Tutorial: Getting Started with TreeRank in R

Nicolas Baskiotis
ENS Cachan - CMLA UMR CNRS No. 8636
Stéphan Clémençon
LTCI UMR Telecom ParisTech/CNRS No. 5141

November, 2010

Abstract

TreeRank is a learning algorithm tailored for ROC optimization in the context of bipartite ranking. It is the purpose of this note to provide a tutorial introduction for the use of the TreeRank package for R statistical software.

1 Introduction

Bipartite ranking, sometimes termed *nonparametric scoring*, is an ubiquitous issue, encountered in anomaly detection, medical diagnosis, credit-risk screening, or information retrieval for instance. In a wide variety of applications, practitioners need to learn scoring/ranking functions for discriminating between two populations from multivariate data with binary labels.

Precisely, here data consist of a set of individual measurements $x = (x^{(1)}, \ldots, x^{(q)}) \in \mathcal{X}$ (the *input* variables), to which a binary label $y \in \{-1, +1\}$ (the *output* variable). These measurements are assumed to be independent realizations of a pair (input/output) of random variables (X, Y). A part, if not all, of this historical database, with $n \geq 1$ observations $\mathcal{D}_n = \{(x_i, y_i) : 1 \leq i \leq n\}$, is dedicated to learning of a *scoring rule* $s : \mathcal{X} \to \mathbb{R}$. By transporting the natural order on the real line onto $\mathcal{X}$, s defines a ranking (*i.e.* a preorder) through: $\forall (x, x') \in \mathcal{X}^2$,

$$x \preceq_s x' \Leftrightarrow s(x) \leq s(x').$$

In bipartite ranking, the goal pursued is to use the training set $\mathcal{D}_n$ in order to produce an accurate ranking, ranking performance being evaluated through ROC analysis. The ROC curve of a scoring functions is the plot of the *false positive rate* against the *true positive rate*:

$$\mathrm{ROC_s} : \mathrm{t} \in \mathbb{R} \mapsto (\mathbb{P}\{\mathrm{s(X)} > \mathrm{t} \mid \mathrm{Y} = -1\}, \mathbb{P}\{\mathrm{s(X)} > \mathrm{t} \mid \mathrm{Y} = +1\}).$$

The closer the to the left upper corner of the ROC space, the more accurate the ranking (instances with positive labels are expected to be top ranked). In regards to this (theoretical) functional performance criterion, increasing transforms of the *regression function* $\eta(x) = \mathbb{P}\{Y = +1 \mid X = x\}$ are the optimal elements (their ROC curve dominates any other ROC curve, everywhere along the false positive rate axis). In practice, an estimate $\widehat{\mathrm{ROC}}_s$ of $\mathrm{ROC_s}$ is computed the following way: one calculates, using a sample $\{(x_i, y_i) : 1 \leq i \leq n\}$ of realizations of the pair (X, Y), empirical counterparts of the class probabilities $\mathbb{P}\{s(X) > t \mid Y = -1\}$ and $\mathbb{P}\{s(X) > t \mid Y = +1\}$ for

threshold values $\{s(X_i) : \ Y_i = -1, \ 1 \leq i \leq n\}$ and next connects the corresponding knots by line segments, producing a piecewise linear ROC curve (notice that another convention, sometimes encountered in practice, consists in plotting a stepwise ROC estimate from these knots). The ROC curve estimate plotted using training data (*i.e.* those used for building s) is called the *training* ROC *curve*, while that plotted using a sample independent from the training set is called a *test* ROC *curve*. A common summary statistic is the *area under the empirical* ROC *curve* (empirical AUC in short), which is a consistent estimate of the quantity

$$\mathrm{AUC(s)} = \mathbb{P}\{\mathrm{s(X)} < \mathrm{s(X')} \mid \mathrm{(Y, Y')} = (-1, +1)\} + \frac{1}{2}\mathbb{P}\{\mathrm{s(X)} = \mathrm{s(X')} \mid \mathrm{(Y, Y')} = (-1, +1)\},$$

where (X', Y') denotes an independent copy of the pair (X, Y). Basic results in ROC analysis from the perspective of bipartite ranking can be found in Section II of [6] (see also the Appendix therein).

The software we present here implements a novel statistical learning algorithm, named TreeRank, for ROC curve optimization. This algorithm is a tree induction procedure, entirely tailored for bipartite ranking and producing, from a training sample, a piecewise constant scoring function of which ROC curve mimics the behavior of an adaptive linear-by-part interpolant of the optimal ROC curve [6, 5]. It produces models that can be easily summarized in the form of an oriented rooted binary tree graph. The ranking can be directly read by perusing the terminal leaves from the left to the right.

As for many other tree-based learning methods, the procedure includes two stages. A greedy top-down recursive partitioning strategy is first implemented, leading to a complete rooted binary tree equipped with a left-right orientation, we call it the *Master Ranking Tree*. Each split corresponds itself to a classification rule, obtained through a *cost-sensitive version of a binary classification methodology* (the celebrated CART method [1] for instance, or SVM methods), we call LeafRank in this context, the cost locally depending on the data, being equal to the empirical rate of positive instances within the node to split, in order to maximize the AUC criterion recursively. A pruning procedure then follows the growing stage, where children of a same parent node are recursively merged so as to maximize a cross-validation based estimate of the AUC criterion. The ROC curve of the resulting scoring function can be shown to converge to the optimal one, in the AUC sense and in sup norm both at the same time. The procedure is described in detail in [6, 3], together with the related background theory.

Beyond the theoretical properties of this ranking algorithm, a crucial advantage of such a tree-structured recursive partitioning method lies in its ability to handle qualitative predictor variables (up to a dummy coding) and incomplete data in both the training samples and future observations to be predicted/ranked.

A **b**ootstrap **agg**regat**ing** method can also be implemented in order to enhance ranking accuracy and stability, refer to [2] for further details.

Throughout this note, the installation and the use of the TreeRank package for *R* statistical software is described.

2 TreeRank package Contents

The TreeRank package implements under the R environment the TreeRank algorithm and provides tools to analyze the results, as well as a graphical user interface for most of the functionalities. It essentially includes the following procedures:

- the TreeRank growing procedure, which produces from the training dataset a complete oriented rooted binary tree, the *Master Ranking Tree*;
- 3 LeafRank versions for the internal splitting rules of the Master Tree : one based on a cost-sensitive version of the CART algorithm, one based on a cost-sensitive random forest algorithm, and one based on cost-sensitive SVM's;
- a TreeRank based algorithm for the statistical problem of testing homogeneity of two samples in a multidimensional setup.
- bagging and pruning tools for the TreeRank algorithm.

It also comprises the tools listed below:

- score computation tools to ranking predictions from ranking trees and new (unlabeled) data;
- tools for computing and displaying ROC and precision/recall curves from ranking trees and (traning and test) data;
- tree manipulation tools, in order to extract sub-ranking trees or to combine ranking trees;
- model interpretation tools, such as variable importance computation;
- graphical interfaces for launching the procedures and for displaying/exploring the results (tree graphics, *etc.*).

The package also includes demo datasets as well as documentation files. The names and characteristics of the demo datasets are collected in the following table:

Data set name	Pyr2D	Gauss2D	Gauss20DFar	Gauss20DClose
Nature	Artificial uniforms	Artificial gaussians	Artificial gaussians	Artificial gaussians
# Attributes	2	2	20	20
Learning sample size	2000	2000	2000	2000
Test sample size	1000	1000	1000	1000
Positives rate	0.5	0.5	0.5	0.5

For each data set, the variable `Name.learn` denotes the learning set, the variable `Name.test` the test set and the variable `Name.roc` the target ROC curve (*i.e.* the optimal one). For all data set, the name of the label attribute is `class` and, by convention, we always take the value 1 as "positive" label.

3 Installation

This section describes the installation steps of the TreeRank package, it is intended for beginning R users.

3.1 Getting R statistical software

R is a language and an environment for statistical computing and graphics. It is available for free under the terms of the GNU General Public Licence at http://www.r-project.org/ for Windows, MacOS and UNIX platforms. Tutorials and documentations are also available on the R homepage.

3.2 TreeRank requirements

The TreeRank package is fully implemented in R, thus no third-party software is required for its basic use. It is however based on other R packages from the official CRAN package repository and some of them need `Tcl/Tk version 8.4` or above for graphical purposes[1]. The Windows version of R installs by default all needed files for `Tcl/Tk`. For UNIX and MacOS R version, `Tcl/Tk` needs to be installed by hand. A build for MacOS is available at http://cran.r-project.org/bin/macosx/tools/ (just install the `dmg` file). For UNIX, please refer to your distribution to find and install the appropriate packages (generally `tcl8.x` and `tk8.x`).

The R packages required for the installation of TreeRank are the following:

- `rpart`, providing a CART implementation for recursive partitioning and regression trees;
- `kernlab`, kernel-based machine learning methods including a SVM algorithm;
- `randomForest`, a random forest algorithm;
- `coin`, conditional inference procedures including two sample problems.

The recommended packages are the following (only used for the graphical user interfaces):

- `tkrplot` for placing R graphics in a `Tk` widget;
- `colorspace` for some nice color palettes;
- `igraph`, a very complete package for graphs and network analysis, used here for visualization purpose only.

Normally, no manual installation of these packages is required, it is done automatically when installing the TreeRank package. In case of troubles, a manual installation can be done through the command `install.packages("NameOfPackage")` in the R shell.

3.3 Getting the TreeRank package

The last version of TreeRank package is available at http://treerank.sourceforge.net/. However, the easiest way to install it is by the R command `install.packages`. To do that, open an R session, and just type in the shell `install.packages("TreeRank")`. A window appears asking you to choose a CRAN repository. To check your installation, try to load the package by the command `library(TreeRank)` and then execute the command `example(TreeRank)`.

[1] If you plan to use only the command line version of TreeRank, there is no need for `Tcl/Tk`. The installation of the graphic related packages are recommended but not required.

4 Loading data

The input used by TreeRank to construct a ranking tree (and thus a scoring function) is a set of labeled examples (x_i, y_i), the input information being described by $q \geq 1$ numerical or categorical attributes, while the output consists of a binary label (conventionally but not mandatory $+1$ for positive examples and -1 for negative ones: "relevant" *vs.* "irrelevant", "sick" *vs.* "healthy", *etc.*). In R, *data frames* are used to represent this type of data. Actually, a data frame is a sort of matrix, where columns can correspond to different type of data. Each column of a data frame has an unique name denoting a variable and each row represents an example. Generally, the label is included in the data frame representation as an additional variable.

In order to import data from files to data frames, R has several functionalities, that allows for dealing with a wide variety of data files. We will discuss further how to import the most common type of data file, the spreadsheet-like text file, in which the data are presented in a rectangular grid, possibly with row and column labels (like `cvs` and MATLAB files; see R documentation to import other file formats). For such files, each line of the file describes an example and each value is separated by a special character named *separator* (usually the space, the tabular or the semi-colon character). These data files can be imported in R by the means of the generic function `read.table`. The following parameters must be specified in order to use this function :

- `file` : the path and filename of the file being imported;

- `sep` : the separator used in the file;

- `headers`: to be set to `TRUE` or `FALSE`, indicating whether the names of the variables are included in the file (corresponding in general to the first line of the file) or not. When the names are not included (the most common case), they can be specified by the optional parameter `col.names`. Otherwise, R uses by default the name Vi for the i-th column.

The R affectation symbol `<-` is used to store the result of the importing procedure.

```
Demo:
# The separator for myData1.cvs is the colon character
# and the file don't contain the variable names.
> table1 <- read.table(file="myData1.cvs", sep=",", headers=FALSE
> table1[1:3,]           # Display the 3 first rows of the data frame
      V1         V2       V3      V4        V5        V6        V7       V8
1  1.18590 -0.0024941  1.33290 1.69090  0.908180 -0.658880  0.325710 0.698940
2  0.87946 -0.5272500  0.12230 1.76700  0.772090  1.242500 -0.060701 2.544200
3  2.09470  0.8184400  1.64860 1.19230 -1.153100 -1.196300 -1.136800 0.600780

> nrow(table1)           # Display the number of rows
[1] 3000
> ncol(table1)           # Display the number of columns
[1] 8
> colnames(table1)       # Display the column names
[1] "V1"  "V2"  "V3"  "V4"  "V5"  "V6"  "V7"  "V8"

# import myData1.cvs  with specified variable names.
>table2 <- read.table(file="myData1.cvs",sep=",",headers = FALSE,
   col.names = c("a","b","c","d","e","f","g","label"))
>table2[1:3,]
       a          b      c      d        e        f         g    label
1 1.18590 -0.0024941 1.3329 1.6909  0.90818 -0.65888  0.325710 0.69894
2 0.87946 -0.5272500 0.1223 1.7670  0.77209  1.24250 -0.060701 2.54420
3 2.09470  0.8184400 1.6486 1.1923 -1.15310 -1.19630 -1.136800 0.60078
```

5 Using TreeRank through the graphical interface

The TreeRank package provides a user-friendly Graphical User Interface (GUI) handling most of the TreeRank features. This section presents how to use it.

The GUI is made up of two interfaces. The first one, described in section 5.1, is used to configure the TreeRank procedure and the dataset used. The second one (section 5.2) allows the user to display/explore the results and to perform model selection (either automatically or else manually).

In the following, it is assumed that the TreeRank package and all of its requirements have been preliminarily installed (see Section 1). The `Gauss2DEasy` toy dataset, included in the package, is used for the demo.

To load the library, execute the following command:

```
Demo:
>library(TreeRank)
```

5.1 The Launching Interface

The `TRGui()` command is used to start the interface. After its execution, a window like the one displayed in Fig. 5.1 normally appears. This window is made up of three frames: data setting, LeafRank options and TreeRank options.

Demo:
```
>TRGui()
```

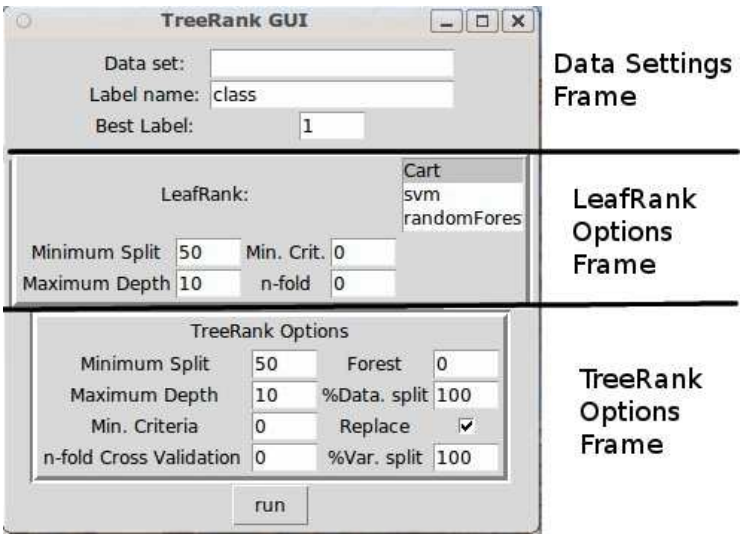

Figure 1: The Launching Interface

5.1.1 Data setting frame

This frame is used to set up the learning dataset. The dataset has to be loaded previously in R as a data frame (see section 4). The field *Data Set* is used to specify the name of the R variable containing the dataset. The field *Label name* indicates the name of the variable in the data frame denoting the class/label/response of the examples. The field *Best Label* indicates which label value corresponds to the "positive" label.

Demo:
Set

- *Data set* to `Gauss2DEasy.learn`
- *Label name* to `class`
- *Best Label* to `1`.

5.1.2 LeafRank options frame

This frame allows to choose the LeafRank algorithm to be used and pick its tuning parameters. The listbox is used to select the LeafRank algorithm (three LeafRank algorithms are provided natively, a CART-based, a SVM-based, and a RANDOM FOREST-based version). The options depend on the used LeafRank algorithm, setting up the learning parameters of the chosen algorithm.

CART options

- MINIMUM SPLIT : the minimum number of observations that must exist in a node, in order for a split to be attempted;
- MAXIMUM DEPTH : set the maximum depth of any node of the final tree, with the root node counted as depth 0.
- MIN. CRIT. : if the criterion (in our implementation the auc) is less than this value, the node is not splitted.
- N-FOLD : number of cases for the n-fold cross pruning. If it is set to 0 or 1, no pruning is done.

SVM options Full documentation for these options can be found in the package KERNLAB.

- C : cost of constraints violation;
- DEGREE/SIGMA : set the value of sigma or degree kernel parameters, depending on the chosen kernel.
- SCALE, OFFSET : set the value of scale and offset parameter for the hyperbolic tangent kernel.
- KERNEL : set the kernel to be used : RBF is a radial basis kernel Gaussian, POLY. a polynomial kernel, TANHDOT a hyperbolic tangent kernel.
- AUTO. PARAMETERS : automatically compute the best kernel parameters.

random Forest options Full documentation for these options can be found in the package RANDOMFOREST.

- #TREE : number of trees to grow.
- #VAR : number of variables randomly sampled as candidates at each split. If it is set to 0, all the variables are used.
- %DATA : percentage of the examples to be used for each tree computation. The examples are drawn randomly from the whole data set.
- REPLACE : if it is checked, the sampling of the examples is done with replacement.
- NODE SIZE : Minimum size of terminal nodes.
- MAX. LEAVES : Maximum number of terminal nodes trees in the forest can have.

Demo:
Select the CART algorithm and keep default options.

5.1.3 TreeRank options frame

The following options set controls the tree growing stage.

- *Minimum Split*: the minimum number of observations that must exist in a node, in order for a split to be attempted.
- *Maximum Depth*: the maximum depth of the tree.
- *Min. crit.*: if the AUC increase is below this threshold, the node is not split and it becomes a leaf.

The *n-fold Cross Validation* option controls the tree pruning stage. If its value n is greater than 1, the tree will be pruned by a n-fold cross validation.

The *forest* option allow to compute a forest of TreeRank rankers. If its value is greater than one, its indicates the number of TreeRank rankers to learn to compute the forest.

The *%Var. Split* option controls the amount of *feature randomization*: it indicates the percent of variables used at each node of the master tree (drawn randomly). The *%Data Split* option indicates the percent of examples of the training dataset used for the learning in case of forest computation. The checkbox *replace* indicates when the sampling is done with replacement or not.

Demo:
Set *%Data Split* option of LeafRank frame to 80% and keep the default options for the others and click the run button.

5.2 The results explorer interface

After the TreeRank computation, a new window is displayed (figure 5.2).

5.2.1 Information displayed

The central frame `#1` displays the ranking tree. Each node is colored according to the percentage of positive examples lying in the node (green for higher percentage, red for lower). The left child of a node is always the best scored node and the right child the worst one. Thus, the leaves are ordered from the best score (on the left) to the worst (on the right).

Selecting a node (by clicking on it) displays the node information at the upper left corner of the window (frame `#2`): the node Id, the number of positive/negative examples belonging to the node, the AUC increase due to the node split and the score if the node is a leaf. Variable importance measures are given on the left side listbox (frame `#3` and displayed as an histogram at bottom right (frame `#7`).

Finally, the training ROC curve is displayed on the top right graph, at the frame `#6`.

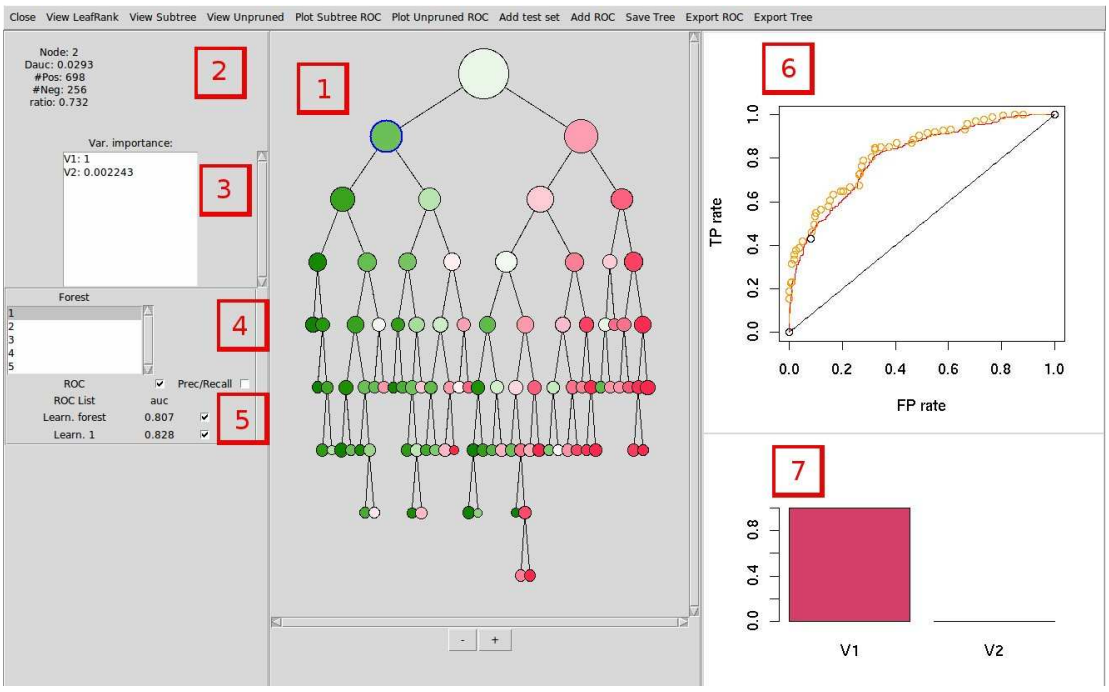

Figure 2: Results explorer interface.

If the current object is a TreeRank forest, the frame `#4` allows to navigate through the different trees. In this case, the plotted tree in the main frame is the selected tree (by default the first one) and the frames `#6` and `#7` refer to this tree.

5.2.2 Interactive tools

As specified above, clicking a node selects the node. Selecting multiple nodes can be done by pressing the *control* key when the click is performed. With a right click on the central frame (containing the tree), a popup menu is displayed. The menu items are also available on the top menu of the window. Some items are specific to nodes, in which case they appear in the popup menu only by clicking on a node. The *View LeafRank* item allows the user to display, on a new window, the LeafRank classifier associated to a given node, but only when such an operation is allowed (depending on the chosen LeafRank algorithm). See section 5.2.3 for more information about the CART LeafRank classifier.

It is possible to consider a subtree of the tree by selecting some nodes. The subtree is constructed by pruning the tree at the selected nodes. The *View Subtree* item opens a new interface corresponding to the selected subtree. The *Plot Subtree* ROC item plots the training ROC curve corresponding to the subtree.

In the case of the pruning option is selected for the learning, the *View Unpruned* item opens a new interface with the unpruned tree. The *Plot Unpruned* ROC item plots the training ROC curve, corresponding to the unpruned ranking tree. The *Add test set* item allows to plot a test ROC curve. The test set has to be in a R data frame variable. Selecting the item opens a dialog box, where the name of the test set variable can be submitted. Hiding/displaying a ROC curve can be done by unchecking/checking the corresponding checkbox on the left side of the window, in the frame `#5`.

The precision/recall curves can be displayed by checking/unchecking the corresponding checkbox on the top of the frame `#5`.

Finally, the *Save tree* item allows the user to export the tree in a R variable; the *Export ROC* item allows the user to save in a *eps* file the frame `#6`, i.e. the ROC curve; the *Export Tree* item allows the user to save in a *eps* file the main frame, i.e. the tree.

5.2.3 CART LeafRank interface

In the case where the CART LeafRank algorithm is used, the interface displayed when selecting the *View LeafRank* item is very similar to the TreeRank interface, but simplified. The tree displayed corresponds to the CART tree. The nodes are not ordered. Finally, each leaf has a green or a red circle, which indicates if the examples reaching this node are send to the left or the right child of the TreeRank tree.

5.3 A simple session example

In this section we show how to use TreeRank package to tune the procedures, to compute various models and compare the results. We assume that the library is loaded and the interface is launched. We will use the `Gauss2ODFar.learn` dataset. The label name of this dataset is `class` and the "best label" is `1`.

To compute the first model, leave all options by default and run the TreeRank procedure. In the resulting interface, the real ROC curve can be displayed by choosing `Add ROC` menu item and filling the textbox with `Gauss2ODFar.roc`. As we can see, the train ROC is similar to the real ROC. To display the test ROC, choose the `Add test set` item and fill it with `Gauss2ODFar.test`. As expected, the resulting ROC is worst than the previous one. To improve the result, a TreeRank forest can be used : in the launcher, choose to compute 10 trees in the forest, and set 80% for data split and var split option. The AUC of the test ROC is thus improve by 0.05.

As the problem to be treated is a gaussian one, we can expect better results using the svm-based LeafRank procedure : in the launcher, choose the svm version for the LeafRank algorithm, run the procedure and compute the test ROC. The resulting test ROC have an AUC near to 0.8 and for the forest version with 10 trees near to the optimal one, 0.84. In comparison, a polynomial kernel outputs poor results.

Finally, a way to improve Cart-based LeafRank is to use the Random Forest-based LeafRank procedure. This version has to be tuned carefully : if the number of tree is too high, a random forest will match perfectly the training dataset and the TreeRank master tree will have a depth of only 1, as the first forest will classify perfectly the training dataset. One can tune the `node size`, `#tree` and the `Max leaves` parameters to prevent this effect. For instances, run the launcher with 10 as `node size` and 200 trees for the LeafRank options and 10 trees for the TreeRank option. The resulting AUC of the test ROC is near to 0.7, which is a great improvement compared to the Cart version.

6 Advanced use of TreeRank

We present in this section the console use of the TreeRank package. The full details can be found in the man page documentation of the TreeRank package.

6.1 Main procedures

- `TreeRank` : the main function to compute a TreeRank scoring function.
- `TreeRankForest` : the forest version of the TreeRank algorithm.
- `LRCart` : the CART-based LeafRank procedure.
- `LRsvm` : the svm-based LeafRank procedure.
- `LRforest` : the forest-based LeafRank procedure.
- `TRplot`: launching the interface for a TreeRank or LRCart object.
- `predict`: a generic method for predicting scores from a TreeRank tree and (unlabelled) data.
- `getClassifier`: return the classifier associated to a specified node from a TreeRank tree. The generic method predict can be used on the returned object.
- `getROC`/`getPREC`: return a matrix with the ROC curve (or precision/recall curve) coordinates from a TreeRank tree and data.
- `auc`: return the AUC of the ROC curve from a ROC curve.
- `varImportance` : return a vector containing the variable importance measure for each variable.
- `depVar` : return the partial dependence on pair of variables.

6.2 A simple session example

We use the same example as in the 5.3 section to illustrate the use of the TreeRank package with the console mode. To compute a TreeRank model with the default parameters for the `Gauss2DFar` dataset, execute the following command :
`treeCart <- TreeRank(formula=class~ .,data=Gauss20DFar.learn,bestresponse=1);`.
The computed tree is stored in the variable `treeCart`. To display the computed tree, execute `TRplot(treeCart)`. To predict scores for a new dataset, for instances `Gauss2DFar.test`, the command is the following :
`predict(treeCart,Gauss20DFar.test)`.
The train ROC curve of the model can be computed by the command :
`rocCartTrain <- getROC(treeCart,Gauss20DFar.learn)`
and the test ROC curve by :
`rocCartTest <- getROC(treeCart,Gauss20DFar.test)`.

The ROC curves can be plotted either by the command `plot(rocCartTest)` or :
`plotROC(list(rocCartTrain,rocCartTest))`.
The AUC of the test ROC can be computed by `aucCartTest <- auc(rocCartTest)`. The variable importance measures are computed by the command `varImportance(treeCart)`. Finally, the partial dependence on pair of variables can be computed by the command :
`vdCart <- varDep(treeCart,Gauss20DFar.test,varx="V5",vary="V10",subdivx=10)` (here on

the pair $(V5, V10)$). The plot can be drawed using the `persp` command : `persp(vdCart)` (or to compute a heatmap : `heatmap(vdCart)`).

In the following we will give examples to parametrize the TreeRank procedure. By default, the TreeRank procedure uses the Cart-based LeafRank procedure. In order to compute a svm-based version, the TreeRank option `LeafRank` has to be specified. The call is the following : `TreeRank(class~ ., Gauss20DFar.learn, 1, LeafRank= LRsvm)`. The customization of the `LRsvm` can be inline in the call or by defining a new procedure with the wanted parameters :

Demo:
```
#Inline :
TreeRank(class~ .,Gauss20DFar.learn,1,LeafRank=
function(...)LRsvm(kernel="polydot", C=2,...)
#By defining a new function
MySvm <- function(...)LRsvm(kernel="polydot", C=2,...)
TreeRank(class~ .,Gauss20DFar.learn,1,LeafRank=MySvm)
```

The following commands define a custom Random Forest LeafRank and compute a TreeRank forest with the same options as in section 5.3 :

Demo:
```
MyRandom <- function(...) LRforest(nodesize=10,ntree=200,...)
TreeRankForest(class~ .,Gauss20DFar.learn,1, LeafRank=MyRandom,
varsplit=0.8,sampsize=0.8,ntree=10)
```

The option details can be found in the manual pages of each function (`ksvm` for the svm-based LeafRank, `randomForest` for the Random Forest version).

7 The two sample problem

The TreeRank package implements a TreeRank version of the two sample problem, testing the homogeneity of two samples in a multidimensional setup([4]). To launch the graphical interface, execute the following command:

Demo:
```
>TwoSampleGui()
```

The only difference between this interface and the TreeRank launcher one is for the upper frame, the data setting frame. The field *1st data set* is used to set one of the two sample to be tested; the other one can be set by the field *2nd data set*. These datasets have to be dataframes (see section 4).

The *Learning % size* field indicates the percentage of the both data set to use for the training. The remaining data are used to compute the Mann-Whitney Wilcoxon test. The *confidence level* field denotes the confidence level of the test. The Mann-Whitney Wilcoxon test is computed with the functions of the *coin* package.

After the computation, the learned tree is displayed with the interface used by TreeRank. The results of the test are shown in the console.

References

[1] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, 1984.

[2] S. Clémençon. Ranking Forests. Technical Report hal-00452577, HAL, 2010.

[3] S. Clémençon, M. Depecker, and N. Vayatis. Adaptive partitioning schemes for bipartite ranking. Technical Report hal-00268068, HAL, 2009.

[4] S. Clémençon, M. Depecker, and N. Vayatis. AUC optimization and the two-sample problem. In *Advances in Neural Information Processing Systems*, 2009.

[5] S. Clémençon and N. Vayatis. Tree-structured ranking rules and approximation of the optimal ROC curve. In *Proceedings of ALT 2008*. Springer, 2008.

[6] S. Clémençon and N. Vayatis. Tree-based ranking methods. *IEEE Transactions on Information Theory*, 55(9):4316–4336, 2009.