GenRBFNSpark: A first implementation in Spark of a genetic algorithm to RBFN design

A. J. Rivera, M. D. Pérez-Godoy, F. Pulgar and M. J. del Jesus

Dept. of Computer Science
University of Jaén. Spain
`(arivera,lperez,fpulgar,mjjesus)@ujaen.es`

Abstract. In the world today, there is a need to extract knowledge from large volumes of data, commonly known as Big Data. This data is characterized not only for a high number of instances, but for its high dimensionality, that is a high number of inputs features and even output classes. Classical machine learning methods must be adapted and re-implemented utilizing more performance paradigms for extracting knowledge from these data sets. An example of these paradigms is Map-Reduce, that promotes the parallel and distributed run of the implemented methods. Spark is a powerful cluster computing platform based on Map-Reduce.
Due to dependencies between data that involve a multi-pass computing mode, there are not implementations of Artificial Neural Networks models in well-known machine learning libraries such as Mahout or MLlib. In this paper a Spark implantation of the genetic method to develop Radial Basis Function Networks, GenRBFNSpark, is presented. In the experimentation, different stage times were taken when the algorithm was run with diverse configurations. These times demonstrate that GenRBFNSpark achieves a remarkable speed up with respect to the local version.

Keywords: Genetic algorithm, RBFNs, Big Data, Spark

1 Introduction

Nowadays, we are living in a new age where millions of data is produced in our world, from different sources and every minute. These sources are numerous and heterogeneous because data of almost every process (medical, industrial, social, ...) can be recorded or stored in distinct media [4]. Obviously, several new challenges are appearing for managing these data. Thus, Big Data can be defined as a field composed by different technologies in order to store, process, analyze, visualize, etc., large data volumes.

Big Data [11] is characterized by its well-known Vs: great Volumes of data, great Variety of formats, high Velocity in the generation of the data, mechanisms to ensure the Veracity of data and so on. All these adjectives describe the difficulties for working in the Big Data field. Data Science appears with the objective of extracting knowledge from these volumes of data, on the basis of the traditionally extracting knowledge tasks. However, Data Science needs for new technologies, instruments, etc. to address new challenges. Spark [9] is one of these new technologies that implements a cluster computing platform for Big Data, allowing processing these large volumes of data using the Map-Reduce paradigm. Spark has proven its efficiency versus other Map-Reduce technologies, such as Hadoop [14]. Moreover, it provides MLlib [7] a library with multiple machine learning algorithms.

Radial Basis Function Networks (RBFNs) [2] are one of the most important Artificial Neural Network (ANN) paradigms in the field of Machine Learning. RBFNs have important features such as: a simple topological structure; the possibility of extracting rules; a universal approximation capability; and that each neuron/RBF has a characteristic locally-tuned response. RBFNs have been successfully applied in most important machine learning areas [3].

However, until now it is difficult to find an ANN Map-Reduce implementation in machine learning libraries, such as MLlib or Mahout, or even in the specialized bibliography. This is due to the existent dependencies between data when an ANN is trained that implies multi-pass computations, a scenario where traditional Map-Reduce technologies, as Hadoop, were inefficient. One way to design RBFNs is the Genetic paradigm [10]. However, this successful line requires a high computational cost. In this paper, GenRBFNSpark, an efficient implementation of a Genetic RBFN design algorithm in Spark is presented.

In the experimentation section, GenRBFNSpark is run with different configurations and stage times were taken. These times demonstrated that an important speed up was achieved. Also the experimentation carried out allows to characterize the data sets in order to know in advance what are the parameters that will influence their run time.

The text is organized as follows. In Section 2, Spark technology is described. A explanation of RBFNs are shown in Section 3. The model proposed GenRBFNSpark, is described in Section 4. Finally, the analysis of the experiments and the conclusions are shown in Sections 5 and 6, respectively.

2 Spark

Spark [9] can be defined as a big data processing technology that runs over a cluster platform. As programming model it implements the Map-Reduce paradigm [5]. Spark is an open source project and started in 2009 as a research project in the UC Berkeley RAD Lab, later the AMPLab. The researchers in the lab had previously been working on Hadoop Map-Reduce [14], an adequate solution for one-pass computation, but not very efficient when multi-pass computations are required. Spark presents several advantages [9], compared to Hadoop or other MapReduce technologies, such as: it allows developing multi-step data pipelines using directed acyclic graph (DAG) pattern, supports in-memory data sharing, runs on top of distributed file systems technologies as Hadoop Distributed File System (HDFS), provides 80 high-level operators or an interactively query data shell, supports SQL queries (Spark SQL), streaming data (Spark Streaming), machine learning MLlib), graph data drocessing (Spark Graph X), supports different languages as Java, Scala, Python or R. In summary, from an efficiency point of view, Spark applications run up to 100 times faster in memory, 10 times faster when running on disk, than Hadoop applications.

The main element in Spark is the resilient distributed data set (RDD), which represents an immutable collection of objects partitioned across a set of machines. RDDs are fault tolerant, thus if a partition is lost, the RDD has enough information able to rebuild just that partition. An RDD can be cached in memory across machines and can be reused in multiple parallel operations. An RDD allows two types of operations: Transformations and Actions. A transformation returns a new RDD. Some of the transformations operations are: map, filter, flatMap, groupByKey. etc. An action operation evaluates and returns a new value. Examples of actions are: reduce, collect, count, first, etc. An important difference between transformations and actions is that transformations are "lazy" operations and therefore any result is computed, all the calculations are scheduled. However when an action function is called, all the processing queries, even the scheduled, are computed at that time and a result value is returned.

A key issue for the data science is that Spark provides a machine learning library, MLlib [7]. This library has multiple machine learning algorithms, including classification, regression, clustering, dimensionality reduction and frequent pattern mining, among others. In order to work with these algorithms MLlib defines different data types such as MLlib local vectors and matrices, stored on a single machine or in a distributed way. Another well-known machine learning library is Mahout [13] but it is implemented over Hadoop, therefore MLlib can outperforms Mahout in similar efficiency orders that Spark outperforms Hadoop.

3 Radial Basis Function Networks

From a structural point of view, an RBFN is a feed-forward neural network with typically three layers: an input layer with n nodes, a hidden layer with m neurons or RBFs, and an output layer (Figure 1).

The m neurons of the hidden layer are activated by a radially-symmetric basis function, $\phi_i : R^n \rightarrow R$, which can be defined in several ways, being the Gaussian function the most widely used:

$$\phi_i(\boldsymbol{x}) = \phi_i(e^{-(\|\boldsymbol{x}-\boldsymbol{c_i}\|/d_i)^2}) \tag{1}$$

where $\boldsymbol{c_i} \in R^n$ is the centre of basis function ϕ_i, $d_i \in R$ is the width (radius), and $\|\|$ is typically the Euclidean norm on R^n. In order to deal with nominal attributes the HVDM [15] distance is used. The output node implements the following function, where weights w_{ij} show the contribution of an RBF to the output node:

$$f_j(\boldsymbol{x}) = \sum_{i=1}^{m} w_{ij}\phi_i(\boldsymbol{x}) \tag{2}$$

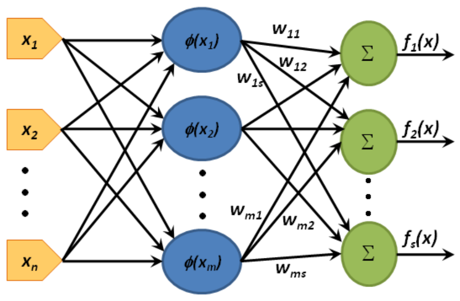

Fig. 1. RBFN Topology

The objective of any RBFN design process is to determine centres, widths and weights. The most traditional learning procedure has two stages: first, unsupervised learning of centres and widths, and then, supervised learning of output weight.

An important paradigm for the RBFN design is Evolutionary Computation (EC). EC uses natural evolution and stochastic searching to design optimization algorithms. A review of EC applied to RBFN design can be found in [3] and the genetic subfield is reviewed in [10].

4 GenRBFNSpark method

GenRBFNSpark is a re-implementation of the Genetic RBFN design method published in [12], for adapting it to the Map-Reduce paradigm promoted by Spark. This method is based on the traditional Pittsburgh evolutionary approach for the design of RBFNs. In this approach each individual is a whole network and therefore contains the coordinates of the center, widths and weights for each RBF. A real codification of the individuals was used considering a variable number of RBFs. The objective of the evolutionary process is to minimize the classification error. The best individual will be the optimal network. The main steps of this algorithm are shown in Algorithm 1.

As an initialization step, for each individual, a random number of neurons (RBFs) between a lower limit and upper limit is considered. The RBFs are allocated using the K-means algorithm [6]. Thus, each RBF center, $\boldsymbol{c}_i$, is placed on a prototype returned by the K-means algorithm. The RBF widths, d_i, will be set to the average distance between the centers and finally, the RBF weights, w_{ij}, are set to zero.

After the initialization, weights are trained using the SVD algorithm [8].

A tournament selection mechanism is applied to the whole group in order to determine the new population. The diversity of the population is promoted by using a low value for the tournament size ($k = 3$).

Mutation and recombination operators are applied to the new RBFNs population. Six mutation operators, usually considered in the specialized bibliography

Algorithm 1 Main steps of Genetic method

```
Initialize RBFN
Train Weights
Evaluation
while Number of Generations not Achieved do
    Mutation
    Recombination
    Train Weights Offspring
    Evaluation Offspring
    Selection
end while
```

[10], were implemented. They can be classified as random operators or biased operators. The mutation random operators are:

- DelRandRBFs: randomly eliminates k RBFs, where k is a pm percent of the total number of RBFs in the RBFN.
- InsRandRBFs: randomly aggregates k RBFs, where k is a pm percent of the total number of RBFs in the RBFN.
- ModCentRBFs: randomly modifies the center of k RBFs, where k is a pm percent of the total number of RBFs in the RBFN. The center of the basis function will be modified in a pr percent of its width.
- ModWidtRBFs: randomly modifies the center of k RBFs, where k is a pm percent of the total number of RBFs in the RBFN. The width of the basis function will be modified in a pr percent of its width.

Mutation biased operators which exploit local information are:

- DelInfRBFs: deletes the k RBFs of the RBFN with a lower weight. k is a pm percent of the total number of RBFs in the RBFN.
- InsInfRBFs: inserts the k RBFs in the RBFN outside the width of any RBF present in the RBFN. k is a pm percent of the total number of RBFs in the RBFN.

With the crossover (recombination) operator two individuals (RBFNs) parents are chosen to obtain an RBFN offspring. The number of RBFs of the new individual will be delimited between a minimum and a maximum value. The minimum value is set to the number of RBFs of the parent with fewest RBFs. In the same way, the maximum value is set to the number of RBFs of the parent with most RBFs. In order to generate the offspring, RBFs will be chosen from the parents at random.

After applying mutation operators, weights are trained using the SVD algorithm. In the evaluation step the fitness for each individual/RBFN is defined as its classification rate.

As it is well known, with Pittsburgh genetic algorithms, where the only objective to optimize is the classification error, the complexity of the individuals (i.e. number of RBFs) grows in an uncontrolled way (because an RBFN with more RBFs usually gives a lower error percentage than an RBFN with fewer RBFs). In this way a maximum and a minimum complexity (chromosome size) was established.

4.1 Key issues in the implementation of GenRbfnSpark

As mentioned, GenRbfnSpark is an implementation of the genetic RBFN design method published in [12], using the Map-Reduce paradigm included in the Spark environment. As base programming language Scala, the native programming language of Spark, is chosen. The fundamentals of the classic genetic methodology are followed and the parallelism promoted by Spark-Scala is exploited to speed-up the most costly steps such as training or evaluation but in no case the individuals are trained with a portion of the data. As advantage this implementation maintains the good performance properties of the genetic algorithms without risk of possible losses in the classification rate.

If the main steps of GenRbfnSpark are analyzed, the following key steps must be highlighted: the initialization step, the weights learning phase and the evaluation (fitness calculation) step. In these steps the whole data set and therefore all the instances are processed for each individual or RBFN. This implies the calculation of the network outputs and therefore of RBFs outputs, using the HVDM distance, a costly measure. In comparison, the remaining steps that involve different operations over the individuals have a cost almost negligible. In this way, we defined and RDD of instances to promote a distributed and fault tolerant environment for operating across the machines of the cluster. In the initialization phase, as we need a version of K-means, we adapted the existing K-means Spark code in the examples directory of the official Spark distribution. This adaptation takes into account: the use of the HVDM distance, implemented by the authors, in order to adequately address nominal attributes; a new add operator for vectors that contain nominal attributes and a sub-procedure to detect similar prototypes, in this case these prototypes are substituted for new instances.

For the weights calculation step, we used the SVD algorithm, specifically the SVD version implemented in the MLlib library. Obviously, we wrapped this version in a high order module that instantiate the necessary matrices and vectors, such as the design matrix. At the end, the results are adequately processed in order to correctly establish the weights of the RBFN.

Finally, evaluating (determine the fitness) an individual or RBFN is also a costly process that involves processing all the instances of the data set in order to obtain the classification error. For this a process a typical Map-Reduce operation has been implemented. In the Map phase each instance is evaluated by the model and the result is compared with the real output, return 1 if they are similar or 0 if they are different. In the reduce phase all these returns are added.

5 Experimentation and Results

With the aim of testing the new GenRBFNSpark implementation the following experimentation has been designed. To begin, five large data sets have been chosen from Keel [1]: *poker, census, connect-4, shuttle* and *letter*. The description of the data sets is in Table 1. The parameters used in GenRBFNSpark as genetic algorithm are show in Table 2. Also, it was tested that achieves similar classification rates that the original one presented in [12].

Table 1. Data set properties

Data sets	#Attributes (R/I/N)	#Examples	#Classes	#Examples-Partition
poker	10 (0/10/0)	1025010	10	922508
census	41 (1/12/28)	142521	3	128269
connect-4	42 (0/0/42)	67557	3	60802
shuttle	9 (0/9/0)	58000	7	52199
letter	16 (0/16/0)	20000	26	18000

For each data set, three experimentations were carried out. As first experimentation, GenRBFNSpark was run in local mode (with the option –master local, one partition was generated). Then, GenRBFNSpark was run in cluster mode with the standard parameters, this configuration divided the data in two partitions. Finally, GenRBFNSpark was run in cluster mode forcing that data was divided in four partitions. For each data set we run the first partition of 10 fold cross validation standard scheme. The cluster used has 16 nodes and each node disposes of 2 x Intel Xeon E5-2670v2 and 64 GB of RAM.

Table 2. Parameters for GenRBFNSpark

Parameter	*Value*
Generations of the main loop	100
Individuals	50
Minimum number of RBFs	number of classes in the data set
Maximum number of RBFs	twice the number of classes in the data set
Crossover probability	0.5
Mutation probability	0.1
Mutation widths/centers percent	0.2
Tournament size	3

Times (in seconds) consumed for each run are shown in Table 3. In this table four times are shown for each partition data set: the initialization time (where the K-means algorithm was applied), the first training time for 50 RBFNs, the first evaluation time (time for calculating the fitness of 50 RBFNs), and the total time (that include the previous times and the time for run 100 generations).

From Table 3, the following analysis can be extracted. In general, for all data sets, times were approximately divided by two when the standard cluster mode was used (2 partitions) instead of the local mode (1 partition). In the same way, times were approximately divided by two when four partitions are used instead of two. These results show a remarkable speed up, around 2, that demonstrates the adequate performance of the developed implementation.

For the run time behavior of GenRBFNSpark, it is necessary not only take into account de number of instances, but also the number and type of input attributes and the number of the output classes. In this way, the number of attributes defines the cost of evaluating an RBF and the number of classes affect to the number of the RBFs in the individuals. For example, *letter* has fewer instances than *census* but many more output classes.

Now, the objective is to characterize the run time of a genetic RBFN paradigm taking into account the parameters of a data set given. It can be observed as *shuttle*, a data set with more than 50000 instances it is the fastest data set, faster than *letter*, which have half of instances. This fact can be explained because *shuttle* has a number of input attributes and output classes relatively low. Therefore, it is extracted as conclusion that in a genetic RBFN paradigm implemented in Map-Reduce the run time are strong linked with the number of input attributes and output classes and can be more important that the number of instances.

Finally, the classification rates achieved by the different runs of GenRBFNSpark are showed in Table 4. As expected, the classification rate for a data set does not depend of the partition configuration chosen for GenRBFNSpark.

6 Conclusions

Currently, there is a need for implementing classical machine learning methods using Big Data computing technologies that speed up the run of large data sets. An example of this technology is Spark, a cluster computing environment that implements the Map-Reduce paradigm in a more efficiency way than others technologies such as Hadoop.

There are not implementations about ANNs in the well-known machine learning libraries due to dependencies between data associated to the model computation. In this paper an implementation of the genetic paradigm for designing RBFNs was presented. From the run times can be concluded that a remarkable speed up was achieved when the algorithm is run. Also, the main characteristics that influence the run time have been identified. Concretely, besides, the number of instances or the number input features, the number of output classes and its possible combination with nominal attributes penalize the run times.

Acknowledgments: This paper is partially supported by the Spanish Ministry of Science and Technology under the Project TIN 2012-33856, FEDER founds.

Table 3. Results: Run time in seconds

Data sets	Step	Local	2 Partitions	4 Partitions
poker	Initialization	29305	16006	6247
	Train (50 Ind.)	4070	2032	1228
	Evaluation (50 Ind.)	2246	1224	602
	Total	177599	89639	50841
census	Initialization	9462	4915	2332
	Train (50 Ind.)	1058	567	283
	Evaluation (50 Ind.)	310	178	86
	Total	39120	20836	10348
connect-4	Initialization	128	228	104
	Train (50 Ind.)	340	194	45
	Evaluation (50 Ind.)	142	94	15
	Total	10585	6309	3299
shuttle	Initialization	1137	563	323
	Train (50 Ind.)	168	89	51
	Evaluation (50 Ind.)	65	47	23
	Total	6196	3442	2065
letters	Initialization	1829	1065	521
	Train (50 Ind.)	274	139	83
	Evaluation (50 Ind.)	341	159	82
	Total	16927	7953	4244

Table 4. Results: Classification rate

Data sets	Local	2 Partitions	4 Partitions
poker	0.550	0.549	0.550
census	0.950	0.950	0.950
connect-4	0.620	0.658	0.666
shuttle	0.882	0.883	0.882
letters	0.661	0.675	0.670

References

1. J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *J. of Mult.-Valued Logic & Soft Computing*, 17:255–287, 2011.
2. D. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

3. O. Buchtala, M. Klimek, and B. Sick. Evolutionary optimization of radial basis function classifiers for data mining applications. *IEEE Transactions on System, Man, and Cybernetics, B*, 35(5):928–947, 2005.
4. IBM Big Data. Ibm - bringing big data to the enterprise. http://www-01.ibm.com/software/data/bigdata/, 2015.
5. J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
6. R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.
7. Machine Learning Library (MLlib) for Spark. Mllib. http://spark.apache.org/docs/latest/mllib-guide.html, 2015.
8. G. Golub and C. Van Loan. *Matrix computations.* Hopkins University Press. 3rd edition, 1996.
9. M. Hamstra, H. Karau, M. Zaharia, A. Konwinski, and P. Wendell. *Learning Spark: Lightning-Fast Big Data Analytics.* O'Reilly Media, 2015.
10. C. Harpham, C.W. Dawson, and M.R. Brown. A review of genetic algorithms applied to training radial basis function networks. *Neural Computing and Applications*, 13:193–201, 2004.
11. M. Minelli, M. Chambers, and A. Dhiraj. *Big Data, Big Analytics:Emerging Business Intelligence and Analytic Trends for Today's Businesses.* John Wiley and Sons, 2013.
12. M. D. Pérez-Godoy, A. J. Rivera, C. J. Carmona, and M. J. del Jesus. Training algorithms for radial basis function networks to tackle learning processes with imbalanced data-sets. *Applied Soft Computing*, 25:26–39, 2014.
13. Apache Mahout Project. Apache mahout. http://mahout.apache.org/, 2015.
14. T. White. *Hadoop, The Definitive Guide.* O'Reilly Media, 2012.
15. D.R. Wilson and T.R. Martinez. Improved heterogeneous distance functions. *Journal on Artificial Intelligence Research*, 6(1):1–34, 1997.