Proceedings 1st International Workshop on Advanced Analytics and Learning on Temporal Data
AALTD 2015

CourboSpark: Decision Tree for Time-series on Spark

Christophe Salperwyck[1], Simon Maby[2], Jérôme Cubillé[1], and Matthieu Lagacherie[2]

[1] EDF R&D,
1 avenue du Général de Gaulle, 92140 Clamart, France
[2] OCTO Technology,
50 avenue des Champs-Élysées, 75008 Paris, France

Abstract. With the deployment of smart meters across many countries, data are being collected at a large scale and volume. These data are collected for billing purposes but also to get analytical insights. Our main goal here is to build an understandable model able to explain the electric consumption patterns regarding several features. We chose to use decision tree models as they are easily comprehensible and have already been parallelized with success. In our industrial context, we often have to work on electrical time-series where the target to predict is neither a label (classification) nor a numerical value (regression) but a time-series representing a load curve. Therefore we use a different split criterion to separate time-series: the inertia. We also need a dedicated method for categorical features since the standard implementation would not work for time-series. This method is based on a hierarchical clustering in order to have a good trade-off between the computational complexity and the exploration of the possible bi-partition splits. We demonstrate the performance of our implementation on datasets with different sizes (up to a terabyte).

Keywords: CourboTree, Hadoop, Spark, Decision Tree, Parallelization

1 Introduction

With the deployment of smart meters across many countries, data are being collected at a large scale and volume. Electric meters measure and transmit electric power consumption from every individual household and enterprise at a rate of a measurement from every 24 hours down to 10 minutes to a centralized information system. In France, EDF provides electricity to more than 35 million customers leading to a massive amount of data to process. These data are collected for billing purposes but also to get analytical insights. Our main goal in this paper is to build an understandable model able to explain the electrical consumption patterns regarding several features such as localization or type of contract. We chose to use decision tree models as they are easily comprehensible and have already been parallelized with success in the Hadoop ecosystem.

CourboSpark is part of the X-Data project: `http://www.xdata.fr/`

Decision trees are well known methods in machine learning which are mainly used for classification and regression tasks. In our industrial context, we often have to work on electrical time-series where the target to predict is neither a label (classification) nor a numerical value (regression) but a time-series representing a load curve. Many versions of decision trees were proposed on top of Hadoop. The Spark [3] implementation seems to be the most suitable for our use-case. Therefore we extended the current Spark/MLlib [4] implementation of decision trees so that it can deal with a time-series as a target.

We first present previous work on parallel decision trees and explain why we choose to reuse the MLlib implementation. Then we describe the time-series regression problem. In section 4, the inertia criterion used to separate time-series is presented. Section 5 will focus on the algorithm adaptation for categorical features with a method based on a hierarchical clustering. Finally, we demonstrate the performance of our implementation on datasets with different sizes (up to a terabyte).

2 Previous work on parallel decision trees

Our goal is to port our in-house software, CourboTree [1, 2], into a distributed system so that more data can be processed. Our CourboTree software build decision tree on time-series based on the inertia criterion.

Many implementations of parallel decision trees have been proposed. In this paper we focus on implementations that can run on top of Hadoop. Hadoop clusters offer a very good trade-off between computing power/storage and price. Hadoop is based on horizontal scaling: the computing power/storage is quasilinear with the number of nodes in the cluster. To have more power, more nodes have to be added into the cluster. Table 1 presents different implementations of decision trees on Hadoop. We used the following criteria to compare these implementations:

- partitioning: horizontal means the algorithm will parallelize computations on the lines, vertical on the columns of the dataset;
- engine: the execution engine that will run the parallel algorithm. These engines can also optimize the execution graph of the algorithm (as for Spark);
- target type: categorical for classification, numerical for regression (can be both)
- ensemble: ability to build an ensemble of trees (random forest, bagging of random trees...)
- pruning: does the algorithm prune the tree to avoid over-fitting?
- open-source: is the source code available so that we can easily reuse it?

As we aimed to reuse an open-source implementation and our datasets mainly grow in an horizontal way, we choose to adapt the MLlib implementation. Moreover it uses the Spark engine which is faster than the original Map/Reduce engine in Hadoop.

	Partitionning	Engine	Target type	Ensemble	Pruning	Open-source
MLlib [4]	Horizontal	Spark	Num + Nom	Yes	No	Yes
MR C4.5 [5]	Vertical	MR	Nom	No	No	No
PLANET [6]	Horizontal	MR	Num + Nom	Yes	No	No
SAMOA [7]	Vertical	Storm/S4	Num + Nom	Yes	Yes	Yes

Table 1. Comparison of parallel decision trees in the Hadoop ecosystem (MR: original Map-Reduce, Storm/S4: streaming frameworks, Spark: new computing engine – widely used to replace MR).

3 Problem description: time-series regression

The problem is the same as the one stated in CourboTree [1, 2]: explain load curves pattern using explanatory features. It can be seen as time-series regression. For this problem, we define our dataset as follows:

- $1, ..., n$: the examples of the dataset;
- $w_1, ..., w_n$: the weights of the examples;
- $X_1, ..., X_j, ..., X_p$: the p explanatory features where x_{ij} is the value for the example i for X_j, these features can be either numerical or categorical;
- $Y_1, ..., Y_k, ..., Y_q$: the q numerical variables defining the time-series where y_{ik} is the value for the example i for Y_k. This time-series is the target of the regression.

Therefore an example i is described as the following tuple: $w_i, x_{i1}...x_{ip}, y_{i1}...y_{iq}$. There can be missing values for either the explanatory features or the time-series.

As in CourboTree, the model used to explain the load curves is a tree:

- l: a node/leaf of the tree,
- g_l: the center of gravity of the examples in l. Its coordinates $g_{l1}, \dots, g_{lq}$ are the weighted mean of the examples in the node/leaf l for the q points of the time-series Y.

4 Split criteria: inertia

Decision trees used in classification aim to lower the impurity in the leaves and use a criterion such as the entropy gain or the Gini coefficient. For regression task, variance reduction is often used. As we deal with time-series we want to lower the variance between the time-series within a leaf/node. In this paper we use the euclidean distance to compute the variance. Other distances could be used.

Given a node t, its intra inertia is defined as:

$$I_w(l) = \sum_{i \in l} w_i \sum_{k=1,\dots,q} (y_{ik} - g_{lk})^2$$

This criteria can be used in the same way as the criterion used in classification and regression trees. The best split to divide a leaf l into two leaves l_L and l_R is the one minimizing the intra-inertia of these two new leaves:

$$argmin\Big(I_w(l_L) + I_w(l_R)\Big)$$

Computing this intra-inertia is expensive but we can use the König-Huyghens theorem which states that the global inertia is the sum of the intra-inertia I_w and inter-inertia I_B:

$$I = I_w + I_B$$

Therefore we can maximize the inter-inertia instead of minimizing the intra-inertia (which is also known as the "Ward's method" [8] in statistics). This corresponds to find the centers of gravity of the two new leaves which are the furthest possible (relatively to their center weights). The computation of the inter-inertia is much more effective. For each potential split points we do not need to recompute the intra-inertia on all the points to the two new centers of gravity, but just the distance between the two new centers of gravity.

For times-series this means to maximize the distance between the average curves in the two leaves (l_L, l_R) having weights (w_L, w_R):

$$argmax\left(\frac{w_L.w_R}{w_L + w_R}\sum_{i\in 1\ldots q}(g_{Li} - g_{Ri})^2\right)$$

As computing the inter-inertia criterion is more effective we have only used this criterion in CourboSpark. Both criteria are sums that can be parallelized. Therefore the computation can be easily spread across the nodes in the cluster in order to take advantage of its computational power.

5 Categorical variables: hierarchical clustering

For numerical feature, the MLlib algorithm has a parameter to specify the number of "bins" that will be tested as split points. These bins correspond to quantiles computed on a random sample of the whole dataset. For the categorical features the next section explains how the current implementation works and how it was adapted for time-series.

Building a binary decision tree with categorical feature requires to evaluate all the bi-partition/split of the modalities. The bi-partition with the lowest value of the criteria is the one chosen to split the node into two leaves. This evaluation can be costly as for m modalities there are $(2^{m-1} - 1)$ possible bi-partitions. With m up to 30, the computation of all these partitions can be computed in a reasonable time (less than a minute on recent hardware). Our datasets have categorical features up to 100 modalities (the number of French departments for example). An exhaustive test of all the bi-partitions would take too much time as the number of bi-partitions to test is exponential with the number of modalities.

There are many ways to deal with categorical features in a binary decision tree in order to limit the number of potential split points. The simplest one is to only test the bi-partitions having one modality versus all the other modalities. A more sophisticated approach is based on ordering the modalities. For classification, modalities are ordered relatively to their entropy or Gini and for regression relatively to their means. This option is the one chosen in the MLlib decision tree implementation. As we deal with time-series, we can not order modalities. We can illustrate that with 4 stable curves: (a) at 0.6, (b) at 0.4, (c) at -0.45 and (d) at -0.55. Their average curve is at 0.0. If we order them by their euclidean distance to the average curve we have: (b,c,a,d), which gives 3 possible bi-partitions: ((b,(c,a,d)), ((b,c)(a,d)) and ((b,c,a)(d)). Looking at this example, the best bi-partitions is ((a,b)(c,d)) but was not evaluated using this heuristic. Therefore we chose to use the hierarchical clustering [8] to find better splits without doing an exhaustive test of all the bi-partitions.

Hierarchical clustering is a bottom-up heuristic. The first step is to aggregate the curve of the two modalities that maximize the loss of inertia. Using the inter-inertia criterion, this aggregation corresponds to the computation of the new average curve based on the weighting sum of the average curve of the two modalities. These two aggregated modalities are seen as one cluster and then we can continue to merge the modalities/cluster in the same way. Figure 1 shows an example of hierarchical clustering with 7 modalities. The algorithm stops once there are only two clusters. In order to optimize the final bi-partition we perform a post-optimization by trying to move all the modalities from one cluster to the other one. Modalities are moved one by one, until no loss of inertia can be achieved. The complexity of this heuristic is much lower: $O(m^3)$.

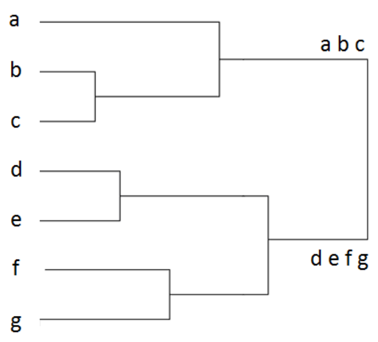

Fig. 1. Example of ascending hierarchical clustering.

6 Experiments

This section presents our experiments on a generated, but realistic, dataset with different sizes.

Dataset: In order to control the experiments we have developed a generator. This generator takes as parameters: the depth of the tree, number of points in the time-series, number of features, number of modalities for categorical features. We configured the generator to have a tree of depth 4, with 10 numerical and 10 categorical features (50 modalities each). Each time-series has 144 points (step of 10 minutes for one day). We generated from 10 to 1,000 millions time-series.

Configuration: The experiments were run on a cluster of 10 machines. Each machine has 32 cores, 48 GB of RAM and 14 SATA spinning disks. Spark was configured to run with 9 executors with 20 GB and 8 cores each.

Results: The results of the the experimentation are presented in Figure 2. As we can see the Spark MLlib implementation of decision tree scales almost linearly with the dataset size.

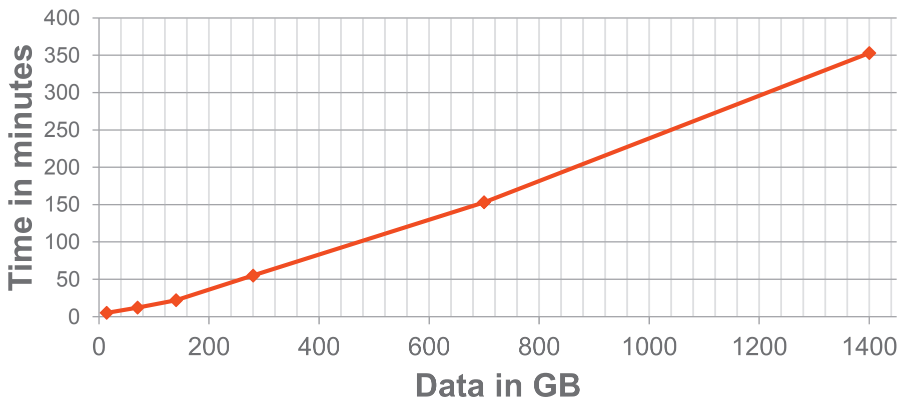

Fig. 2. Time to build the tree depending on the dataset size (10 to 1,000 millions curves).

More experiments were conducted with similar results. We tested datasets with up to 100 features, and also up to 500 modalities for categorical features.

7 Future works

We plan to extend our implementation so that it can deal with "cyclic features" as day of the week, month of the year... which are common in our datasets.

This case is similar to numerical feature but gives two times more possible split points.

The MLlib library only build trees which are balanced (each leaf is at the same depth). The current CourboTree implementation first grows the part of the tree that gives the greatest loss of inertia and therefore can produce unbalanced trees. As we would like to have the lowest global inertia for a given number of leaves, we would either need to drive the tree construction to expand just a part of the tree or to do post-pruning to remove parts of the tree. A next step could be to use this pruning to control over-fitting.

More generally we plan to do more extensive tests to study how the Spark configuration (number of executors, memory...) impact performance depending on the datasets properties (number of features, modalities/bins...).

References

1. Stéphan, V., Cogordan, F.: Courbotree : Application Des Arbres De Regression Multivariés Pour La Classification De Courbes. XXXVIèmes Journées de statistique à Montpellier. `http://www.agro-montpellier.fr/sfds/CD/textes/stephan1.pdf`
2. Stéphan, V.: CourboTree : Une Méthode de Classification de Courbes Appliquée au Load Profiling pp. 129–138. Revue MODULAB 33 (2005). `http://www.modulad.fr/archives/numero-33/stephan-33/stephan-33.pdf`
3. Zaharia M., Chowdhury M., Franklin M.J., Shenker S., Stoica I.: Spark: Cluster Computing with Working Sets, Proceedings of the 2nd USENIX conference on Hot topics in cloud computing (2010).
4. Amde M., Das H., Sparks E. Talwalkar A.: Scalable Distributed Decision Trees in Spark MLLib, Spark Summit 2014. `http://spark-summit.org/wp-content/uploads/2014/07/Scalable-Distributed-Decision-Trees-in-Spark-Made-Das-Sparks-Talwalkar.pdf`
5. Dai W., Wei J.: A MapReduce Implementation of C4.5 Decision Tree Algorithm. International Journal of Database Theory and Application. Vol. 7, No. 1, pp. 49–60 (2014) `http://www.chinacloud.cn/upload/2014-03/14031920373451.pdf`
6. Panda B., Herbach J., Basu S., Bayardo R.: PLANET: massively parallel learning of tree ensembles with MapReduce. Proceedings of the VLDB Endowment, pp. 1426–143 (2009).
7. De Francisci Morales G. , Bifet A.: SAMOA: Scalable Advanced Massive Online Analysis. Journal of Machine Learning Research, Volume 16, pp. 149–153 (2015).
8. Ward, J. H. : Hierarchical Grouping to Optimize an Objective Function. Journal of the American Statistical Association 58 , no. 301 pp. 236–244 (1963).