Regression trees. Application to power systems security assessment.

Technical report. University of Liege.

C. Olaru

September 23, 1997

A regression tree (RT) uses a regression model in a tree structure to predict a continuous numerical variable, rather then a discrete class like in decision (or classification) trees.

The same like for others computer based learning methods, past experience is summarized by a learning sample. This consists of the measurement data on N cases observed in the past together with their numerical computed dependent variable. A case from the *learning set* (LS) consists of data (x,y) where x falls in a measurement space X and y is a real-valued number. The variables in x are referred to as the independent variables, the variable y is usually called the response or dependent variable.

A regression analysis aims at construction of a predictor d(x) for the real value y, starting from a learning sample L, having two purposes: (1) to predict the response variable corresponding to future measurement vectors as accurately as possible; (2) to understand the structural relationships between the response and the measured variables.

After the construction of the predictor d(x), i.e. the building of the tree structure learning by examples from LS, the tree is used to predict the response in new cases. The accuracy of the predictor is determined by testing the tree and estimating the differences between the real value of y and the value predicted by the regression tree. The pruning approach is coming then, trying to improve the reliability and the simplicity of the tree in the same time, making a compromise between the two and getting the right sized tree.

1 Regression tree building

A tree structured predictor is constructed by repeated splits of subsets of LS, into two descendant subsets, beginning with the complete LS. Those subsets which are not split are called terminal subsets and are associated with terminal nodes of the tree, nodes which have no successor nodes. In each terminal node t of a regression tree, the predicted response value $y(t)$ is constant. The non terminal nodes are called internal nodes or test nodes.

The same like in classification, three elements are necessary to determine a tree predictor:
- a way to select a split at every intermediate node;
- a rule for determining when a node is terminal;
- a rule for assigning a value y(t) to every terminal node t.

To determine the best way to split a tree node, a measure of accuracy must be defined. Suppose a learning sample L consisting of N cases $(x_1, y_1), ..., (x_N, y_N)$ is used to construct a predictor d(x). If we have a very large test sample $(x'_1, y'_1), ..., (x'_M, y'_M)$ of size M, the accuracy of d(x) can be measured in different ways:
- as the averaged absolute error

$$\frac{1}{M} \sum_{n=1}^{M} \mid y'_n - d(x'_n) \mid \tag{1}$$

- as the averaged squared error

$$\frac{1}{M}\sum_{n=1}^{M}(y'_n - d(x'_n))^2 \tag{2}$$

where $d(x'_n)$ is the predicted value for y'_n, n = 1, ..., M.

The former measure leads to least absolute deviation regression. The methodology revolving about the latter measure is **least square regression**, needs easer computations then the other one and will be used in our framework.

1.1 Definitions

Assume that the random vector (X, Y) and the learning sample L are independently drawn from the same underlying distribution.

The *mean squared error* $R^*(T)$ of the tree structured predictor d is defined as

$$R^*(T) = E\{Y - d(X)\}^2. \tag{3}$$

A way to estimate internally this error is resubstitution estimation. The learning sample L is used both to construct d(X) and to estimate the error $R^*(T)$. After the predictor is constructed, the cases in L are run through the tree structure predictor. If N is the number of cases in L, then the *resubstitution estimate* for $R^*(T)$ is defined as

$$R(T) = \frac{1}{N}\sum_{n=1}^{N}(y_n - d(x_n))^2. \tag{4}$$

Using this kind of estimator for $R^*(T)$ can give an overly optimistic picture of the accuracy of the predictor, because the resubstitution estimate is computed using the same data used to construct d, instead of an independent sample.

1.2 Growing procedure

The first problem in the construction of the tree is how to use L to determine the binary splits of X into smaller and smaller pieces such that the accuracy of the final tree is as much as possible. The best split at a node t must be one which most successfully separates the high response values from the low ones. The basic top down induction of the tree in the least square regression model aims at produce an as simple as possible tree providing a minimum mean squared error. The algorithm starts with the complete learning set at the top node of the tree. At each step a test function is selected in order to split the current set of examples into two subsets, called the successors of the current node. This process stops when there are not further nodes to be developed.

1.3 Splitting criterion

The splitting criterion is based on an optimal splitting rule which determine the best candidate test to split the current node. It consists of a search for a locally optimal test maximizing a given score function. In order to find out a properly measure for the score function, we need to know how to minimize the resubstitution estimate for the mean squared error.

The response value $y(t)$ in a node t of a regression tree that minimizes R(T) of the tree is the average of the responses for all the cases from L falling into node [BR 84]. That is, the minimizing $y(t)$ is

$$\bar{y}(t) = \frac{1}{N(t)}\sum_{(x_n,y_n)\in t} y_n \tag{5}$$

where N(t) is the total number of cases in t. If we denote the current set of terminal nodes by $\tilde{T}$, we can write

$$R(T) = \frac{1}{N} \sum_{t \in \tilde{T}} \sum_{(x_n, y_n) \in t} (y_n - \bar{y}(t))^2. \tag{6}$$

Setting

$$R(t) = \frac{1}{N} \sum_{(x_n, y_n) \in t} (y_n - \bar{y}(t))^2 \tag{7}$$

then (6) can be written as

$$R(T) = \sum_{t \in \tilde{T}} R(t). \tag{8}$$

That is, the resubstitution estimate for the mean squared error of a tree is the averaged sum of total squared deviations of the y_n in every node t from their average, for all the terminal nodes.

At each node of the tree we must define a candidate set S of binary splits s. Generally this set S is generated by a set of questions Q, where each question in Q is of the form {Is x $\in$ A?, A $\subset$ X}. Because in our case we handle only ordered variable $x_1, ..., x_N$, whose measured values are real numbers, Q includes all questions of the form {Is $x_n <$ c?} for c real number. The associated split s sends all x_n in t that answer "yes" to left successor node t_L and all x_n in t that answer "no" to right successor node t_R. A question from Q is referred to as candidate test and the successors of the node as test outcomes.

Given any set of splits S of a current terminal node t in $\tilde{T}$, the best split s^* of t is defined as the split in S which most decreases R(T) of the tree [BR 84]. More precisely, for any split s of t into t_L and t_R, the decrease of R(T) is

$$\Delta R(s, t) = R(t) - R(t_L) - R(t_R). \tag{9}$$

The best split s^* of t is a split such that

$$\Delta R(s^*, t) = \max_{s \in S} \Delta R(s, t). \tag{10}$$

Thus, a regression tree is formed by iteratively splitting nodes so as to maximize the decrease in R(T).

Let $p(t) = \frac{N(t)}{N}$ be the resubstitution estimate for the probability that a case chosen at random from the learning sample L falls into node t. The sample *variance* of the y_n values in the node t is defined as

$$s^2(t) = \frac{1}{N(t)} \sum_{(x_n, y_n) \in t} (y_n - \bar{y}(t))^2. \tag{11}$$

We can write

$$R(t) = s^2(t) p(t) \tag{12}$$

and (8) can be written as

$$R(T) = \sum_{t \in \tilde{T}} s^2(t) p(t). \tag{13}$$

Then the best split of t minimizes the *weighted variance* [BR 84]

$$s_w^2(s, t) = p_L s^2(t_L) + p_R s^2(t_R), \tag{14}$$

where $p_L = \frac{N_L(t)}{N(t)}$ and $p_R = \frac{N_R(t)}{N(t)}$ are the proportion of cases in t that go left and right, respectively. $(N_L(t) + N_R(t) = N(t))$.

This is equivalent with saying the best split s of node t have to maximize the *variance reduction* provided by the split at the node

$$\Delta s^2(s, t) = s^2(t) - s_w^2(s, t) = s^2(t) - p_L s^2(t_L) - p_R s^2(t_R). \tag{15}$$

In practice, rather than using the variance reduction directly to find the best split for a node, it is preferable to normalize, in order to obtain values belonging to the unit interval [0...1]. In our method we use the normalization of the variance reduction by the variance of the y_n values in that node.

Thus our score measure is defined by

$$Score(s,t) = \frac{\Delta s^2(s,t)}{s^2(t)} = 1 - \frac{N_L(t)}{N(t)}\frac{s^2(t_L)}{s^2(t)} - \frac{N_R(t)}{N(t)}\frac{s^2(t_R)}{s^2(t)}. \tag{16}$$

and the optimal splitting rule can be formulated like: for every current node t it is chosen the split which maximize the above score function.

In order to be able to calculate this score at every node of the tree the information stored at the current terminal nodes should describe the distribution of the response variable $y(t)$, in the form of an estimate of its expected value, $\bar{y}(t)$ in our case, and a value of its variance $s^2(t)$ in the node.

1.4 Stop splitting criterion

The stop splitting criterion is based on a stop splitting rule which allows us to decide whether a current terminal node should be further developed. The growing procedure continues recursively to split the recently created successors until this stop criterion is met for a node. Than this node it becomes a terminal node of the final tree.

There are two situation which lead us to a terminal node. When the variance of the response $y(t)$ in the local learning set of the node is sufficiently reduced, there are no reasons to further split the node and the node is called a *leaf*. Another reason for stopping the split is related to the *over-fitting* problem, which may occur when the learning set of a terminal node becomes too small to allow a reliable choice of a good split and is not possible to reduce the variance further in a statistical significant way. This kind of terminal node is called *deadend*. Stop splitting at deadends must prevent the over-fitting and allow a compromise between accuracy and simplicity of the final tree.

For every new successor node obtained in the growing procedure, first is verified whether the node is a leaf. Then, if it is not, we check whether the node is a deadend. If the node is neither a leaf or a deadend, the node is a non terminal node and will be split further.

The stop splitting rule for a leaf is

$$s^2(t) \leq \beta^2, \tag{17}$$

which is equivalently with

$$\sigma(t) \leq \beta, \tag{18}$$

where $\sigma(t) = \sqrt{s^2(t)}$ is the standard deviation of the y_n values in the node t and β is a threshold, a user defined parameter. The node for which [18] is accomplished is a leaf.

The *total variance* of the response value in a node t is defined as

$$s_T^2(t) = N(t)s^2(t) = \sum_{(x_n,y_n)\in t} (y_n - \bar{y}(t))^2 \tag{19}$$

and the *total variance reduction* in the node can be written

$$\Delta s_T^2(t) = N(t)s^2(t) - N_L(t)s^2(t_L) - N_R(t)s^2(t_R) = Score(s,t)N(t)s^2(t). \tag{20}$$

Two kinds of deadends may appear in the growing procedure, we call them *a priori* deadends and *a posteriori* deadends. We decide whether a node is an a posteriori deadend after we check if the node variance is significantly reduced *because* of the node splitting. An a priori deadend may be reached if the total variance of the node is *already* insignificant, without the node splitting.

The stop splitting rule for an a priori deadend is

$$s_T^2(t) \leq \alpha, \tag{21}$$

i.e., if the total variance in the node is less than a threshold α, the node is an a priori deadend.

The stop splitting rule for an a posteriori deadend is

$$\Delta s_T^2(t) \le \alpha \tag{22}$$

i.e., if the total variance reduction in the node is less then the same threshold, the node is an a posteriori deadend.

Obviously, if [21] is true for a node, [22] will be accomplished anyway.

1.5 The building algorithm

The proposed algorithm for obtaining a regression tree starting with a learning set is briefly described below:

1. for (each new node t, beginning with the root node)
A. if (the node is a leaf) go to **1**
B. if (the node is an a priori deadend) go to **1**
C. if (the node is a test node) compute the test T and the node successors

Further, we use the term *candidate attribute* to denote a independent variable used as input variable for learning, in the growing procedure of the tree. All the candidate attributes available for learning form the vector $x \in X$ of independent variables, denoted here by the term *candidate attributes list*.

The expanding algorithm applied at step **C** consists in the next steps:

C1. for (each attribute a from candidate attributes list)
- compute the test T which provide the maximal score for a
- compute the partition of the current LS, corresponding with chosen T
C2. sort the optimal partitions calculated at **C1**, one for an attribute, in decreasing order by the score
C3. select the first optimal partition from the sorted list for which:
(the node do not become an a posteriori deadend) and
(the score is maximum)
C4. assign to the node t the test T corresponding with the partition selected at **C3**
C5. compute the successors of the test node t by applying the test T

The test T is computed at step **C1** in different ways, depending on the attribute type in the list of the candidate attributes.

For each *ordered* attribute a from the list, the steps for determining the test T corresponding with the maximal score at the current node are:

C1.1. sort the cases from the current LS of the node in increasing order by the values of the attribute a and organize the cases with the same value for a in buckets

C1.2. start from two lists of buckets: yes-list initial empty and no-list initial compound by all the buckets computed at **C1.1**

C1.3. for (each bucket from no-list)
- move the bucket from no-list in yes-list
- compute the score related to currently partition in yes-list and no-list

C1.4. select the partition for which the score was maximum

C1.5. compute the threshold v by making the average between the attribute value of the last case added in yes-list and the attribute value of the first case which was supposed to be removed from no-list

C1.6. construct the test T of type $a < v$

C1.7. establish the splitting of the node in concordance with the test T and its related partition in yes-list and no-list.

In the end of the building procedure of the tree we can notice what are the candidate attributes which influence most strongly the response, looking at the attributes present in all the test nodes. Thus, the tree structured predictor has the capability to identify the structural relationship between the response and the measured variables.

2 Regression tree testing

After the tree is built the next phase is to use it like a predictor, to predict the responses for unseen cases. A new case is directed through the tree starting with the root node and applying the encountered tests until a terminal node is reached. Than the response expected value stored there is used as an estimate of the response in the current case. In the least square regression the expected value stored in every terminal node is the average of the responses for all the cases from LS falling into node in the growing procedure of the tree.

In order to test the capabilites of the final tree, a test set (TS) of unseen cases is used to estimate the errors between the real value of the response and the estimated value in every case.

2.1 Definitions

Let define the following error measures for the regression tree:

- the *mean error*

$$E\{Y - d(X)\}; \tag{23}$$

- the *mean absolute error*

$$R_a^*(T) = E\{\mid Y - d(X) \mid\}; \tag{24}$$

- the *mean squared error*

$$R^*(T) = E\{Y - d(X)\}^2; \tag{25}$$

-the *relative mean squared error*

$$RE^*(T) = \frac{E\{Y - d(X)\}^2}{E\{Y - \mu\}^2} = \frac{R^*(T)}{R^*(\mu)} \tag{26}$$

where μ is a mean of y defined by

$$\mu = E\{Y\}; \tag{27}$$

- the *relative mean absolute error*

$$RE_a^*(T) = \frac{E\{\mid Y - d(X) \mid\}}{E\{\mid Y - \hat{\nu} \mid\}} = \frac{R_a^*(T)}{R_a^*(\hat{\nu})}; \tag{28}$$

where $\hat{\nu}$ is a sample median of y and

$$\nu = \nu(Y). \tag{29}$$

Test sample estimation is a way to estimate these errors efficient and more honest then with a resubstitution estimate. The cases in L are divided into two sets L_1 and L_2 and only the cases in L_1 are used to construct d(X). Then the cases in L_2 are used to estimate the errors. Care needs to be taken so that the cases in L_2 can be considered as independent of the cases in L_1 and drawn from the same distribution.

If M is the number of cases (x, y) in the *test set* (TS), then the test sample estimates for the above error measures are:

- the *test sample estimate for the mean error*

$$\frac{1}{M}\sum_{n=1}^{M}(y_n - d(x_n)); \tag{30}$$

- the *test sample estimate for the mean absolute error*

$$R_a(T) = \frac{1}{M}\sum_{n=1}^{M} \mid y_n - d(x_n) \mid; \tag{31}$$

- the *test sample estimate for the mean squared error*

$$R(T) = \frac{1}{M}\sum_{n=1}^{M}(y_n - d(x_n))^2; \tag{32}$$

- the *test sample estimate for the relative mean squared error*

$$RE(T) = \frac{\frac{1}{M}\sum_{n=1}^{M}(y_n - d(x_n))^2}{\frac{1}{M}\sum_{n=1}^{M}(y_n - \mu)^2} = \frac{R(T)}{R(\mu)} \tag{33}$$

where μ is estimated from the learning sample

$$\mu = \frac{1}{N}\sum_{n=1}^{N} y_n; \tag{34}$$

- the *test sample estimate for the relative mean absolute error*

$$RE_a(T) = \frac{\frac{1}{M}\sum_{n=1}^{M} \mid y_n - d(x_n) \mid}{\frac{1}{M}\sum_{n=1}^{M} \mid y_n - \hat{\nu} \mid} = \frac{R_a(T)}{R_a(\hat{\nu})} \tag{35}$$

where $\hat{\nu}$ is estimated from the learning sample.

2.2 The testing algorithm

The proposed algorithm for testing a regression tree using a test set is briefly described below:

1. for (each case from TS)
A. direct the case through the tree, applying the encountered tests until a terminal node is reached
B. take the estimated value of the response stored in this terminal node and compare with the real value already known from the data base
C. compute the errors

At step **C** are computed all the errors defined in section [2.1]. For every distribution of error, squared error and absolute error, the next statistics are computed: the maximum value, the minimum value, the mean value, the standard deviation (σ) and the standard error estimate ($\sigma/\sqrt{M}$).

3 Regression tree pruning

The stop splitting criterion for growing a RT is responsible for the complexity of the final tree and in our case, in some situations, has deficiencies, transposed in dishonest results given by the tree. For every practical problem for which the RT approach is applied exists three types of trees, in terms of complexity: the too small tree, obtained with a too high threshold α in the stop splitting rule, the right sized tree and the too large one, resulted using a too low threshold α. By definition, the *complexity* of a tree is $C(T) = \mid \tilde{T} \mid -1$ where $\tilde{T}$ is the number of the terminal nodes. In the case of binary trees, this is equal with the number of test nodes. Usually, the user dœs't know a priori the value of α corresponding with the right sized tree.

A too large tree will have higher true errors then the right sized one, presenting higher variance, and a too small tree will not use some of the information available in L, tending to be biased. The both bias and variance are generating incorrect results for the prediction. Therefore, in order to find the right sized tree, the one who makes a compromise between variance and bias, a satisfactory approach is: instead of stop the splitting when a threshold is reached, to continue the splitting until all terminal nodes are very small, resulting in a large complete tree, then to prune selectively this large tree getting a decreased sequence of subtrees, and finally, to choose from this sequence, the subtree having the lowest error, i.e. the subtree providing the highest global variance reduction.

3.1 Definitions

The sample variance of the y_n values in the root node t_0 may be written as

$$s^2(t_0) = s^2(T) = \frac{1}{N}\sum_{n=1}^{N}(y_n - \bar{y}(t_0))^2 \tag{36}$$

where

$$\bar{y}(t_0) = \frac{1}{N}\sum_{n=1}^{N} y_n \tag{37}$$

and N is the number of cases in the learning sample.

The *total variance of a RT* is defined as the total variance in the root node

$$s_T^2(T) = Ns^2(T) = \sum_{n=1}^{N}(y_n - \bar{y}(t_0))^2. \tag{38}$$

The *total variance reduction provided by a RT* is the reduction of the total variance of the RT by the total residual variance at its terminal nodes

$$\Delta s_T^2(T) = N[s^2(T) - \sum_{k=1}^{\tilde{T}} p_k s^2(t_k)] = Ns^2(T) - \sum_{k=1}^{\tilde{T}} N(t_k)s^2(t_k) \tag{39}$$

where $p_k = \frac{N(t_k)}{N}$ is the proportion of cases in L that reach the terminal node t_k of the tree, $\sum_{k=1}^{\tilde{T}} p_k = 1$ and $s^2(t_k)$ is the variance in the terminal node t_k.

By analogy with decision tree approach [WE 94], the *quality* of a regression tree is defined as a linear combination of the total variance reduction of the tree and the tree complexity:

$$Q_\alpha(T) = \Delta s_T^2(T) - \alpha C(T) \tag{40}$$

where the parameter α is in our case the threshold used for stopping splitting at a deadend.

The quality measure has the advantage of being additive, for any arbitrary decomposition of the tree into subtrees, the tree quality being equal with the sum of the component subtrees qualities [WePa 93]. In the same time, $Q_\alpha(T)$ allows to compare different trees. The pruning procedure is relying on these properties.

3.2 Sequence of pruned trees

For every value of α must be found the subtree $T_\alpha \prec T$ which maximizes the quality measure, i.e.

$$Q_\alpha(T_\alpha) = \max_{T_{\alpha_k} \prec T} Q_\alpha(T_{\alpha_k}). \tag{41}$$

If we think of α as the complexity cost per terminal node, the quality measure is formed by subtracting from the total variance reduction of the tree a cost penalty for complexity. If α is small, the penalty for having a large number of terminal nodes is small and T_α will be large. For α sufficiently large, the maximizing subtree T_α will consist of the root node only and the tree T will have been completely pruned.

Although α runs through a continuum of values, there are at most a finite number of subtrees of T. Thus, the pruning process produces a finite sequence of subtrees $T = T_{\alpha_1} \succ T_{\alpha_2} \succ ... \succ \{t_0\}$ and a corresponding increasing sequence of α values $0 = \alpha_1 < \alpha_2 < ...$, such that, for $\alpha_k < \alpha < \alpha_{k+1}$, T_{α_k} is the smallest subtree of initial T maximizing the quality. In particular, for $\alpha = 0$ the pruned tree is the full tree T and for $\alpha \to \infty$ the pruned tree shrinks to a single nod with zero complexity.

In the process of generating the sequence of pruned trees a new set of cases is considered, different from the initial learning set LS used for building the complete tree T. We denote it by the term *pruning set* (PS).

3.3 Error estimation and tree selection

To select the right sized tree from the sequence of subtrees, honest estimates of errors are needed. In our case, resubstitution estimates of the errors defined in section [2.1] are used, considering the cases from the same PS, which are supposed to be independent of the cases in LS and TS.

Then the best subtree in terms of quality is extract from the sequence looking at the error estimators. The "n SE rule" is applied in order to choose the simplest tree whose error is comparable with the lowest error found in the whole sequence.

Looking at the mean absolute error (MAE) estimate of the subtrees, if T^* is the subtree for which

$$R_a(T^*) = \min_k R_a(T_{\alpha_k}) \tag{42}$$

and SE the standard error of the MAE estimate, then the tree selected T_α is the smallest tree satisfying

$$R_a(T_\alpha) \leq R_a(T^*) + n * SE. \tag{43}$$

For $n = 0$ the rule its reduced to the selection of the subtree with the minimum mean absolute error estimate. For $n = 1, 2, ...$ the standard error is considered and the selected tree will be prevented from fitting too perfectly the PS, therefore unbiased.

3.4 The pruning algorithm

The proposed algorithm for pruning a regression tree is briefly described below:

A. build a complete tree setting $\alpha = 0$
B. for (given values of increasing α) extract the corresponding pruned tree of maximal quality
C. select the tree with minimal MAE estimate and compute its SE
D. choose the optimal pruned tree in accordance with 1 SE rule.

At step **B**, the pruned trees are extracted in a bottom up fashion (forward pruning) or in a top-down approach (backward pruning). The same like for decision trees, in practice, the corresponding sequences of pruned trees are slightly different.

4 Experimental simulations of RTs

4.1 OMIB results - illustrative transient stability problem

The data base is composed of 5000 normal simulated operating states (see [WE 97]). The candidate attributes considered are P_u and Q_u (the amount of active and reactive power of the generator) randomly sampled uniformly in the ranges (700...1300 MW) and (-665...990 MVar) respectively. The goal is to compute the critical clearing time (CCT), measure of the system security level in terms of transient stability. The response of the RT in the building stage is considered the CCT computed for every case by a step-by-step dichotomic search, the standard deviation of the numerical computation error of the CCT in the data base being estimated to about 0.002 ms. In order to get a tree as accurate as possible, the parameter β for the stop splitting at a leaf is chosen 0.002 ms, i.e. a node is a leaf when his variance is at most 0.000004 ms. The parameter α is set $\alpha = 0$ in order to get a full tree in the first stage of the pruning approach.

Table 1: Complexities before and after pruning.

N	$C(T)$	$C(T_\alpha)$
100	74	42
250	162	122
500	273	224
750	369	297
1000	433	374
1500	588	499
2000	662	594
3000	824	715

Table 2: Mean absolute errors before and after pruning.

N	$R_a(T)$ - *msec*	$R_a(T_\alpha)$ - *msec*
100	0.013703	0.014142
250	0.007925	0.008078
500	0.006183	0.006364
750	0.005097	0.005134
1000	0.004568	0.004708
1500	0.003611	0.003638
2000	0.003439	0.003515
3000	0.002891	0.002891

The experiments are made for different sizes of LS in order to study the influence of the LS size N on the results (complexity and error). The LSs are chosen randomly from the first 3000 states of the data base, PS includes the states from 3001 to 4000, and TS - the last 1000 states from the data base. After pruning (forward pruning), the both trees, complete and optimal pruned, are tested on the same TS. The results in terms of complexity are shown in table 1 and in terms of absolute errors - in table 2. Table 3 presents the CPU times for building and pruning the full trees on a SUN Sparc 10 workstation. The optimal parameter α resulted after pruning, the one who determine the right sized tree is presented in table 4 for every LS size considered.

As we can notice from tables 1 and 2, the complexity increases with the size of LS and the mean absolute error decreases with the size of LS. Fig. 1 depicts the values of complexity corresponding to the optimal pruned tree in comparison with the complexity of the full tree and fig. 2 - the values of MAE corresponding to the optimal pruned tree for every size.

CPU time is a increasing function of N (see table 3) and implicitly depends of tree complexity, being a result of the computations amount.

A imposed observation is that the optimal pruned trees are much simpler then the initial trees and the accuracy is quite slightly different, i.e. the complexity is significantly reduced with a insignificant increase in error.

Table 3: CPU times.

N	building - *sec*	pruning - *sec*
100	2	4
250	2.35	10.8
500	2.8	12.7
750	3.7	22.7
1000	5.1	58.7
1500	7.6	72.7
2000	10.5	91.2
3000	15	112

Table 4: Optimal value of parameter α.

N	100	250	500	750	1000	1500	2000	3000
α	0.000125	0.000040	0.000023	0.000019	0.000019	0.000017	0.000016	0.00002

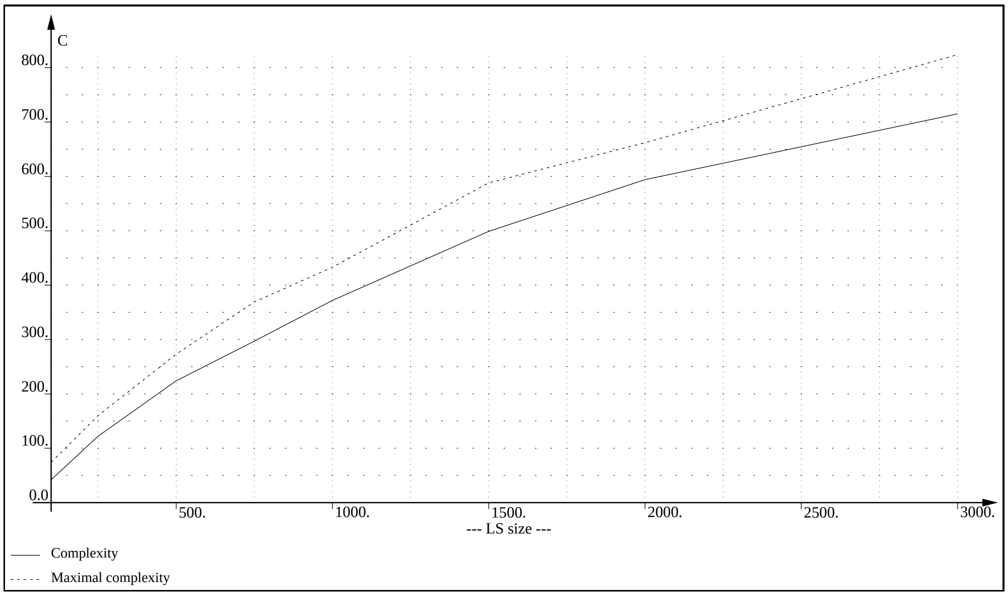


Figure 1: Influence of N on the complexity.

In all the cases, the attribute from the attributes list which influence most the CCT response of the RT is found to be P_u, the amount of active power of the generator. The RTs explain in the OMIB problem almost 100% of the variance of the CCT, the total variance reduction in percentages being in the range [99.898%...99.971%] for all the trees.

4.2 VST data base results - voltage stability problem

The data base is composed of 5000 pre-disturbance states (see [WE 96]), characterized by pre-contingency attributes, i.e. parameters describing the topology and the electrical state, and by pre- and post-contingency load power margins (LPM) pre computed for various contingencies by classical methods. For building the trees 136 candidate attributes are considered. The goal is to estimate the severity of a contingency in a given operating state, i.e. the difference between pre- and post-contingency margins. This determines less complex trees and more interpretable then in the case of estimating the post-contingency LPM value [WE 96]. Knowing the pre-contingency margin and the severities for different operating states, for a given contingency, the post-contingency margin can be then computed easily, the parameter who gives us information about how much stabile is the system in terms of voltage stability.

For a given contingency the LS, PS and TS are selected as the relevant operating states for this contingency, respectively from the first 3000 of the data base, from the next 1000 and from the last 1000. Three contingencies are considered in order of increasing problem complexity:
- the tripping of a synchronous condenser;
- the tripping of a line;
- the tripping of a generator.

The table 5 shows the number of relevant states in LS, PS, TS, for every contingency.

The parameters α is chosen $\alpha = 0$ for getting a full tree and for β parameter different values are considered, in order to see how this parameter influence the tree accuracy and complexity.

The results in terms of complexity, errors and CPU times are presented in tables 6, 7 and 8, respectively, for every contingency. The last columns in this tables show the optimal parameter α resulted after pruning

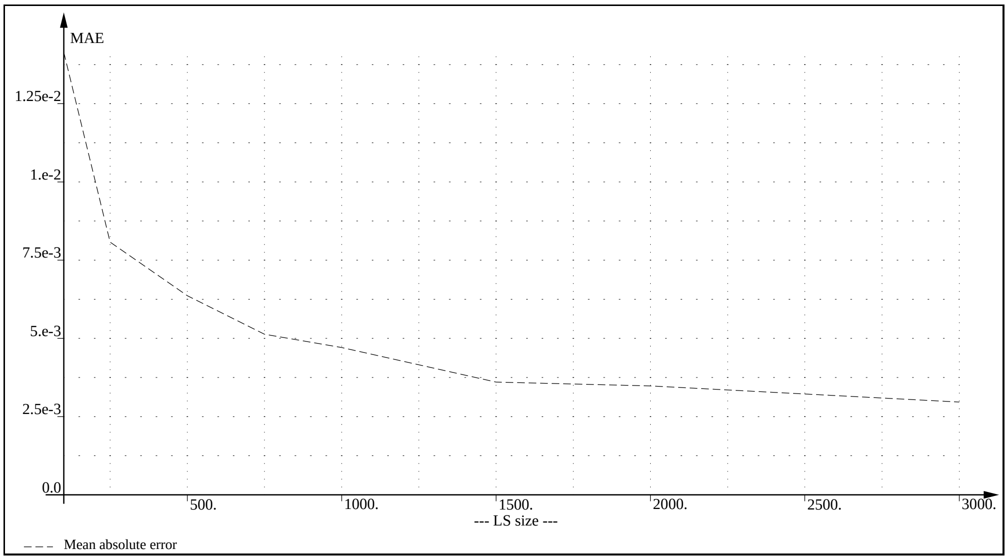


Figure 2: Influence of N on the mean absolute error.

Table 5: Relevant states.

Disturbance	LS size	PS size	TS size
Synch. cond. tripping	2697	911	919
Line tripping	2775	922	913
Generator tripping	2430	815	796

for every case.

Fig. 3 draws a regression tree for the synchronous condenser tripping contingency, and $\beta = 25$. Each node of the tree is represented by a box. Above the box appears the name of the node, test (T), leaf (L) or deadend (D) and the number of its learning states. The total number of different nodes is indicated above the top-node (the root node). The node box itself is subdivided into upper and lower parts. Their relative heights are proportional to the relative sizes of the learning and test sets at the node. In the upper part, the mean value of the corresponding subtree response (security in this case) and its standard deviation (in brackets) for the learning set are indicated. In the lower part, the mean value of the subtree absolute error and its standard deviation for the test set are indicated (for the root node the mean and standard deviation are referring to the global absolute error of the tree). Every part of a node box represents these values so as one can see how larger is standard deviation at the node and how the variance decreasing from top-node to terminal nodes. Below the test nodes, the corresponding tests are indicated. Analyzing the tree, the attributes which influence most the system behavior after contingency can be extracted looking at the test nodes.

Fig. 4 shows the upper part (27 test nodes from the total of 88) of the initial complete tree who generated by pruning the tree from fig. 3. Figs. 5, 6, 7 and 8 present the complexity, the mean absolute error, the total variance reduction and the quality respectively, depending on the α evolution in the generating process of forward pruning sequences for the tree considered in fig. 4. Fig. 9 sketches the error distribution of the same RT with respect to the pre-computed severity in the data base, for the learning set. One can notice the distribution in 7 oblique lines, due to the fact that the RT has 7 terminal nodes and the same number of posible constant results. Fig. 10 shows the RT error distribution.

The results obtained for the considered contingency are showing that:

- the complexity in every case decreases very much after pruning and is a decreasing function of β; also, the complexity of full tree is significantly higher for the generator tripping contingency, fact generated because of the complexity of the problem (high mean severity, high variability) [WE 96] ;
- the accuracy of the trees is better in every contingency case for a value of $\beta = 25$, a β too high generating

Table 6: Synchronous condenser tripping.

β	$C(T)$	$C(T_\alpha)$	$R_a(T)$ - MW	$R_a(T_\alpha)$ - MW	building - sec	pruning - sec	α
35	15	4	26.151824	25.878187	74	2.5	63375
25	88	6	26.198262	24.644522	890	16.7	63375
10	608	6	26.244044	24.644522	3749	80	63375

Table 7: Line tripping.

β	$C(T)$	$C(T_\alpha)$	$R_a(T)$ - MW	$R_a(T_\alpha)$ - MW	building - sec	pruning - sec	α
35	38	5	23.32217	22.93463	483	76	491379
25	80	7	22.03426	21.98475	1118	126	263724
10	544	7	26.07002	21.98475	3698	311	263724

less accurate trees and more complex; also, for the generator tripping contingency the mean absolute errors are much bigger then for the others contingencies, due to the same observation about complexity problem, meaning that the RT does not learn well the relationship between the attributes and severity in this particular case; the error of the tree prediction for the severity is *reduced* after pruning, due to the apparition of over-fitting phenomenon in the stage of learning (see fig. 6);

- CPU times for both building and pruning are decreasing with β and increasing with the tree complexity, as expected, but, this values are referring to the full trees, once we determined the optimal value of α for a given problem, building directly the corresponding tree takes much less time.

References

[BR 84] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone, "Classification and regression trees", 1984;

[WePa 93] L. Wehenkel, M. Pavella, "Decision tree approach to power system security assessment", Electrical Power and Energy Systems, Vol. 15, No. 1, Feb. 1993;

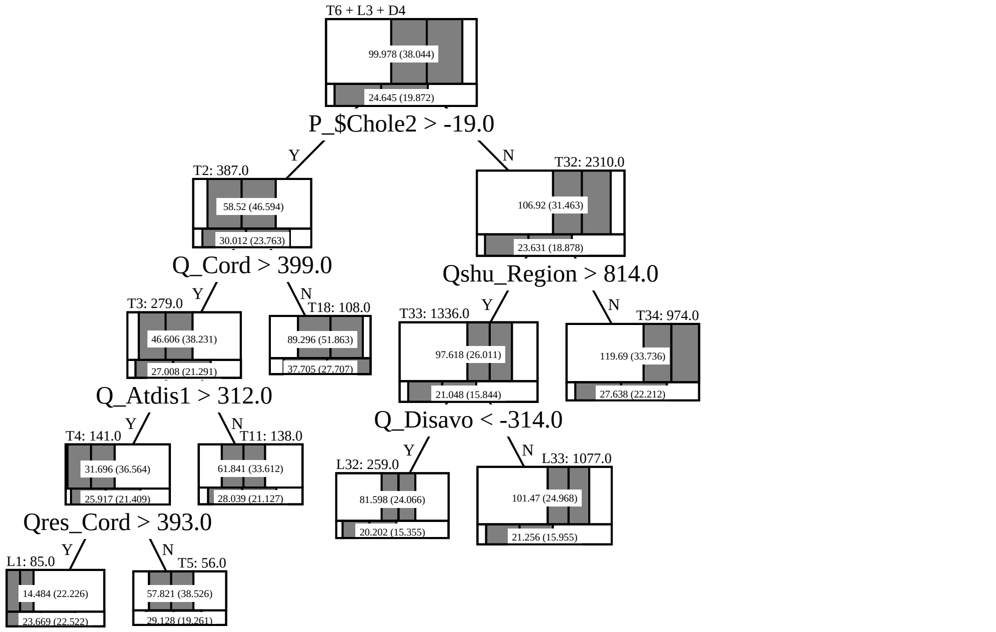
[WE 94] L. Wehenkel, "Machine Learning approaches to power system security assessment", Aggregation thesis, University of Liege, Belgium, may 1994;

[WE 96] L. Wehenkel, "Contingency severity assessment for voltage security using non-parametric regression techniques", IEEE Transactions in Power Systems, Vol. 11, No. 1, Feb. 1996;

[WE 97] L. Wehenkel, Y. Jacqumart, "Tutorial course on Automatic Learning Methods. Application to Dynamic Security Assessment", PICA'97 - IEEE Power Industry, feb. 1997.

Table 8: Generator tripping.

β	$C(T)$	$C(T_\alpha)$	$R_a(T)$ - MW	$R_a(T_\alpha)$ - MW	building - sec	pruning - sec	α
35	182	26	44.693457	42.099810	1372	42	74991
25	354	26	47.805284	42.099810	2086	114	74991
10	855	26	49.634436	42.099810	2980	164	74991



Learning set regression (w.r.t. DELTA_MARP)

- Very Negative : -7.743 -> Mean-Sigma (= 61.934 In Ls)
- Negative : Mean-Sigma -> Mean (= 99.978 In Ls)
- Positive : Mean -> Mean+Sigma (= 138.02 In Ls)
- Very Positive : Mean+Sigma -> 153.42

Test set regression

Error-Type	Number	Mean	Min	Max	Sigma	St.Err.
Negative	479	24.2005954	0.4707521	119.6879	19.8154921	0.9053929
Positive	440	25.1277972	0.312115	178.3121	19.9228263	0.9497835
Total	919	24.6445223	0.312115	178.3121	19.8723527	0.6555284

Figure 3: Synchronous condenser tripping. Optimal pruned tree. N = 2697, M = 919. $\beta = 25$, $\alpha = 63375$.

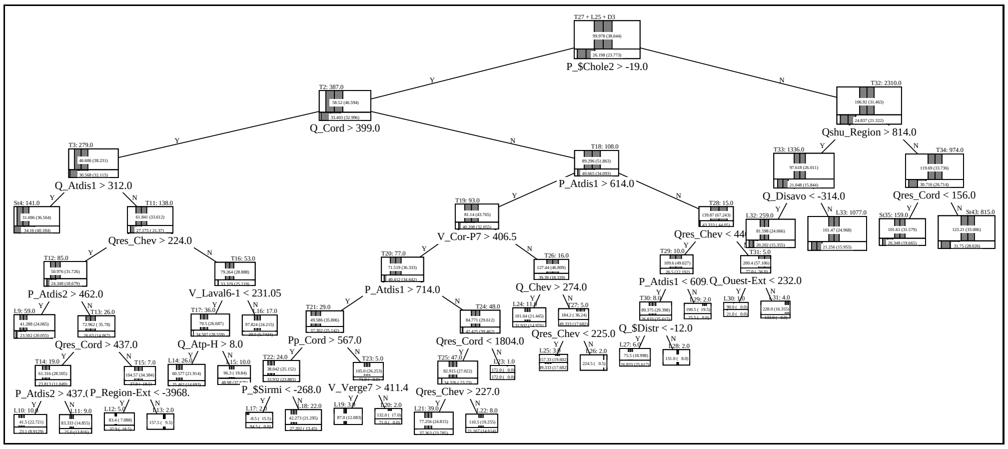


Figure 4: Synchronous condenser tripping. Initial complet tree. N = 2697, M = 919. $\beta = 25$, $\alpha = 0$.

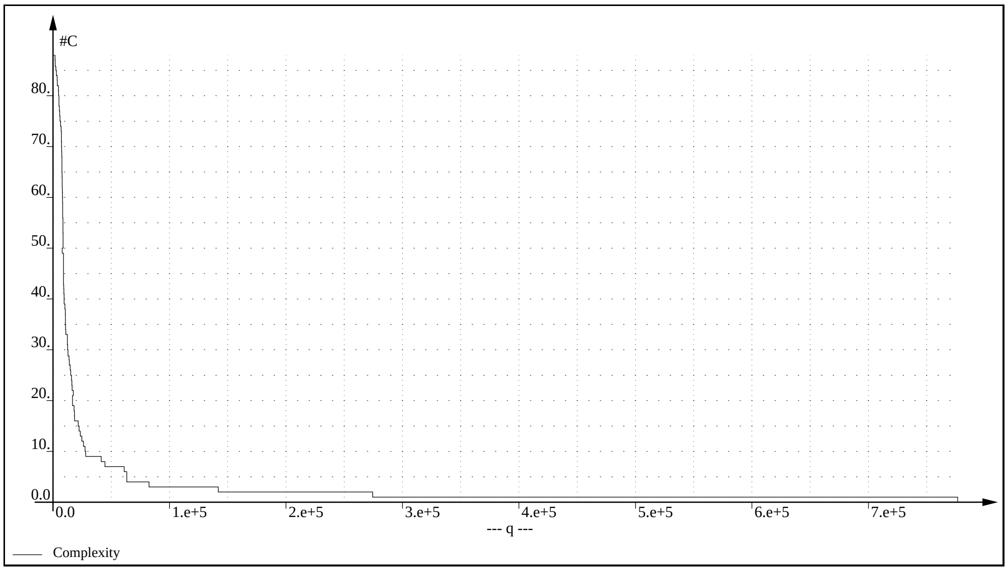


Figure 5: Forward pruning sequences. Complexity.

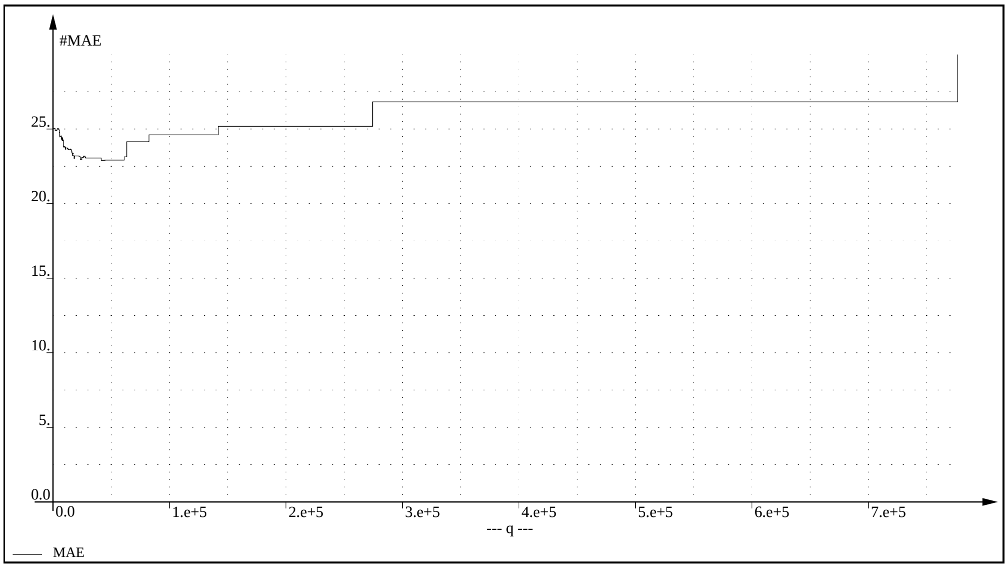


Figure 6: Forward pruning sequences. Mean absolute error.

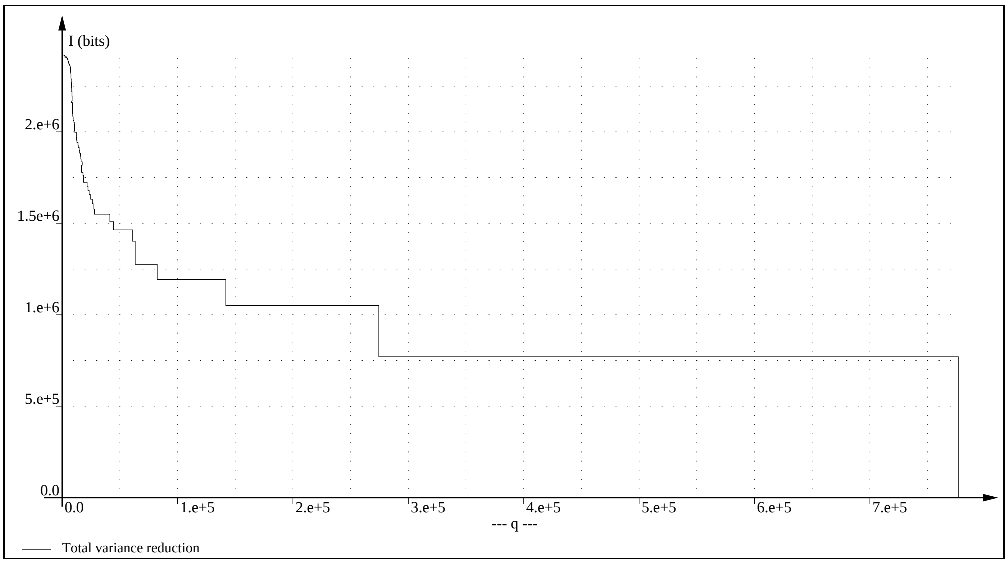


Figure 7: Forward pruning sequences. Total variance reduction.

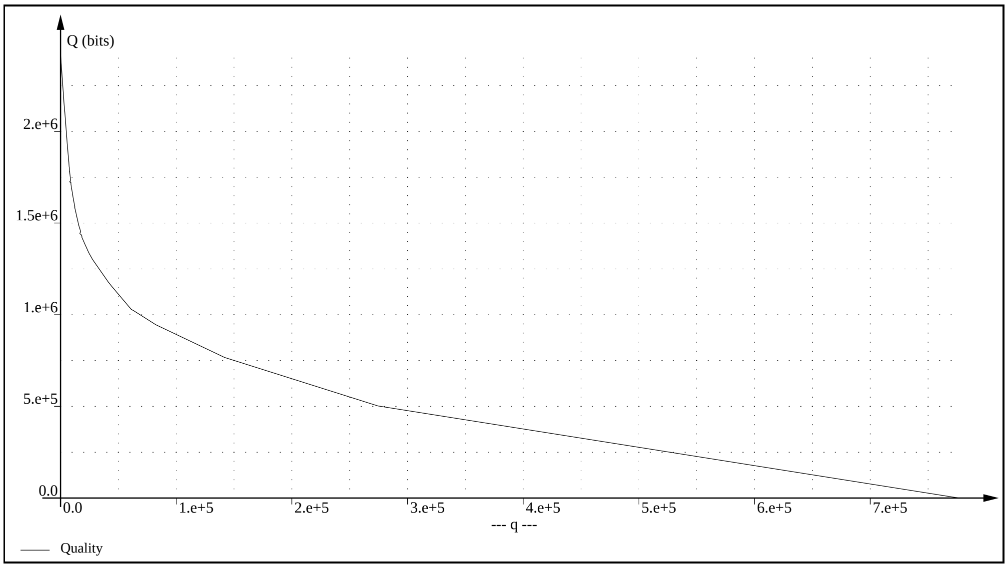


Figure 8: Forward pruning sequences. Quality.

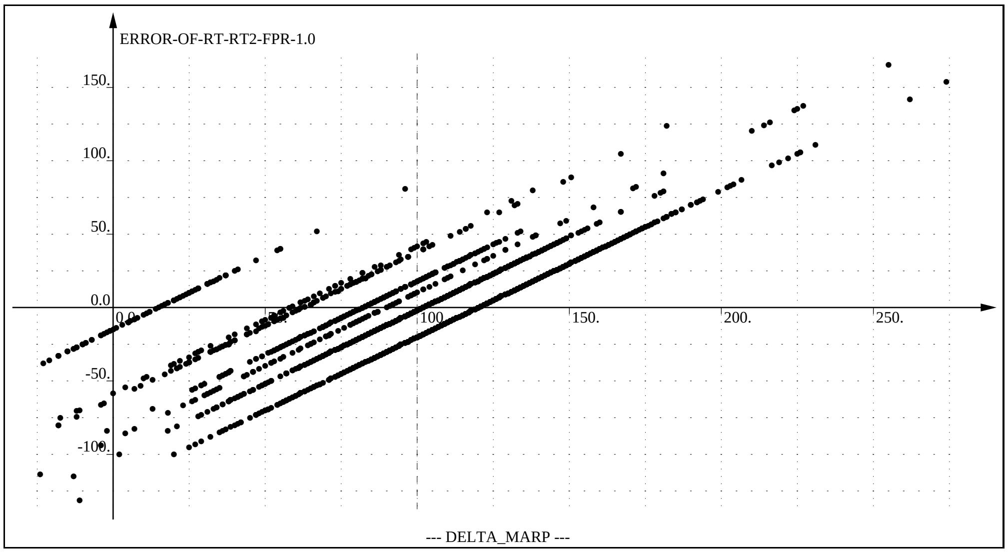


Figure 9: Correlation severity - errors of pruned RT.

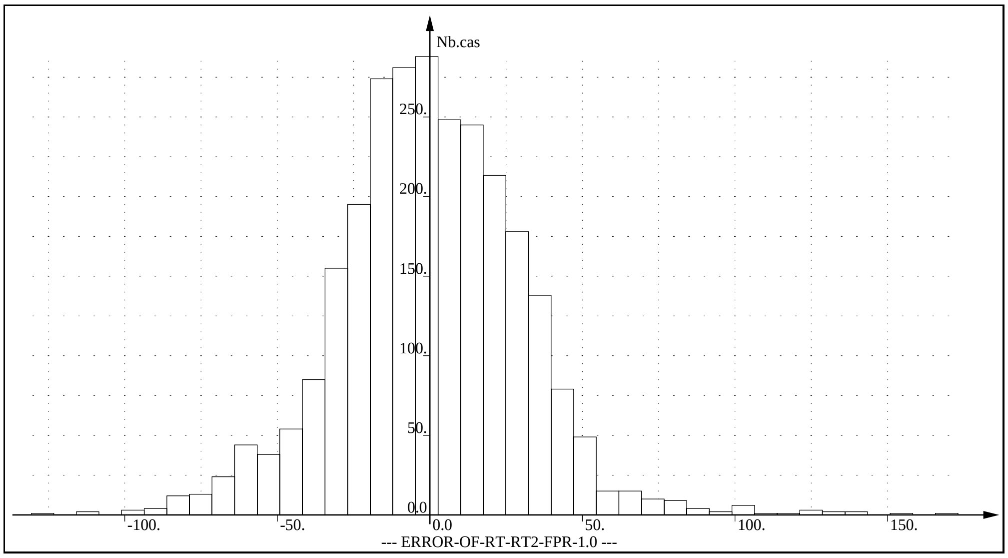


Figure 10: Distribution of RT errors.