Credit Risk Evaluation System on Big Data using Neurorule

Rajiv Ranajn[1] Dr. Srikanth Prabhu[2]

Abstract— The phrase credit analysis is used to depict any process for evaluating the credit feature of the counterparty. Before offering credit, a bank or other loaner acquires information about the party asking a loan. In the case of a bank offering credit, this might involve the party's annual earnings, existing loans, whether they lease or own a home, etc. Based upon the credit score, the loaning body will decide whether or not to offer credit. The difficulty of credit-risk evaluation is a very questioning and important monetary analysis problem. In recent times, researchers have found that neural networks carry out very well for this intricate and unorganized problem when compared to more established statistical approaches. A major fault associated with the use of neural networks for conclusion making is their lack of clarification capability. While they can achieve a high anticipating accuracy rate, the reasoning behind how they reach their decisions is not easily available. In this paper, the results from analyzing real life credit-risk evaluation data set using neural network rule extraction methods are presented. The dataset chosen is Statlog German credit data set which is readily available for public use at the Statlog repository. Hadoop MapReduce has been used and is a software framework for easily writing applications which handle huge amounts of data in terms of multiterabyte datasets in parallel on large bundles i.e. many thousands of nodes of asset hardware in a dependable, fault-tolerant manner have been used. It has been carried out using Eclipse. This is the reason MapReduce programming is used to work with big data. In addition, Partitioning Around Medoids Clustering algorithm (PAM algorithm) is implemented using map-reduce programming in what way, the medoids are the standards of the neural network to examine credit risk which cluster the customers into the clusters pertaining to the different neurorules. Thus, this paper is aimed to show the derivation of rules from educated neural networks and representing these rules and using it to train large data sets in order to support a reasonable and valuable way for building credit-risk evaluation expert systems.

Key words: Neurorule, Credit Risk Evaluation System

I. INTRODUCTION

Big Data can be defined as high quantity, velocity, and array of information resources that demand cost-effective, creative forms of information processing for improved insight and decision making. Since a long time, financial institutions have accumulated heaps of information on their customers and services about the transactions, undertakings and credit card balances that are their lifeblood. And they change much of this data into periodical reports for use by their customers, accountants, and auditors. Credit risk refers to the risk that a debtor will default on any type of debt by failing to make necessary payments. In recent times, there has been a burst of developments in the field of credit risk modeling by a number of monetary organizations. The difficulty of credit-risk judgment is a very disputing and important financial analysis problem. For loans to individuals or small businesses, credit quality is typically evaluated through a process of credit scoring. A standard formula is applied to the data, which gives a number, which is called a credit score.

Now having derived the credit score, the loaning institution will settle an issue whether or not to offer credit. Neural networks have been shown to be very strong pattern identification techniques for classification in a variety of domains. Their ubiquitous estimation property combined with their parallel processing capacity is one of the major drivers behind their accomplishment. Unluckily, an often mentioned disadvantage associated with using neural networks is their cloudiness. Neural networks are commonly stated as black box methods because they generate intricate mathematical examples which relate the outputs to the inputs using a set of biases and non-linear activation functions that are hard for humans to understand. In many application fields, it would be very beneficial to have a set of rules that clarifies why a specific classification is made. The field of our current preliminary study is financial credit-risk evaluation. The significance of having a good model that helps the credit expert in deciding and ruling has been stressed by many researchers. Recent developments in algorithms that derive rules from educated neural networks allow us to create descriptive categorization standards that are as accurate as the networks. The purpose of our research is to examine if these neural network rule extraction methods can generate significant and accurate rule sets for the credit-risk evaluation problem.

II. RELATED TERMS AND DEFINITIONS

In this section, we discuss terms and definitions associated with this paper.

A. Neurorule

Neurorule can be defined more accurately as a decompositional algorithm that mostly derives propositional rules from trained 3 layered feedforward neural networks. It consists of the following steps:

1) To train a neural network to meet the pre-defined accuracy demand.
2) To remove the repeated connections in the network by shortening while maintaining its exactness.
3) To approximate the hidden unit activation values of the shortened network by clustering.
4) To derive rules that shows the network's outputs in terms of the discretized hidden unit activation values.
5) To generate rules that show the discretized hidden unit activation values in terms of the network inputs.
6) To merge the two sets of rules generated in steps 4 and 5 to reach to a set of rules that correlates the inputs and outputs of the network.

B. Datasets

Each and every neural network rule derivation techniques was practiced on real life credit-risk evaluation data sets. The data set used for this research paper is Statlog German credit data set which can be availed easily as it is publicly available at the Statlog repository. The inputs not only include sociodemographic variables but loan specific

information such as, what was the purpose of the loan and how long were its duration.

Table 2.2.1 displays the characteristics of the data set. Table 2.2.2 presents the attributes for the German credit data set.

	Number of Inputs	Data Set Size	Training Set Size	Test Set Size
German credit	20	1000	666	334

Table 1: Characteristics of Data Set

Attribute	Type	Values
Checking account	qualitative	1: < 0 DM; 2: ≥ 0 and < 200 DM; 3: ≥ 200 DM/salary assignments for at least one year; 4: no checking account
Duration (in months)	numerical	
Credit history	qualitative	0: no credits taken/all credits paid back duly; 1: all credits at this bank paid back duly; 2: existing credits paid back duly till now; 3: delay in paying off in the past; 4: critical account/ other credits existing (not at this bank)
Purpose	qualitative	0: car (new); 1: car (old); 2: furniture/equipment; 3: radio/television; 4: domestic appliances; 5: repairs; 6: education; 7: vacation; 8: retraining; 9: business; 10: others
Credit amount	numerical	
Savings account	qualitative	1: < 100 DM; 2: ≥100 and < 500 DM; 3: ≥500 and < 1000 DM; 4: ≥1000 DM; 5: unknown/no savings account
Present employment since	qualitative	1: unemployed; 2: <1 year; 3: ≥1 and < 4 years; 4: ≥4 and < 7 years; 5: ≥7 years
Installment rate in percentage of disposable income	numerical	
Personal status and sex	qualitative	1: male, divorced/separated; 2: female, divorced/separated/married; 3: male, single; 4: male, married/widowed; 5: female, single

Table 2: Attribute for German Credit

We should be aware of that the duration attribute was specified as mentioned below:

Duration less than or equal to 15 months then 1, else 2.

C. Extracted Rule Sets and Decision Tables

Decision tables (DTs) are a tabular chart or image used to portray and evaluate decision circumstances, where the state of a number of conditions together regulates the execution of a set of actions. A Decision Table is made up of four quadrants, detached by double-lines, pairing vertically and horizontally. The horizontal line isolates the table into a condition part as above and an action part as below. The vertical line isolates subjects as left from entries as right. The condition subjects are the benchmarks that are pertinent to the decision-making process. They show the attributes about which data is needed to labialize a given case. The action subjects show the possible conclusions of the decision-making process (the classes of the classification complication). Each and every condition entry describes an important subset of values called a state, for a given condition subject called attribute, or can be left empty if its value is not important with relation to that particular column. Finally, every action entry carries a value assigned to the corresponding action subject (class).All the columns in the entry part of the DT thus constitute a classification rule, showing what actions apply to a unique possibility of condition states. If the case be that each column only contains simple states one without contraction or totally irrelevant, the table is called an expanded DT, and else the table is called a contracted DT.

Figure 1 shows the DT generated from the rules, for the German credit data set derived by Neurorule. It is important to notice that the fully expanded DT contains a count of 1800 columns, which can be calculated easily by multiplying attribute values (2 * 5 * 3 * 5 * 4 * 3). This can be further simplified to nine columns by methods of table contraction, which is a relatively small number.

condition subjects	condition entries
action subjects	action entries

Fig. 1: DT Quadrants

German Credit									
1. Checking account	< 3							≥ 3	
2. Savings account	< 3						≥ 3	–	
3. Other parties	< 3					3	–	–	
4. Credit history	< 4			= 4			–	–	–
5. Duration	1		2	1	2		–	–	–
6. Other installment plans	< 2	≥ 2	–	–	< 2	≥ 2	–	–	–
1. Applicant = Good	–	x	–	x	–	x	x	x	x
2. Applicant = Bad	x	–	x	–	x	–	–	–	–
	1	2	3	4	5	6	7	8	9

Fig. 2: Decision table for Neurorule on German Credit

D. Hadoop HDFS & MapReduce

1) HDFS (Hadoop Distributed File System) Distributed filesystem.

2) MapReduce Distributed data processing model.

3) Hadoop provides a dependable shared repository and analysis system for large-scale data processing.

- Storage is provided by HDFS.
- The analysis is provided by MapReduce.

E. HDFS

1. The HDFS operates on top of an existing filesystem.
2. The files are stored as blocks whose default size is 64MB.
3. It provides reliability through replication as each block is replicated across several Data Nodes. The NameNode stores metadata and manages access.

F. Mapreduce

MapReduce can be described as a combination of software framework for handling of (large) data sets in a dispersed fashion over many numbers of machines. The main logic behind MapReduce is mapping the dataset in to a compilation of <key, value> pairs, and then reducing overall pairs using the same key.

1) Almost every data possibly can be mapped into <key, value> pairs.
2) b) The keys and values may be of any data type: strings, integers and always <key, value> they should form a pair.Figure 1.4 shows the overall flow of a MapReduce operation in our implementation. The user program calling the MapReduce function causes the following sequence of events to happen.

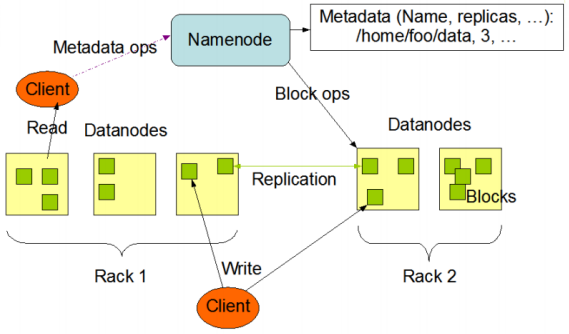

Fig. 3: HDFS Architecture Diagram

1) The MapReduce library in the user program begins its operation by dividing the input values into M pieces of typically 16-64MB per piece (easily changeable by the user via an optional parameter). It then creates and stores up many copies of the program on a large number of machines.
2) One of the replicas of the program the master is exclusive and special. The rest copies are workers that are assigned work by the master. So we have M map tasks and R reduces tasks and these are to be assigned. The master picks workers not working currently and assigns each one either a map task or a reduce task.
3) A worker who is appointed a map task scans the details of the corresponding input split. It determines key/value pairs between the input data and passes each pair to the user-designed map function. The intermediary key/value pairs produced by the map function are stored in memory.
4) Regularly, the stored, buffered pairs are written to local disk, divided into R regions by the partitioning function. The place of storage of these buffered pairs on the local disk is known back to the master who is controlling and these areas to the curtail workers.
5) A reduce worker is whenever observed by the master about these places; it applies remote procedure calls to fetch the stored data from the local disks of the map workers. When a reduce worker has read all intermediary data for its division, it arranges it in an order, by the intermediary keys so that each and every occurrences of the same key is grouped together. The sorting is necessary because typically many different keys map to the same reduce task. If the amount of intermediary data is too huge to fit in memory, an outside sort is used.
6) The reduce worker repeats over the sorted intermediary data and for each unique intermediary key found, it passes as a parameter the key and the equivalent set of intermediary values, as a pair to the users reduce function. The output of the reduce function is attached to the last output file for this reduce partition.
7) When all map tasks and reduce tasks have finished, the master wakes up the user program. Now, the MapReduce call in the user program goes back to the user code. After successful implementation, the consequence of the execution of the mapreduce execution is ready for use in the R output files (one for every reduce task, with file names given by the user). Typically, users do not need to fuse these R output files into one file; they often give as parameters these files as input to another MapReduce call or use them from a dissimilar scattered application that is able to accord with input that is divided into many files.

G. Map Reduce Programming In Java

The MapReduce framework functions particularly on (value, key) pairs, that are the foundation views the data to the job as a set of (value, key) pairs and generates a set of (value, key) pairs as the output of the job, possibly of different kinds. The value and key classes have to be serializable by the foundation and thus required to start the Writable interface. Furthermore, the key classes have to put in action the WritableComparable interface to assist the progress of sorting by the foundation.

H. Mapper

The main function of Mapper is to map input key/value pairs to a set of intermediary key/value pairs. Maps are the single tasks that alter input records into intermediary records. The altered intermediary records do not necessarily have to be of the same type as the input ones. As it might be that a given input pair may map to zero or sometimes many output pairs. The Hadoop MapReduce framework generates one map task for each InputSplit created by the InputFormat for the job. So in total, Mapper operations are passed the JobConf for the job via the JobConfigurable.configure (JobConf) method and override it to load themselves. The framework then asks for the map (WritableComparable, Writable, OutputCollector, Reporter) with these parameters, for each key/value pair in the InputSplit for that task. Output pairs do not necessarily have to be of the same types as input pairs. As it might be that a given input pair may map to zero or sometimes many output pairs. Output pairs are congregated with calls to OutputCollector.collect (Writable Comparable, Writable), with these parameters. Applications can use the Reporter to report development, set application-level status messages and renew Counters, or just show that they are alive. Every intermediary value correlated with a given output key are subsequently arranged by the framework, and passed to the Reducer(s) to find out the final output. Users can modify the grouping by defining a Comparator via JobConf.set OutputKey Comparator Class (Class). The Mapper outputs are arranged in an order and then separated per Reducer. The total count of partitions is same in number as the count of reduce tasks for the job. The number of maps is usually directed by the full size of the inputs, that is, the total number of blocks of the input files. The intermediary, sorted outputs are stored always in a simple (key-length, key, value-length, value) format. Applications can make changes to if, and how, the intermediary outputs are to be constricted and the CompressionCodec to be implemented via the JobConf.

1) The Map() Function:

The map() function marks the beginning of the processing. It forms the key-value pair on the collected dataset and then divides it on distributed node of commodity servers. The map () function is aware of where the data dwells on distributed file system and runs or makes the execution happen there. The gain with this is that the processing of data is done on the location where the data dwells. In normal programming paradigm the data is retrieved to the place where the applications running and then the processing begin and it happens to be a high priced operation. On the contrary in the map-reduce programming model, the task is carried where the data dwells and hence it is very fast, competent and adequate.

I. Reducer

Reducer diminishes a set of intermediary values which share a key to a smaller set of values. The number of reduces for the job is set by the user via Job-Conf.setNumReduceTasks(int). Overall, Reducer implementations are passed the JobConf for the job via the JobCon_gurable.con_gure(JobConf) method and can override it to initialize themselves. The framework then

gives a call to the function reduce(WritableComparable, Iterator, OutputCollector, Reporter), for each <key, (list of values)>pair in the grouped inputs. Input to the Reducer is the sorted output of the mappers. In this phase the framework retrieves the pertinent partition of the output of all the mappers, via HTTP.

1) The Reduce() Function:

The following stage in the MRP(Map Reduce Programming model is the Reduce() function, which is the second phase. So once mapping finishes, the reduce() function works on the intermediate data set by fetching them from disk/memory or any other location. Amassing the data from all of these processes is the last result that happens because of reduce() function. The expandability and extensibility are the main advantages of MapReduce programming model.

III. DESIGN

A. Pam Clustering Algorithm

The PAM clustering algorithm is used to partition data set into k number of clusters based on k number of medoids which represent each cluster.

B. Neurorule

The total number of good and bad customers in the training data set of 1000 data rows which is already known from the previous method of using PAM Clustering Algorithm is used to find the probability of occurrence of each of the attributes used in neurorule.

C. Determining Credit Risk (Good/Bad) for a Customer

The attributes of German credit data set are used in neurorule to categorize a customer as a good/bad customer based on the credit risk of the customer.

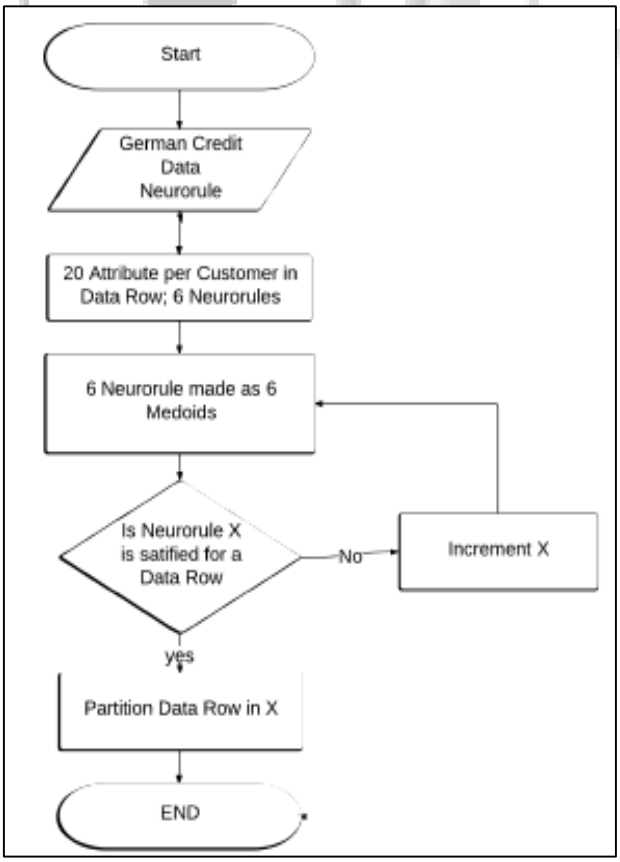

Fig. 4: Applying PAM Clustering Algorithm to Data Set

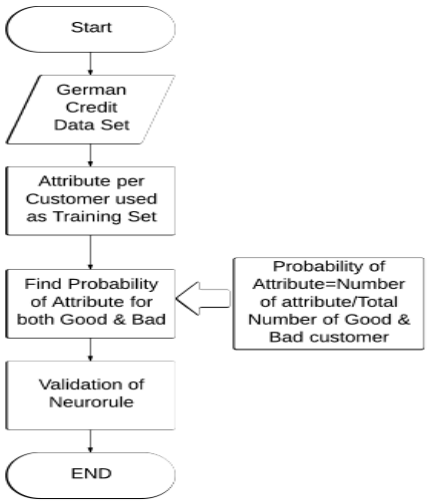

Fig. 5: Validating Neurorule

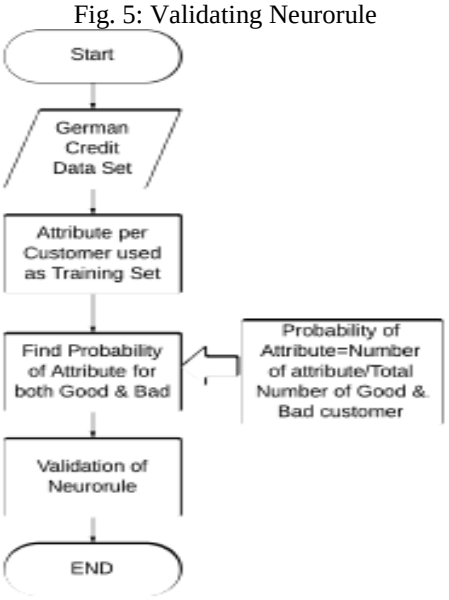

Fig. 6: Determining Credit Risk (Good/Bad) for a Customer

IV. METHODOLOGY

A. Pam Clustering Algorithm

The (Partitioning around medoid) short form as PAM a clustering algorithm is established primarily on the investigation and look out for k representative objects or medoids among the observations of the considered dataset. These observations should hence, exhibit the formation of the data. So after finding a set of k medoids, k clusters are created by assigning every observation to the immediate medoid. The goal here is to find k representative objects which will or possibly minimize the sum of the differences of the examinations to their closest representative object. So obviously, when medoids are not stated, the algorithm first looks for a good initial set of medoids (this is called the build phase) and when it doesn't happen then it finds a local minimum for the objective function, that is, trying to find a solution such that there is no single shift of an inspection with a medoid that will decrease the objective (this is called the swap phase).

- The PAM algorithm works over two kinds of input as suggested above:
- The first is the matrix showing every item and the values of its variables.

- The second is to work with the dissimilarity matrix directly.

In this project, the first kind of input i.e. representation of every entity and the values of its variables is used. The most common function of k-medoid clustering is the Partitioning Around Medoids (PAM) algorithm and be stated as below:

1) Initialize:

start with randomly choosing (without replacement) k of the n data points as the medoids

2) Association of each data points to the closest medoid.

("closest" here is defined using any authentic distance metric, and these can be any of these like Euclidean distance, Manhattan distance or Minkowski distance)

3) For each medoid m

a) For each non-medoid data point o

Interchange m and o and find out the total cost of the configuration

4) Here the configuration with the lowest cost has to be selected.

5) Repeat steps 2 to 4 until there is no change in the medoid.

- In a general analysis the algorithm follows this path:
- Build phase

1) k entities are chosen to become the medoids, or in case these entities were provided are used as the medoids;
2) The dissimilarity matrix is calculated if it was not informed;
3) Every entity is assigned to its closest medoid;

- Swap phase

4) For each cluster try to find out if any of the entities of the cluster goes below the average dissimilarity coefficient if it does select the entity that causes the most lowering of this coefficient as the medoid for this cluster;
5) Even if at least there has been a change in medoid from one cluster go to (3), else end the algorithm.

- The medoid of each cluster as the points which are extracted as the Neurorule in the german credit data set.
- For instance, each point symbolizes the data row with specific characteristics i.e.
- Cluster 1: data rows with Checking account >= 3
- Cluster 2: if data rows with (Duration = 1) and (Other installment plans>=2)
- Cluster 3: if data rows with (Other installment plans>=2) and (Credit history = 4)
- Cluster 4: if data rows with (Duration = 1) and (Credit history= 4)
- Cluster 5: if data rows with (Savings account>=3)
- Cluster 6: if data rows with (Other parties = 3)
- Each of the data rows lying in any of the above clusters denotes a good customer.
- The rest of the leftover data rows are taken as outliers clustered to form separate cluster 7 which denotes a bad customer.
- In this manner, total numbers of good and bad customers are calculated by partitioning the data set into seven clusters.

B. Validation of Neurorules

Neurorule is a type of decompositional algorithm that primarily extracts propositional rules from trained 3 layered feedforward neural networks.

C. Propositional Neurorules

The 7 propositional rules extracted by neurorule are as follows:

1) Rule 1: Checking account >= A13
2) Rule 2: (Duration = 1) and (Other installment plans>=A142
3) Rule 3: (Other installment plans>=A142) and (Credit history =A34)
4) Rule 4: (Duration = 1) and (Credit history = A34)
5) Rule 5: (Savings account>=A63)
6) Rule 6: (Other parties = A103)
7) Rule 7: If neither of rules 1 to 6 apply

The German credit data set is used as training set. The 21st attribute of 1000 data row set is analyzed which specifies whether each customer is liable to be called a good or bad credit risk. Accordingly each data row i.e. each customer attribute information is stored at two at different places, one file for each good and bad customer. The occurrence of each attribute which is vitally used in neurorule to categorize a customer as good/bad is counted and summed up for good and bad customers. The total number of good and bad customers in the training data set of 1000 data rows which is already known from the previous method of using PAM Clustering Algorithm is used to find the probability of occurrence of each of the attributes used in neurorule.

Probability of attribute(x), P(x) = Number of occurrence of attribute(x)/Total number of good customers

Similarly, for bad customer data set,

For good customer,

Probability of attribute(x), P(x) = Number of occurrence of an attribute(x)/Total number of bad customers

The 7 propositional rules extracted by neurorule is validated using the probability of occurrence of attributes used in the seven rules by comparing the probability of occurrence for both good and bad. An attribute whose probability of occurrence is higher for good customers than bad customers is viable to be associated with the rule for partitioning the data set into the good customer cluster. Similarly, an attribute whose probability of occurrence is higher for bad customers than good customers is viable to be associated with the rule for partitioning the data set into the bad customer cluster. After applying the procedure, it can be concluded that Neurorule is able to extract very concise rule sets with high classification accuracy on the test set for given data sets.

D. Evaluation of Credit Risk (Good/Bad) for a Customer

Following are the neurorules which evaluate the credit risk involved in each customer.

if (Checking account ≥ 3) **then**
Applicant = good

if (Duration = 1) and (Other installment plans≥2) **then**
Applicant = good

if (Other installment plans≥2) and (Credit history = 4) **then**
Applicant = good

if (Duration = 1) and (Credit history = 4) **then**
Applicant = good

if (Savings accoun t≥3) **then**
Applicant = good

if (Other parties = 3) **then**
Applicant = good

Default class: Applicant = bad

Fig. 7: Nurorules

E. Map Reduce Programming in Java

MapReduce programming is done in java specifically designed to process a huge amount of data in parallel. The scalability together with extensibility are the main advantages of MapReduce programming model. The map function in the map-reduce java program scans each data row in the data set. The attributes on which neurorule are based are of great importance in determining if the data row which belongs to a particular customer x belongs to which cluster. It is these seven neurorule which partition the data set into different clusters. Each data row of a customer belongs to one of the seven clusters. The ones which belong to cluster 1 to 6 are categorized as good credit risks or good customers. The ones which belong to cluster 7 are categorized as bad credit risks or bad customers. The data row i.e. the attributes for a customer entered by the user is stored in a file which is given as input to the map-reduce program through the driver class which calls the map and reduce functions. It also includes the output directory of the file generated after completion of the job. The driver class contains a map function which reads the record with the default record reader. The default record contains the key as the cumulated character count for that line and the value as the whole line as a Text till the newline character. Using the split() method of the String class, we split using the delimiter (" " in our case) to get the array of strings. The required entries in the array are used to partition the set based on the seven neurorules and finally the cluster to which it belongs is the output key of the mapper.

V. IMPLEMENTATION

1) Firstly, the German data credit set is taken as an input and using map reduce programming in java, the number of data rows in each of the seven clusters is displayed in an output file.
2) Next, the German data credit set is taken as an input and the probability of occurrence of each attribute is found which are used in portioning of the cluster using neurorule.
3) In the next step, the console asks for entering the attributes of the customer. The user can enter data for any number of customers.
4) On receiving the data attributes of each customer, it is stored in a file and is processed using map reduce() programming and functionalities in java.
5) The output that is obtained here is a file which contains information as to serially which customer belongs to which cluster of the neurorule and the credit risk is evaluated for the customer whether the customer is good or bad.

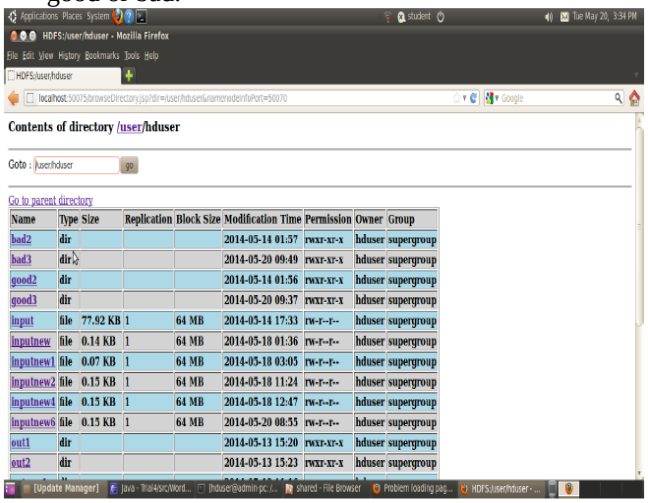

Fig. 8: Contents of directory in HDFS

VI. RESULTS

The project after implementation gives the following results:

A. Partitioning Data using PAM Algorithm

The German credit data set which is taken as input to the map reduce program, each data row of the set i.e. each customer Is partitioned into seven clusters with medoid as the seven propositional rules of Neurorule. By doing this, the total number of customers belonging to each cluster is found out and in a broader picture; the total number of good or bad customers is calculated.

B. Validation of Neurorule

The German credit data set which is taken as input to the map reduce program is used to find the chances of happening of each of the attributes used in the seven propositional rules of the Neurorule for both good and bad clusters of customers separately in two different files. This explains the validity of the neurorule.

C. Credit Risk Evaluation

The data set pertaining to a set of customers along with the twenty sociodemographic attributes is analyzed using neurorule using map reduce programming to find the credit risk of the customers whether they are good or bad. The output file also contains the cluster to which each of the customers belongs to.

VII. CONCLUSION

The above three experiment gives us the validity of neurorule and its application in determining whether the credit risk of the customer is good or bad. The three methodologies mentioned in the sections above gives us an insight as to what neurorules actually are and how the rules extracted by neutral networks play a vital role as parameters in determining and evaluating the credit risk of the

customers and determining if a customer is a credit worthy or not with a good predictive accuracy almost 73.5% accuracy. The experiments were conducted on real life financial credit-risk evaluation data sets, the German credit data set which served as a training set. We have tried to acknowledge the seven propositional neurorules and validate these rules using map reduce programming in java. We have also used these neutral networks extracted rules in determining if a prospect customer is liable to be a good or bad customer depending on the customer's credit-worthiness. Hence, the use of neural network rule extraction, along with decision tables, forms a viable and valuable alternative for building credit-risk evaluation expert systems. As a part of the future work, there is scope for the creation of a more advanced credit risk evaluation system with higher accuracy and precision.

REFERENCES

[1] Setiono and Liu, "Symbolic representation of neural networks, "Academy of Marketing Science, pp. 71-77, 1996.
[2] P. D. H. Hofmann. (2013) Statlog (german credit data) data set. [Online].Available: http://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data)
[3] V. J. and W. G., "Decision tables to expert system shells," pp. 265-282, 1994.
[4] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," OSDI'04 Proceedings of the 6th conference on Symposium on Operating Systems Design Implementation - Volume 6, pp.10-10, 2004.
[5] C. M. Bart Baesens, Rudy Setiono and S. Viaene, "Building credit-risk evaluation expert systems using neural network rule extractionand decision tables," International Conference on Information Systems, pp. 159-168, september 2013.
[6] Poonam and M. Dutta, "Performance analysis of clustering methods for outlier detection," International Conference on Advanced Computing and Communication Technology, pp. 89-95, 2012.
[7] Wikipedia, "k-medoids --- Wikipedia, the free encyclopedia,"2014, [Online; accessed 22-May-2014]. [Online]. Available: http://en.wikipedia.org/wiki/K-medoids

