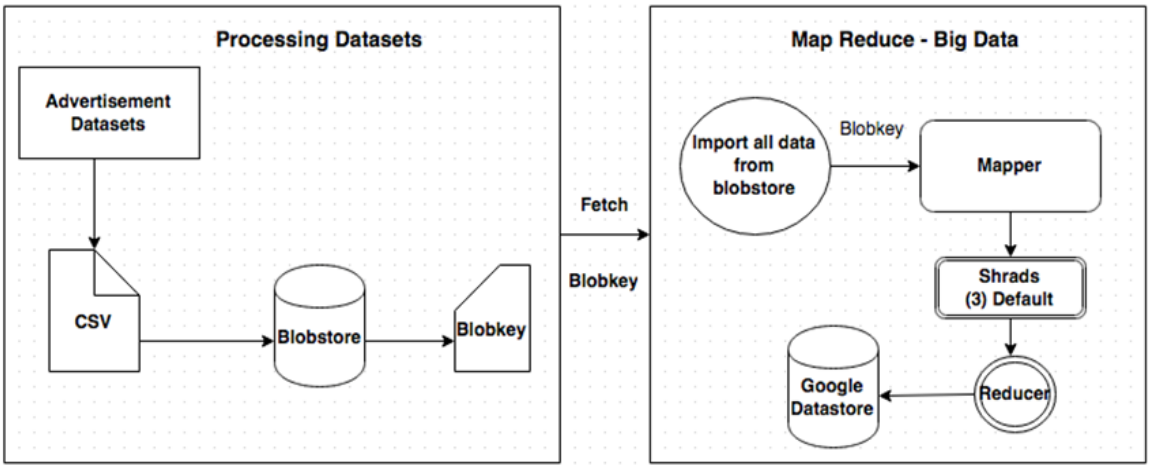
The tasks for all shards are placed in a single task queue. App Engine dynamically determines how many instances of a module to spin up in order to handle the task load. The number of instances may change while a stage is running. The diagram below shows a moment in time during the map stage when only two instances of the module running the Map are handling the three current tasks (t1, t2, t3) associated with the shards.

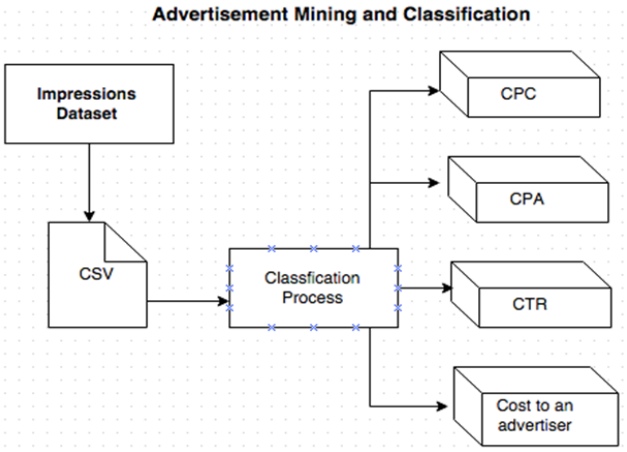


The use of task queues, along with dynamic instance scaling, helps to distribute the workload efficiently and transparently. Dividing execution into slices also offers a level of fault

tolerance. Without slicing, if an error occurs while processing a shard, that entire shard would need to be re-run. With execution broken into slices, it is possible to detect the failure of a slice and attempt to re-run the slice a number of times before declaring a complete failure of the shard, possibly starting the shard again, or failing the entire job.

3. Proposed Architecture





The advertise dataset Is taken in to consideration in order to effectively apply the mining and clustering technique to classify the advertisements to different categories based on several factors like click rate, cost of advertising etc.

The main work towards the step of Algorithm is: Firstly, the samples from the original dataset is selected. Then, the samples will be the training set for growing K trees accordingly to achieve the K classification results . Before that, the classification is relied on a mathematical calculation which is depending on the other parameters of the dataset.

The final classification of datasets is done on as CPC, CPA, CTR, Cost of Advertisement.

4. Modules

4.1. Login – Authentication

Authentication is used to make the application much secured and allow only the authorized person to use the application. Google Datastores is used to store the user information, and the same is applied during the authentication process.

4.2. Split data into Subset

This process is technically done by using map reduce algorithm. The Mapper is used for splitting as the data set is huge in quantity. For Splitting, Map Reduce uses the concept called InputSplit. InputSplit represents the data to be processed by an Individual Mapper.

4.3. Data Parallel Processing (Mapper)

In this module, we implement or configure the map reduce job for dividing the input data in to different clusters and share it among multiple nodes and process in parallel.

4.4. Combine Intermediate Result (Reducer)

In this module, we combine the intermediate results from all individual map reduce job, that we allocated on the previous module. After combing, the negotiation of K value for feature subset selection is processed.

4.5. Collect AD Impressions Datasets

- Cost-per click is important because it is the number that is going to determine the financial success of your paid search advertising campaign.
- CPA advertising tracks the person clicking on an ad and determines if that person then also creates a desired transaction on the destination site.

4.6. Classification process (using Map Reduce)

- The advent of revolution in technology and internet has caused increase in marketing platform for advertising companies.
- The classification is done in order to categorize the advertisements based on different factors that includes, CPA, CPC. Etc.
- There are various models for determining the cost of advertising. They are Cost per Impressions (CPM), Cost per Click (CPC) and Cost per Action (CPA).

- CPM denotes Cost per 1000 Impressions (frequency of the ad display). If the advertisement is shown 2000 times the cost will be equal to 2 CPM price.

5. Conclusion

OCCT, a one-class decision tree approach for performing one-to-many and many-to-many data linkage using MapReduce is presented in the paper. The proposed method is based on a one-class decision tree model that encapsulates the knowledge of which records should be linked to each other. Implementation using MapReduce will reduce the execution time of many-to-many data linkage. It enhances parallelism as linkage is executed in a distributed environment. The proposed method will be very efficient for large datasets.

References

[1] Ma'ayan Dror, Asaf Shabtai, Lior Rokach, Yuval Elovici "OCCT: A One-Class Clustering Tree for Implementing One-to-Many Data Linkage", IEEE transactions on Knowledge and Data engineering, Vol. 26, No. 3, March 2014
[2] C. Li, Y. Zhang, and X. Li, "OcVFDT: One-Class Very Fast Decision Tree for One-Class Classification of Data Streams," Proc. Third Int'l Workshop Knowledge Discovery from Sensor Data, pp. 79- 86, 2009.
[3] Anne-Laure Boulesteix, Silke Janitza J ochen Kruppa, Inke R. K¨onig "Overview of Random Forest Methodology and Practical Guidance with Emphasis on Computational Biology and Bioinformatics " pre-review version of a manuscript accepted for publication in WIREs Data Mining & Knowledge Discovery , July 25th 2012
[4] Chen, W Y; et al. (2011). "Parallel Spectral Clustering in Distributed Systems". *IEEE Trans. Pattern Anal. Mach. Intell.*, 568-586.
[5] Jiawei Hanl, Yanheng Liul, Xin Sunl "A Scalable Random Forest Algorithm Based on MapReduce" presented at the IEEE Summer Power Meeting , 2013 IEEE
[6] Aditya B. Patel, Manashvi Birla, Ushma Nair "Addressing Big Data Problem Using Hadoop and Map Reduce" presented at NIRMA university international conference on engineering NUiCONE-2012, 06-08December, 2012.
[7] Jyoti Nandimath , Ankur Patil, Ekata Banerjee, Pratima Kakade "Big Data Analysis Using Apache Hadoop" presented at IEEE IRI 2013, August 14-16, 2013, San Francisco, California, USA
[8] J. Dean and S. Ghemawat, "MapReduce: simplified data processingon large clusters," Commun. ACM, vol. 51, no. I, pp. 107-113, 2008.
[9] Apache Software Foundation. Official apache hadoop website, http://hadoop.apache.org/, Aug, 2012.
[10] O'Reilly; Third edition, Tom White. Hadoop: A definitive guide.2012