An introduction to ffcr

Marcus Buckmann

2021-01-31

Introduction

This document gives an introduction on the functionality of the *ffcr* package. This package allows the construction of two families of transparent classification models: *fast-and-frugal trees* and *tallying* models. A fast-and-frugal tree is a decision tree with a simple structure: one branch of each node exits tree, the other continues to the next node until the final node is reached. A tallying model gives pieces of evidence the same weight. The package contains two main functions: `fftree` to train fast-and-frugal trees and `tally` to train tallying models.

To illustrate the functionality of the package, we use the *Liver* data set (Ramana, Babu, and Venkateswarlu 2011) that we obtained form the UCI machine learning repository (Dua and Graff 2017). It contains 579 patients of which 414 have a liver the condition and the other 165 do not. We predict which patient has a liver condition using medical measures and the age and gender of the people.

We start with loading the package and the data.

```
library(ffcr)
data(liver)
```

Learning fast-and-frugal trees

The `fftr` functions encompasses three different methods to train fast-and-frugal trees. These are named *basic*, *greedy*, and *best-fit*[1] and are described in the book.

Training a fast-and-frugal tree

We train or first fast-and-frugal tree on the Liver data set. If the first column in the data set is the class label, we can simply pass the data set as the first argument.

```
model <- fftree(liver, use_features_once = FALSE, method = "greedy", max_depth = 6)
```

Alternatively, we can call the function using the formula syntax. Here we train the fast-and-frugal tree using only only a few selected features.

```
fftree(diagnosis ~ sex + age  + albumin + proteins + aspartate  , data = liver)
```

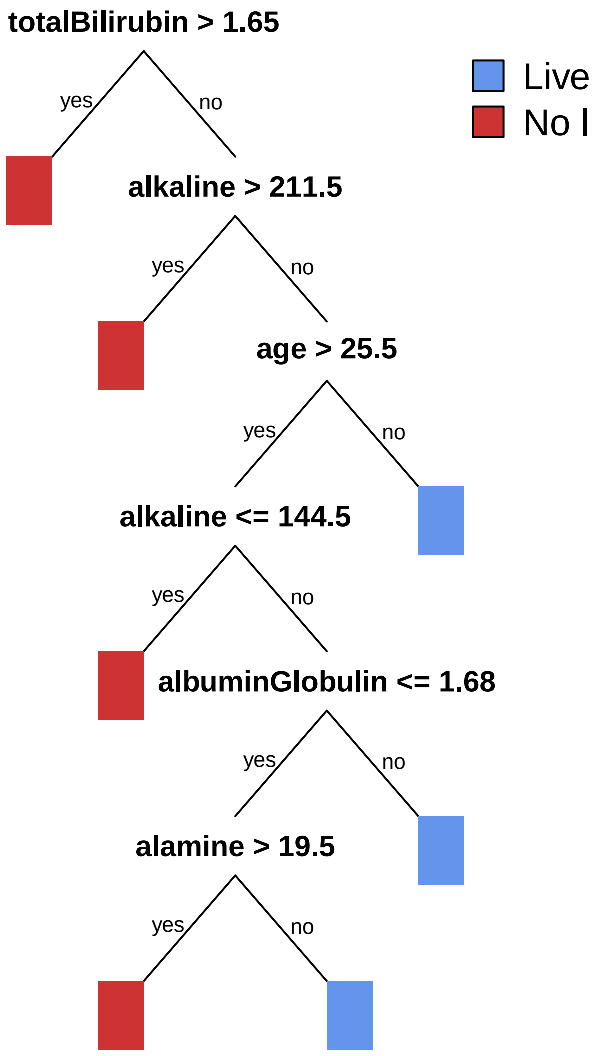
[1]We use the cross-entropy method (Rubinstein 1999). It does not guarantee to find the best possible tree but produces very accurate trees, on average.

The model object shows the structure of the trees and its performance on the data set.

```
print(model)
#> Fast-and-frugal Tree object
#>   type: "recursive"
#> 
#> Call:
#> fftree(data = data, method = "greedy", max_depth = 6, use_features_once = FALSE,
#>     formula = formula)
#> 
#> Formula:
#> diagnosis ~ age + sex + totalBilirubin + directBilirubin + alkaline +
#>     alamine + aspartate + proteins + albumin + albuminGlobulin
#> 
#> Tree:
#> 
#>  Reason: Predicted class - (Proportion of class Liver disease) (Number of objects classified)
#> 
#>   totalBilirubin > 1.65: Liver disease (1.00) (213)
#>     alkaline > 211.5: Liver disease (1.00) (132)
#>       age <= 25.5: No liver disease (0.00) (27)
#>         alkaline <= 144.5: Liver disease (1.00) (27)
#>           albuminGlobulin > 1.68: No liver disease (0.00) (4)
#>             alamine <= 19.5: No liver disease (0.00) (38)
#>               alamine > 19.5: Liver disease (1.00) (138)
#> 
#> 
#> Fitted values:
#>               Observed       Predicted   N
#>          Liver disease    Liver disease   0
#>       No liver disease    Liver disease 510
#>          Liver disease No liver disease   0
#>       No liver disease No liver disease  69
#> 
#> Fitting:
#>     AUC                   NA
#>     Accuracy            0.12
#>     Sensitivity          NaN
#>     Specificity         0.12
#>     Balanced accuracy    NaN
#>     F1 score            0.00
#> 
#>     Depth      6.00
#>     Features   5.00
#>     Frugality  3.01
```

To visualize the tree we use

```
plot(model)
```



How does the fast-and-frugal tree perform in cross-validation? By default the model is fitted to the complete data set but if we set ‘cv = TRUE’, 10-fold cross-validation is used to estimate the predictive performance of the tree. The model saved in the object is fitted on the complete training set. In fitting and prediction, the sensitivity is very high, while the specificity is low. The majority of the patients (71%) have liver disease, therefore predicting liver disease for most objects will produce a highly accurate tree. To avoid that, we can weigh the objects such that the objects in both classes get the same share. Let *p* be the proportion of patients that have liver disease. We weigh the patients with liver disease by 1-p, and the patients without disease by p.

Note how sensitivity and specificity are more similar now:

```
p <- sum(liver$diagnosis == "Liver disease")/nrow(liver)
model <- fftree(liver, weights = c(1-p,p), cv = TRUE)
model
#> Fast-and-frugal Tree object
#>   type: "recursive"
#>
```

```
#> Call:
#> fftree(data = data, weights = c(1 - p, p), cv = TRUE, formula = formula)
#>
#> Formula:
#> diagnosis ~ age + sex + totalBilirubin + directBilirubin + alkaline +
#>     alamine + aspartate + proteins + albumin + albuminGlobulin
#>
#> Tree:
#>
#>  Reason: Predicted class - (Proportion of class Liver disease) (Number of objects classified)
#>
#>   directBilirubin > 1.05: Liver disease (1.00) (165)
#>     alamine > 66.5: Liver disease (1.00) (61)
#>       alkaline <= 211.5: No liver disease (0.00) (242)
#>         age <= 27.5: No liver disease (0.00) (26)
#>           proteins > 7.45: Liver disease (1.00) (15)
#>             sex = Female:  (0.00) (12)
#>               sex = :  (1.00) (58)
#>
#>
#> Fitted values:
#>             Observed       Predicted    N
#>        Liver disease    Liver disease   0
#>     No liver disease    Liver disease 299
#>        Liver disease No liver disease   0
#>     No liver disease No liver disease 280
#>
#> Fitting:
#>     AUC                    NA
#>     Accuracy             0.48
#>     Sensitivity           NaN
#>     Specificity          0.48
#>     Balanced accuracy     NaN
#>     F1 score             0.00
#>
#>     Depth     6.00
#>     Features  6.00
#>     Frugality 2.78
#>
#> Cross-validation:
#>     AUC                  0.66
#>     Accuracy             0.61
#>     Sensitivity          0.57
#>     Specificity          0.73
#>     Balanced accuracy    0.65
#>     F1 score             0.68
#>
#>     Depth     5.80
#>     Features  5.80
#>     Frugality 2.84
```

To make predictions according to a fast-and-frugal tree, we can use the `predict` function. It returns either the class label (*response*), the predicted probability of belonging to one of the classes (*probability*) or the performance across the observations (*metric*). Note that for the latter, the class labels need to be included in the data that is passed to the predict function.

```r
model <- fftree(diagnosis ~ ., data = liver[1:300,], weights = c(1-p,p))

predict(model, newdata = liver[301:310,], type = "response")
#>  [1] No liver disease No liver disease Liver disease    Liver disease   
#>  [5] No liver disease Liver disease    Liver disease    No liver disease
#>  [9] Liver disease    No liver disease
#> Levels: Liver disease No liver disease

predict(model, newdata = liver[301:310,], type = "probability")
#>       No liver disease Liver disease
#>  [1,]                1             0
#>  [2,]                1             0
#>  [3,]                0             1
#>  [4,]                0             1
#>  [5,]                1             0
#>  [6,]                0             1
#>  [7,]                0             1
#>  [8,]                1             0
#>  [9,]                0             1
#> [10,]                1             0

predict(model, newdata = liver[301:nrow(liver),], type = "metric")
#>               AUC          Accuracy       Sensitivity       Specificity 
#>         0.6327268         0.6702509         0.7305699         0.5348837 
#> Balanced accuracy          F1 score    True positives   False positives 
#>         0.6327268         0.7540107       141.0000000        40.0000000 
#>    True negatives   False negatives 
#>        46.0000000        52.0000000
```

Learning tallying models

The package implements two different methods to train tallying models, which are also explained in the book. These are named *basic* and *best-fit*.[2]

```r
p <- sum(liver$diagnosis == "Liver disease")/nrow(liver)
model <- tally(diagnosis ~ ., data = liver[1:300,], weights = c(1-p,p), max_size = 6)

predict(model, newdata = liver[301:nrow(liver),], type = "metric")
#>               AUC          Accuracy       Sensitivity       Specificity 
#>         0.7122545         0.6559140         0.6010363         0.7790698 
#> Balanced accuracy          F1 score    True positives   False positives 
#>         0.6900530         0.7073171       116.0000000        19.0000000 
#>    True negatives   False negatives 
#>        67.0000000        77.0000000
```

[2]We again use the cross-entropy method.

References

Dua, Dheeru, and Casey Graff. 2017. "UCI Machine Learning Repository." University of California, Irvine, School of Information; Computer Sciences. http://archive.ics.uci.edu/ml.

Ramana, Bendi Venkata, M Surendra Prasad Babu, and NB Venkateswarlu. 2011. "A Critical Study of Selected Classification Algorithms for Liver Disease Diagnosis." *International Journal of Database Management Systems* 3 (2): 101–14.

Rubinstein, Reuven. 1999. "The Cross-Entropy Method for Combinatorial and Continuous Optimization." *Methodology and Computing in Applied Probability* 1 (2): 127–90.