Implementation of a classification tree-growing algorithm in R

João Paulo Pizani Flor, Tomaš Kadlec
Department of Information and Computing Sciences,
Utrecht University - The Netherlands
e-mail: j.p.pizaniflor@students.uu.nl

Friday 11th October, 2013

1 Introduction

This report describes the solution to the first practical assignment of the master course "Data Mining" at Utrecht University, taught in the 1st period of the academic year 2013/2014.

2 Organization of the distributed package

The handed-in solution is organized as follows:

R workspace The R workspace image containing both the *code* and the *datasets* used for analysis is under the package root with name "`RData`"

R Source The source code of the functions implemented is located in the package root with the name "`classificationTree.R`". This file is just a *concatenation* of the files "`treeBuild.R`" and "`treeClassify.R`". In a running R session, the code can be loaded by issuing the command `source("classificationTree.R")`.

Datasets The datasets available for usage are available under the "`datasets`" subdirectory, in CSV format.

3 Requirements on the data to be analyzed

According to what was described in the assignment specification, some assumptions are made on the inputs to the algorithm. If some of the following requirements is NOT met, then the program will display **undefined behaviour**. On the other hand, if the input data meets the following requirements, then the algorithm complies with the specification (to the extent of our knowledge)

- The dataset to be analyzed (excluding the classes) should be stored in R as a **dataframe**, i.e, not as a matrix or vector.
- The **class** parameter is a **vector of integer** values, in which each element corresponds to the class of an observation in the dataset.
- Both parameters `nmin` and `minleaf` to the `tree.grow` function should be **lesser than or equal** to the number of observations, i.e.

$$(N_{\text{min}} \leq \text{Ob}) \wedge (\text{MINLeaf} \leq \text{Ob})$$

It's important to notice that in our implementation, the class attribute **does not** need to be binary. The calculation of impurity using the gini index is working in the general case of (integer) categorical values.

4 Training and classification process

We tested our algorithms using two datasets: the "Pima Indians Diabetes Database", provided in the Data Mining course webpage, and the "Adult Breast Cancer Database", obtained from the online UCI Machine Learning repository[1].

The "Pima Indians Diabetes" dataset complied with the requirements of our algorithm, as exposed in section 3. The "Adult Breast Cancer Database" has 16 incompletely recorded cases, i.e. some of the attributes are undefined. We have therefore removed these records to meet the input requirements. Additionaly, the Breast Cancer dataset has a class attribute with a non-standard encoding: instead of using the values 0 and 1, they use 2 and 4. This was not a problem, though, because our impurity calculation uses the general case of the gini index.

It should be noted that the impurity function to be used can be passed **as a parameter** to the `tree.grow` function. We have implemented the **generalized** gini index (`impurity.gini` which is used by default. However, as of R 3.0 the built-in `table` function is notably slow and can use as much as 95% of cpu time when the tree is being constructed. For binary clas attributes the considerably faster version `impurity.gini.bin` can be used.

4.1 Testing the accuracy of the built classification trees

Having chosen the datasets on which to work, we proceeded to growing classification trees with a subset of each dataset, and then using the built tree to classify the remaining data points. Our experimental setting was as follows:

Performance measure We use the proportion of correct guesses of a class value (probability of a correct guess) as measure of the performance of the classification tree. This value, of course, is expressed as a function of the arguments `nmin` and `minleaf`

Size of training set For each of the two datasets used, we established a partition of 70% of the data points used for training and 30% for classification.

Choice of the training set The number of data points to be used for training is sampled randomly from the dataset.

Number of tests performed As a random choice is being performed, we chose to run 10 rounds of this "performance test" over each dataset, giving more confidence in the results obtained.

As the performance of the classification is a function of two arguments (`nmin` and `minleaf`), the plot for this function is 3-dimensional. One example of performance plot (the one for the "PIMA" dataset) can be seen in Figure 4.1. The performance for the other dataset (the "Adult Breast Cancer" dataset), can be visualized in Figure 4.1.

References

[1] K. Bache and M. Lichman. UCI machine learning repository, 2013.

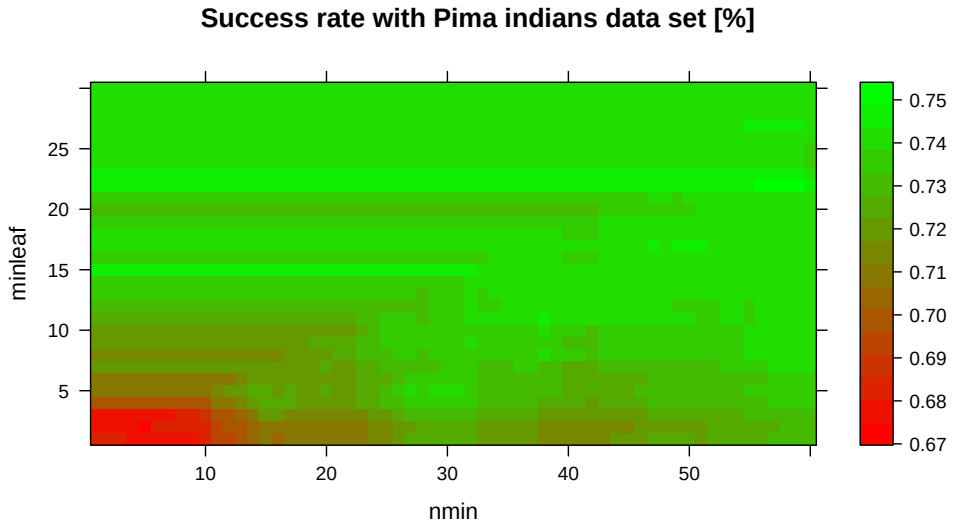

Figure 1: Performance plot for the PIMA dataset

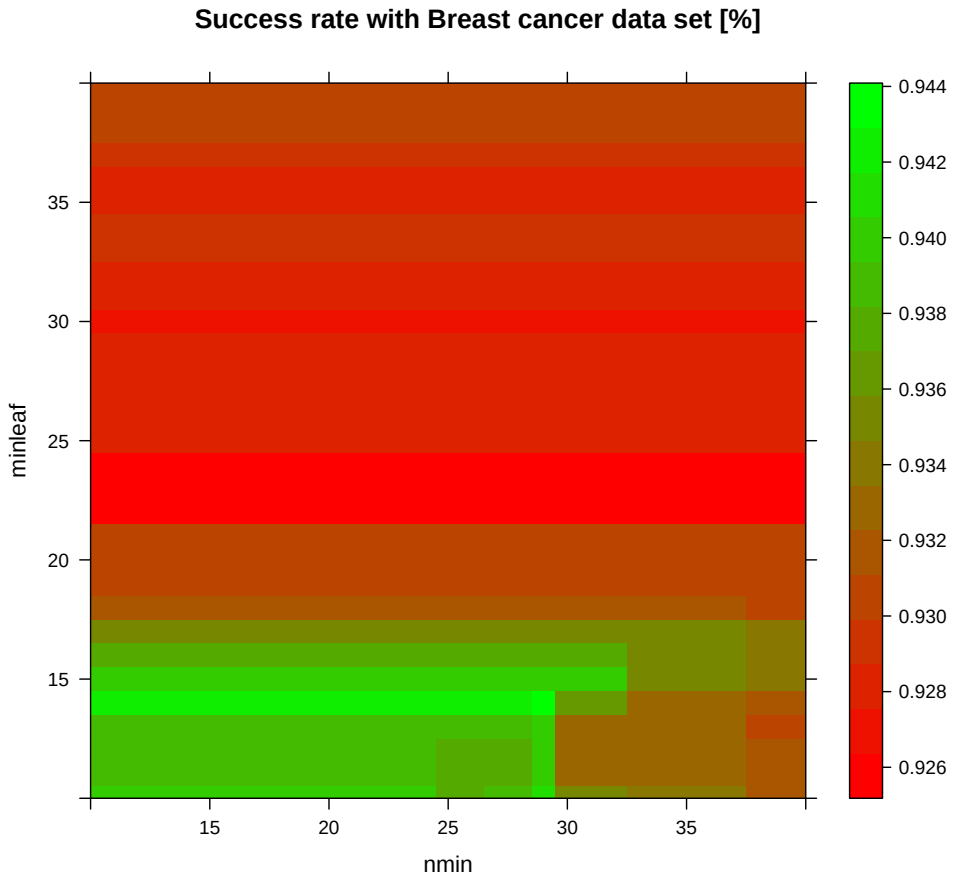

Figure 2: Performance plot for the cancer dataset