Tree Classification

Single classification tree

Differently from logistic regression, trees are non-parametric methods which can be used both for classification and regression purposes. From a geometrical point of view, a tree is a predictor that *stratifies* the predictors space into hyperboxes. In case of classification, all the points falling in a certain box are assigned the same label, which is the one occurring most frequently in the training set in that box (or *region*).

Especially in the regression case, trees are not competitive in terms of accuracy with respect to other methods, like linear regression. Trees work better in the classification case, and they have the big advantage of being interpretable, differently from the logistic regression setting, where meaning of the coefficients is not very intuitive.

After splitting the dataset into a training (70%) and validation (30%) set with the same procedure as in the logistic regression, we build the tree using observations from the training set. A summary of the performance of the tree over the training set is shown below.

```
##
## Classification tree:
## tree(formula = Exited ~ ., data = train_set)
## Variables actually used in tree construction:
## [1] "Age"            "NumOfProducts" "IsActiveMember"
## Number of terminal nodes:  8
## Residual mean deviance:  0.7588 = 5306 / 6992
## Misclassification error rate: 0.1506 = 1054 / 7000
```

First of all, we can notice that only three out of ten variables were used by the tree. Moreover, the resulting tree is quite small: it presents just eight terminal nodes, or *leaves*. Despite the small structure, the tree was able to correctly classify about 85% of the examples in the training set, resulting in a misclassification rate of 15%. We now take a closer look at the tree, and more precisely we consider the distribution of the response in the terminal nodes, indicated by the symbol * at the end of the row.

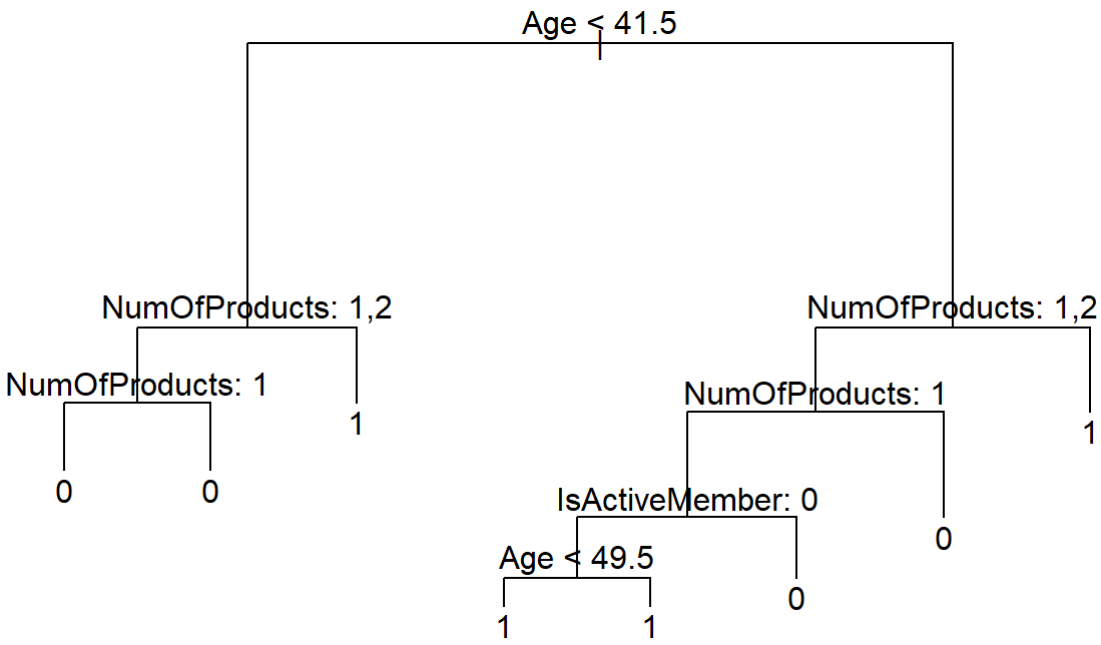
```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 7000 7080.00 0 ( 0.79614 0.20386 )
##    2) Age < 41.5 4736 3328.00 0 ( 0.88767 0.11233 )
##      4) NumOfProducts: 1,2 4648 3036.00 0 ( 0.89931 0.10069 )
##        8) NumOfProducts: 1 2307 2008.00 0 ( 0.84265 0.15735 ) *
##        9) NumOfProducts: 2 2341  857.10 0 ( 0.95515 0.04485 ) *
##      5) NumOfProducts: 3,4 88  103.10 1 ( 0.27273 0.72727 ) *
##    3) Age > 41.5 2264 3039.00 0 ( 0.60468 0.39532 )
##      6) NumOfProducts: 1,2 2131 2785.00 0 ( 0.64008 0.35992 )
##       12) NumOfProducts: 1 1286 1783.00 0 ( 0.50700 0.49300 )
##         24) IsActiveMember: 0 630  802.00 1 ( 0.33333 0.66667 )
##           48) Age < 49.5 402  552.90 1 ( 0.44776 0.55224 ) *
##           49) Age > 49.5 228  177.60 1 ( 0.13158 0.86842 ) *
##         25) IsActiveMember: 1 656  828.50 0 ( 0.67378 0.32622 ) *
##       13) NumOfProducts: 2 845  735.70 0 ( 0.84260 0.15740 ) *
##      7) NumOfProducts: 3,4 133   42.62 1 ( 0.03759 0.96241 ) *
```

We notice that some leaves are very close to be pure:

- The purest leaf is number 7, with 133 training examples attached, 96,24% of which are labeled 1. This leaf collects people who hold more than two products of the bank and whose age is above 41.5.
- The less pure leaf is number 48, with 402 training examples attached, 55% of which are labeled 1. This leaf collects not active members holding just one product and whose age is between 41.5 and 49.5.

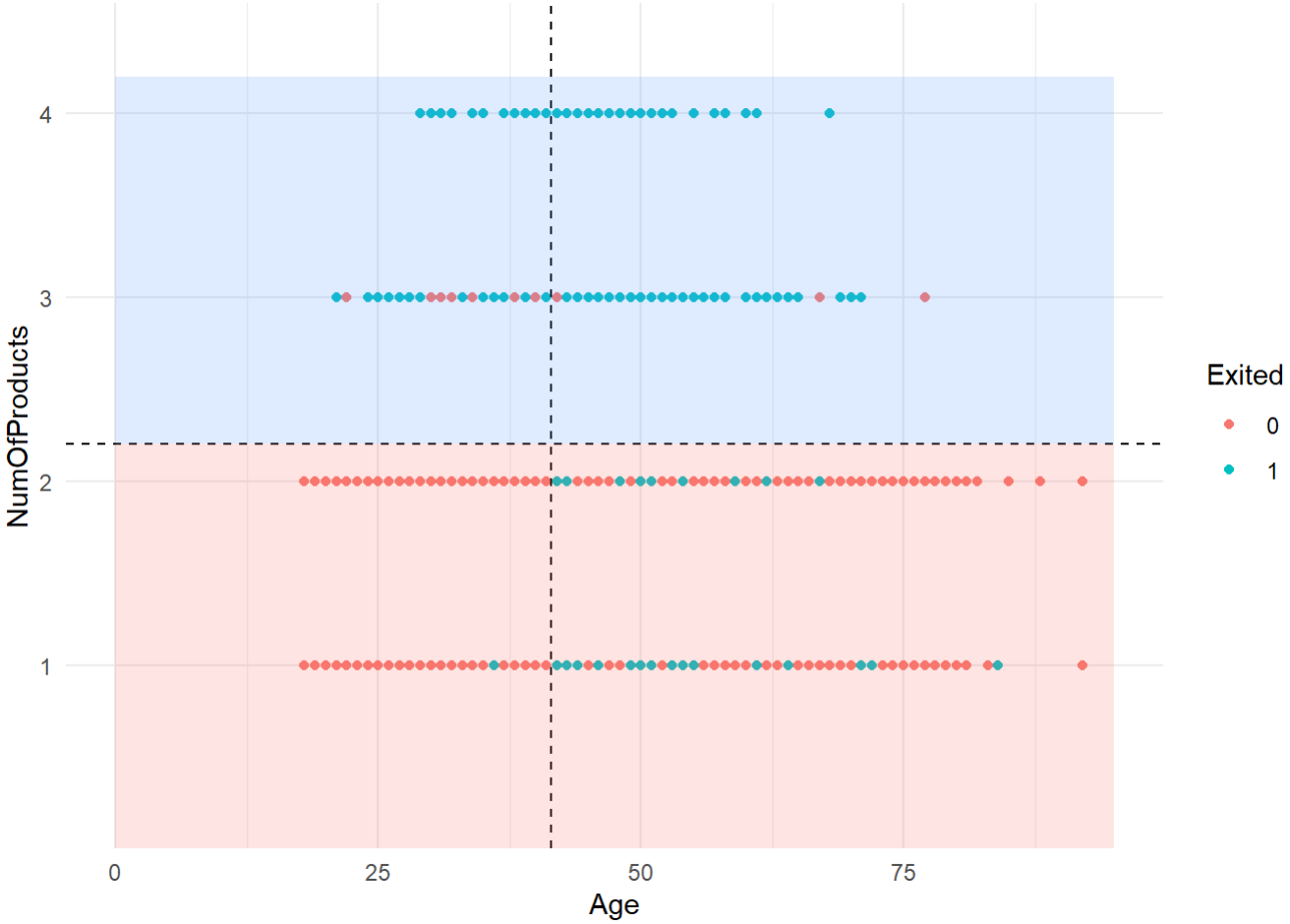
Interestingly, there are two splits which appear to be completely useless, as they assign the same label independently from the result of the associated node test, as in the case of node 24 split. However, this split is not useless as it may appear at first sight. In fact, before the splitting the distribution inside the leaf was only 67-33, which is not so pure. However, node 49 presents a high degree of purity. In other words, even though the split does not reduce the training error, it definitely helps in reducing measures of impurity like the Gini Index or cross-entropy, and that is the reason why the split is performed anyway.

As anticipated in the introduction, the main advantage of trees is their interpretability. This is particularly true in our case, where the tree is very simple and short.



Given any example, one can easily classify it following the corresponding branches on the tree.

Since the predictors involved in the tree are only three, we can also give a simplified representation of the tree in the bi-variate space including `Age` and `NumOfProd` variables.



As we can see in this simplified representation, the two cutoffs divide the predictors space into four regions. We immediately notice that in each region one of the labels is always occurring more than the other: in the top-right quadrant almost all examples are churns, while in the bottom-left region almost all examples are non-churns.

In order to properly evaluate the predictor, we compute the misclassification rate over the test set.

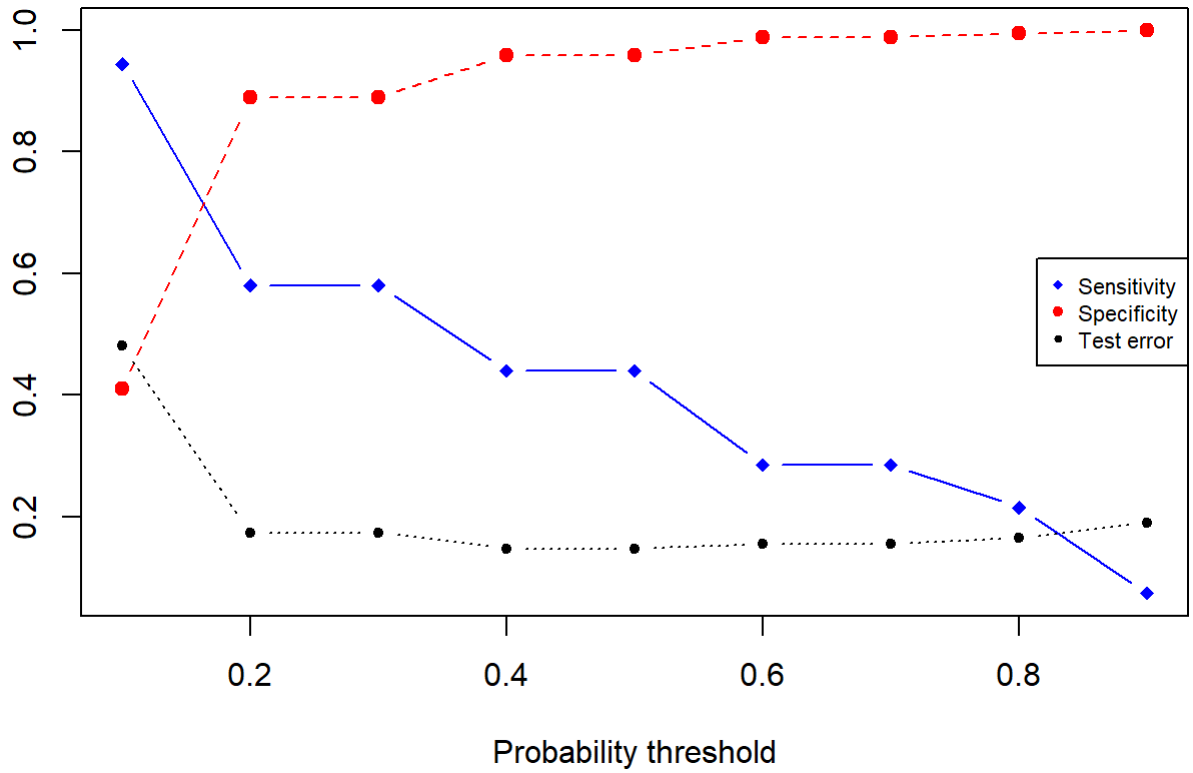
```
## [1] 0.1466667
```

The test error is a little bit smaller than the training error, resulting in an apparently very good performance for this simple classifier also in the test set. However, it turns out that the tree is not so good in predicting actual churns, i.e. the classifier presents a very low sensitivity.

```
##
## tree.pred    0    1
##         0 2292  342
##         1   98  268
```

```
## $Specificity
## [1] 0.9589958
##
## $Sensitivity
## [1] 0.4393443
```

The model is very good at predicting non-churns, while the number of misclassified churns are more than the ones correctly classified. In other words, the model presents a high level of specificity, but a low degree of sensitivity. This may be related to the unbalance in the dataset, where 80% of the observations are non-churns; nonetheless, we would like to boost the goodness of our classifier in terms of sensitivity. Instead of predicting the actual label of the test example, we now predict the probability distribution associated to it, so that we can set different threshold and understand how sensitivity and specificity evolves as we change the assignment threshold.



The probability threshold of 0.2 seem to yield the optimal performance for our tree according to all of the three analysed aspects. More precisely, we obtain a tree with the following characteristics:

```
## $Sensitivity
## [1] 0.5803279
##
## $Specificity
## [1] 0.8891213
##
```

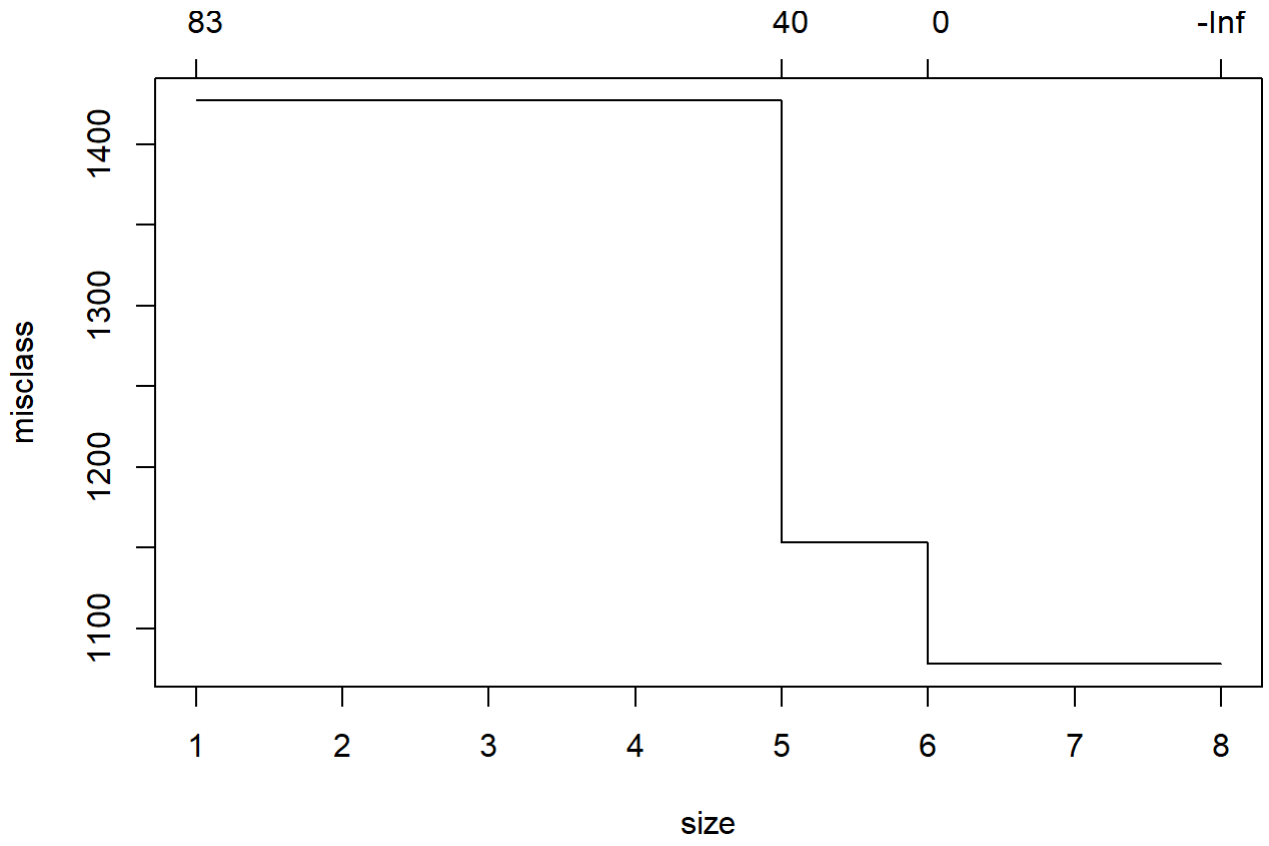
```
## $`Test misclassification rate`
## [1] 0.1736667
```

In other words, the gains in terms of sensitivity definitely outclass the losses in terms of test error and specificity.

We can also try to build a tree with the package `CHAID`, which allows for multiple (and not just binary) splits, growing the tree vertically as well as horizontally. [Coudn't install the package CHAID, eventually Ozlu will show the result]

Pruned Trees

Even though our tree is already short and performs pretty well, we can try to prune it, i.e. reduce its complexity and consequently the variance of the prediction, at the price of a higher bias. Pruned trees involve the selection of a tuning parameter that penalizes the total number of leaves T. We use cross validation to choose the optimal value for that parameter, and then we plot the misclassification rate of the output classifier as the value of the parameter changes.

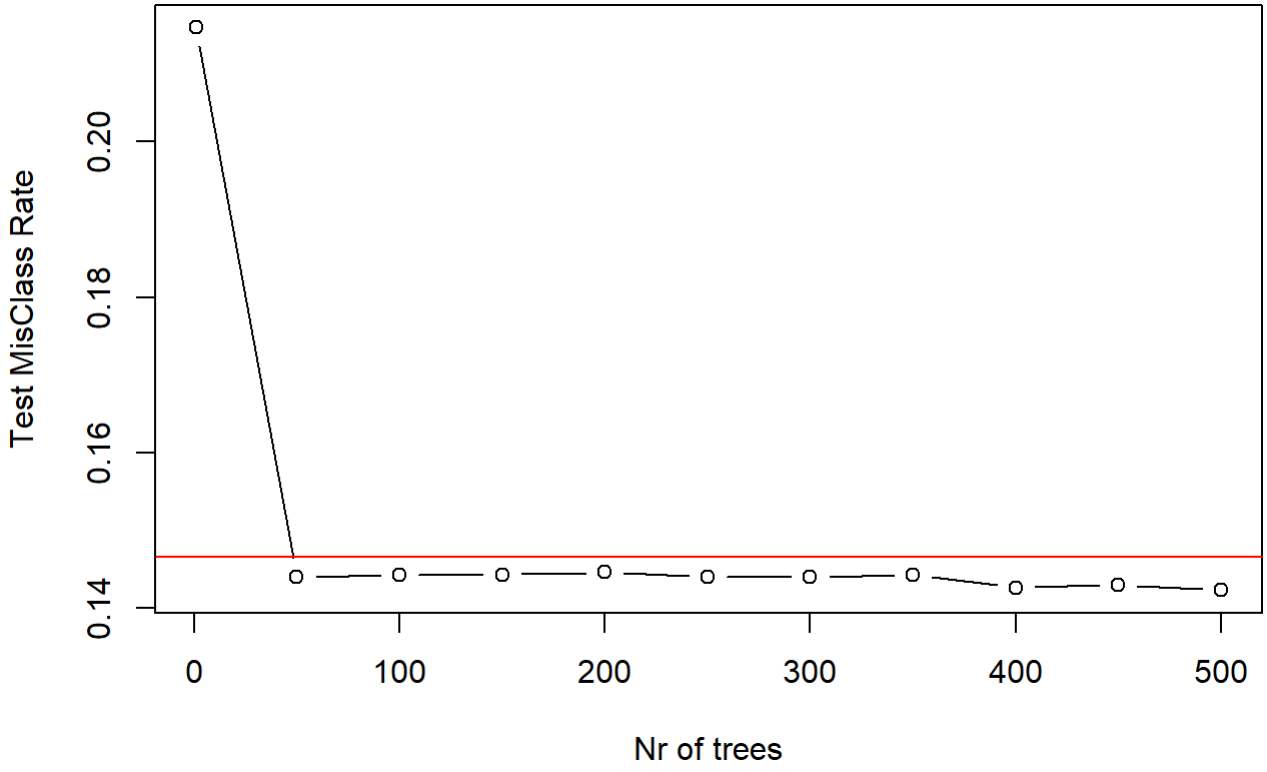


As one could expect, pruning a short tree doesn't improve the predictive performance; indeed, we observe the opposite. The graph in fact shows that the lower misclassification rate is the one reached by the largest tree, which is our starting one.

Bagging

As already stated in the introduction, trees are usually not so powerful predictors in terms of accuracy. One of the main techniques to improve their performance is bagging, which is based on bootstrap. Basically, we perform bootstrapping to create several different training sets from our original one; then a tree is grown on each of this training set and the final prediction for any given example is the majority of the label predicted by the whole set of trees. By averaging the predictions, we successfully reduce the variance of the prediction, just like the variance of the sample mean of

a random variable is lower than the variance of the random variable itself. We therefore analyse how the training error changes as the number of bootstrapped sample increases.



As we can see, the misclassification rate dramatically decrease initially, but then there is not much difference between performing bagging with 100 trees or with 400 trees, i.e. bootstrapped training sample. More precisely, the minimum value for the test error is

```
## [1] 0.1423333
```

Reached by the forest of 500 trees. However, notice that all the values of the training error between 50 and 500 trees are within the interval of 5 thousandth, so there is not really much difference between those classifiers. With respect to the single tree (whose test error is the red line in the

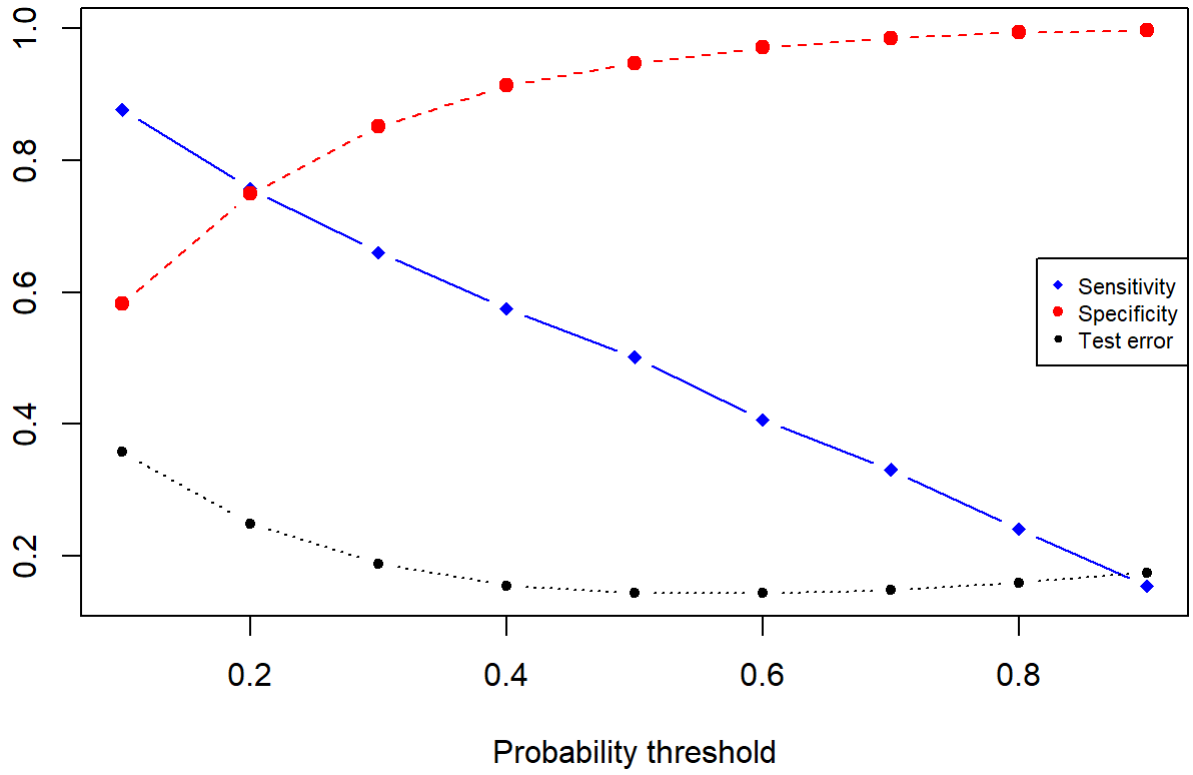
graph), we managed to improve the misclassification rate by just 0,4%.

We now plot the confusion matrix associated to the bagged predictor with 500 trees.

```
##
## bag.pred    0    1
##        0 2264  303
##        1  126  307
```

```
## $Specificity
## [1] 0.9484709
##
## $Sensitivity
## [1] 0.5032787
```

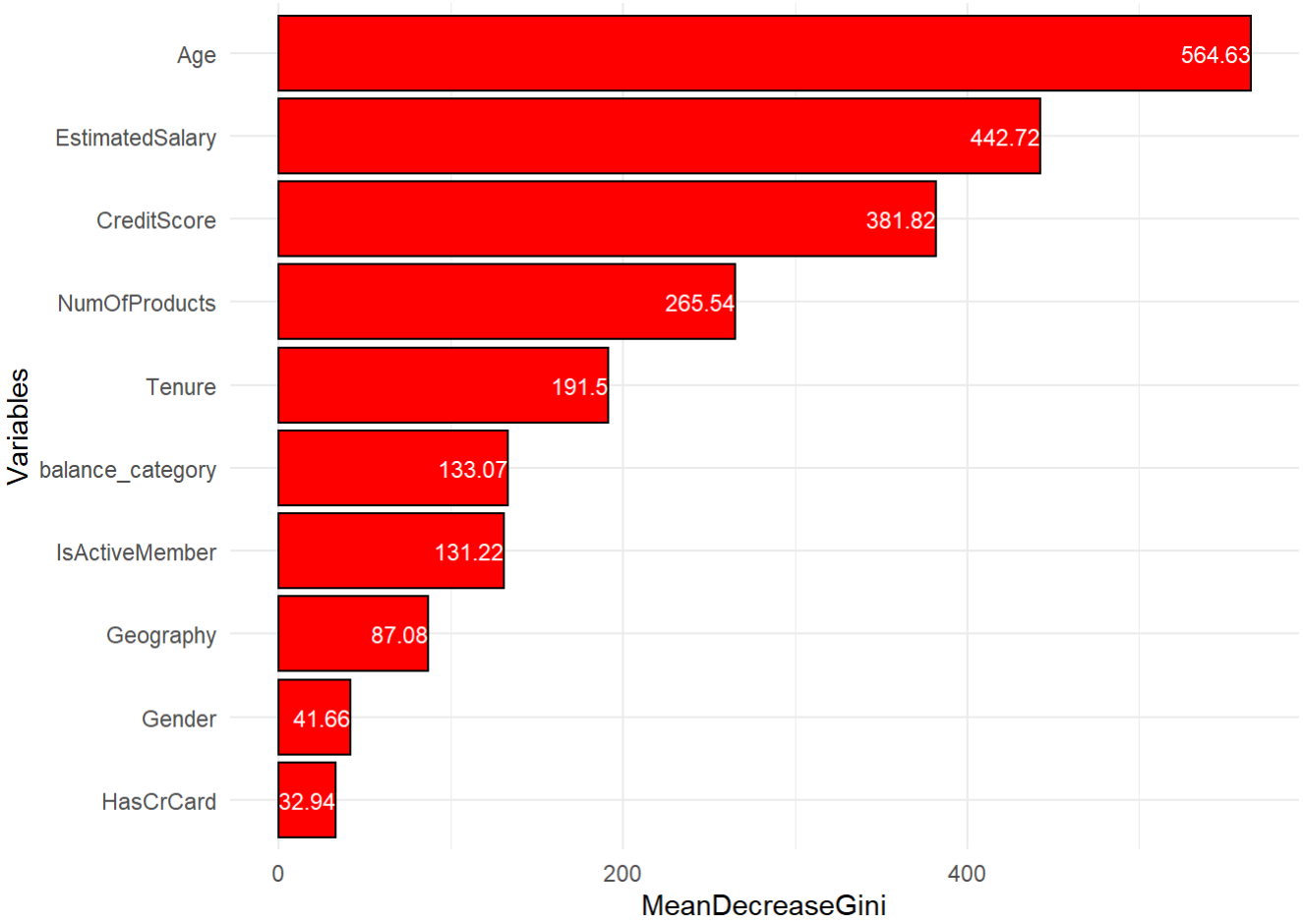
Once again, the results are not really comforting in terms of sensitivity, which for our “optimized” tree was 58%, and so higher than the one shown above. We therefore change our approach, and again predict the probability distribution of the label rather than the label itself. The sensitivity-specificity trade-off graph is shown below.



Again, 0.2 seems to be the optimal value, according to which we obtain a classifier with the following characteristics:

```
## $Sensitivity
## [1] 0.757377
##
## $Specificity
## [1] 0.7493724
##
## $`Test misclassification rate`
## [1] 0.249
```

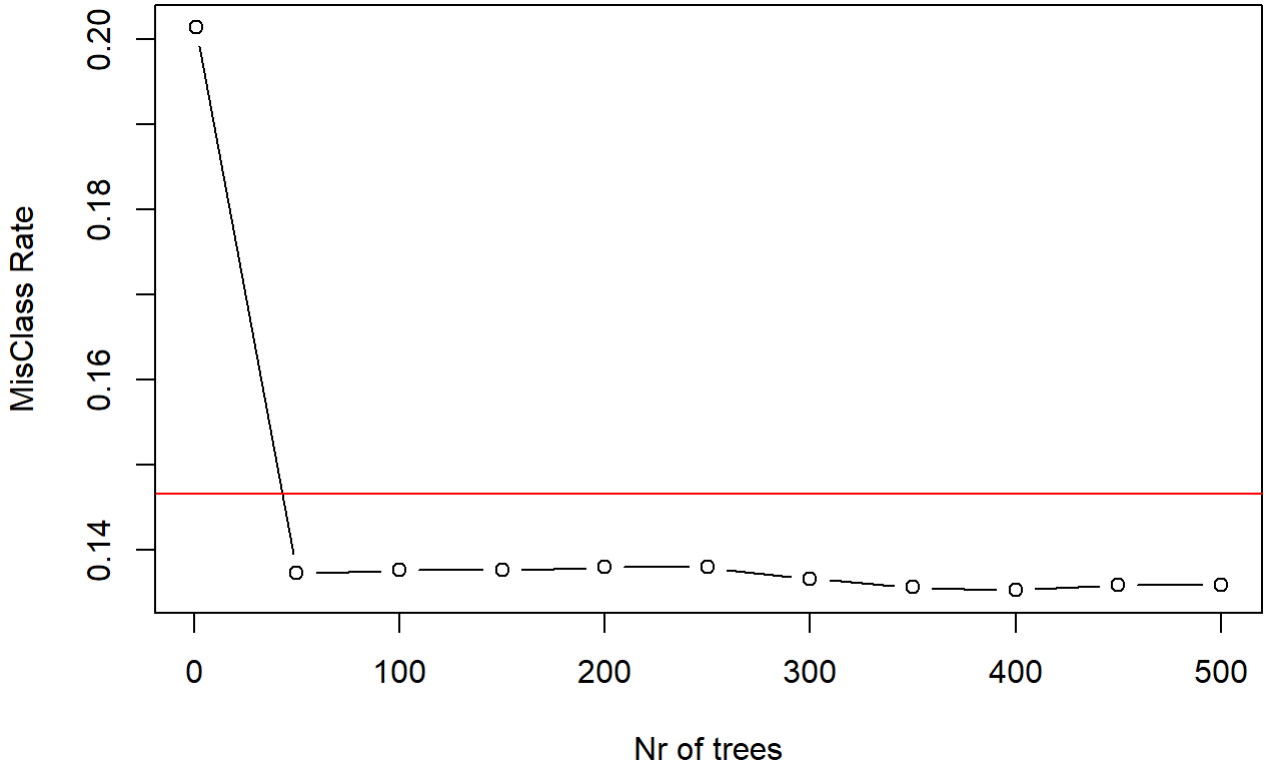
We now investigate which were the variables that were more often considered in the forest we have just grown.



`Age` still plays the most relevant role in obtaining pure leaves, and in fact was placed at the root of the tree we estimated at the beginning. `NumOfProducts` is now in the middle of the ranking, while `IsActiveMember` plays a totally secondary role. `Gender` and `HasCrCard` are the less important variables.

Random Forests

We now consider random forests for our classification problem. This approach is expected to improve the performance of bagging, especially in those settings where some of the available predictors are more correlated than others. In fact, what would happen in this case is that most of the trees built on different training sets will use the same (most correlated) set of variables, resulting in a forest of very similar (and therefore correlated) trees. Unfortunately, averaging correlated predictions does not reduce the variance of the classifier, rather the opposite. Random forests therefore provide a way of *decorrelating* the grown trees. They do so by considering only a random sample of the available predictors for each split. In other words, at each split the algorithm will consider different predictors, so that the final set of trees will not be similar as in the case of bagging. The typical choice for the number of predictors to be considered at each split is the square root of the number of available predictors, so in our case we will select m = 3. As done for bagging, we now investigate how the test error changes with different number of trees in the forest.



As for bagging, also the random forest show a strong decrease initially which is immediately stabilized after 100 trees. However, differently from bagging, the performances of random forest are significantly better than the single tree, whose test error is represented by the red line.

The minimum test error with random forest is

```
## [1] 0.1353333
```

Reached for 400. We therefore consider this particular random forest and plot the associated confusion matrix.

```
##
## rf.pred    0    1
##       0 2299  330
##       1   91  280
```

As for bagging, also the random forest shows a low degree of sensitivity, even lower than bagging, which correctly classified half of the churns even before the sensitivity optimization.

```
## $Specificity
## [1] 0.9619247
##
## $Sensitivity
## [1] 0.4590164
```

Since random forest is basically bagging with random predictors considered at each split, we apply once again the 0.2 threshold and see if there are improvements in terms of sensitivity. The results are shown below.

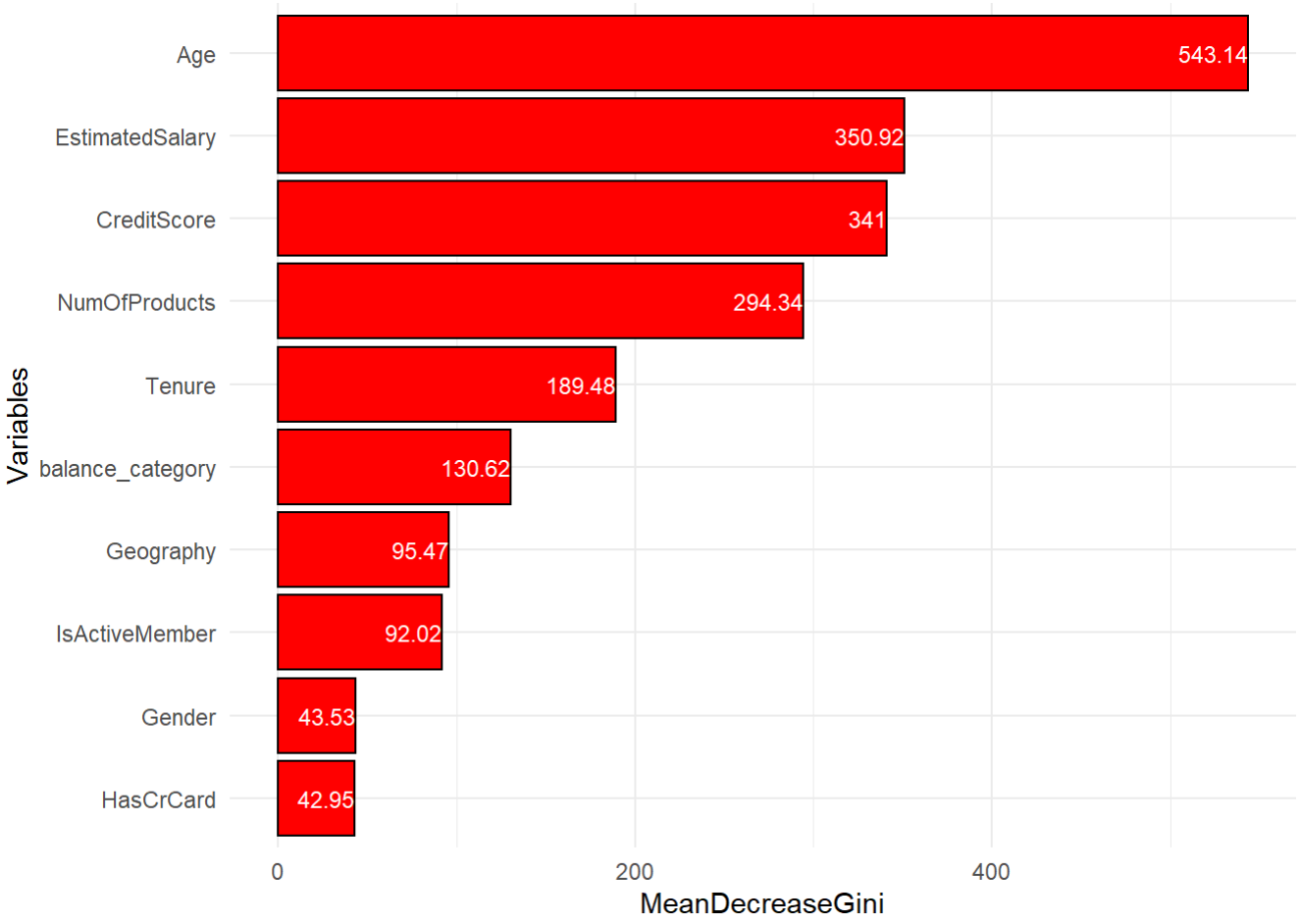
```
##
## rf.pred    0    1
##       0 1859  147
##       1  531  463
```

```
## $Sensitivity
## [1] 0.7590164
##
## $Specificity
```

```
## [1] 0.7778243
##
## $`Test misclassification rate`
## [1] 0.226
```

With respect to bagging, the random forest yields a higher value of specificity and a lower test error, while the sensitivity is more or less the same.

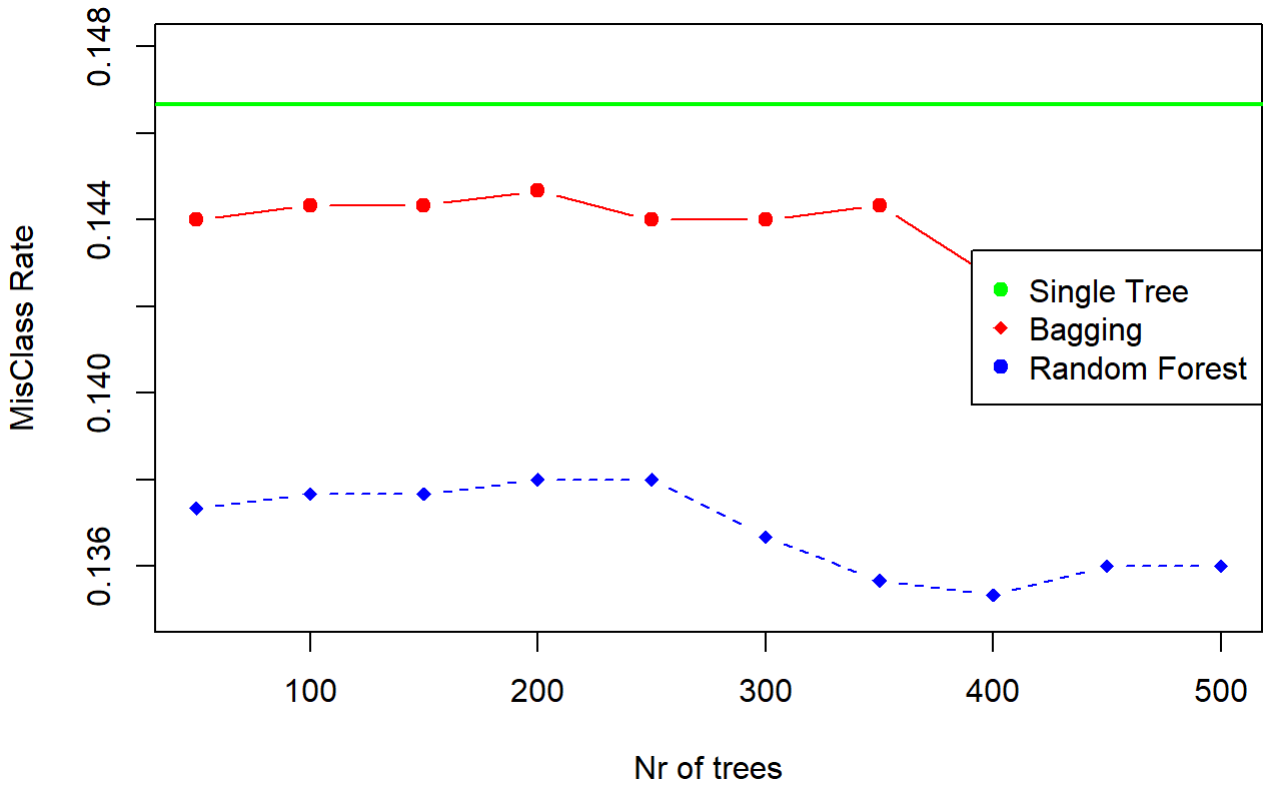
We know investigate which were the variables more relevant in the decrease of the Gini Index for the random forest we have just grown.



The plot is not very different from the one shown for bagging, with `Age` still playing the leading role.

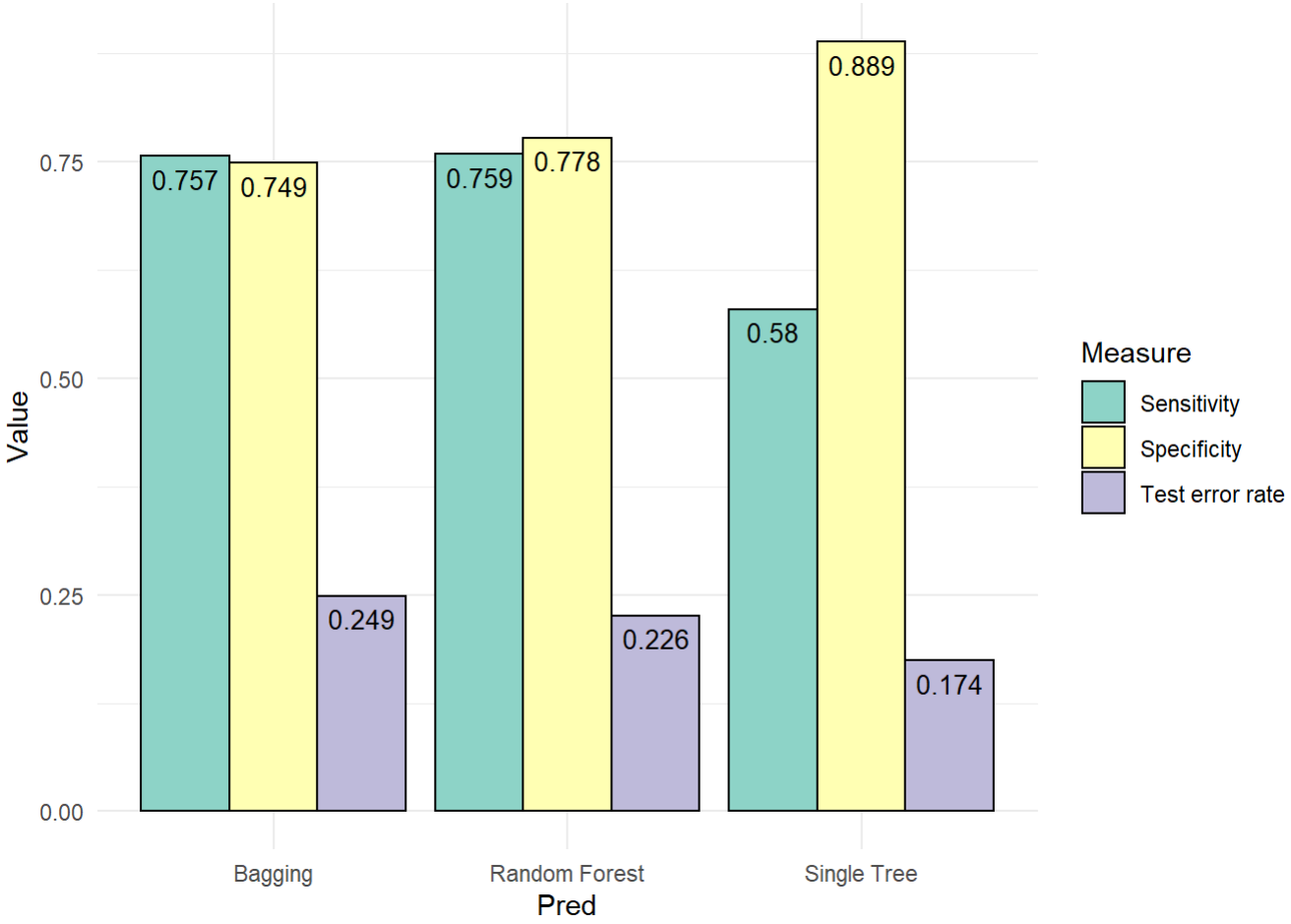
Overall comparison

We now compare random forest and bagging as the number of trees enlarges.



As shown by the graph, the performance of bagging and single tree do not differ much; only for a large number of trees we can observe some relevant difference. The real improvement is with the random forest, whose curve is always below the other two: the minimum test error of random forest is in fact 1% lower than the test error of the single tree. Overall, random forest performs significantly better than bagging irrespective of the number of trees grown.

The graph below instead summarizes the performances of the three algorithms in their “sensitivity-optimized” version.



With respect to the other two methods, the single tree presents a very high imbalance between sensitivity and specificity, so we would discard this classifier. Bagging and random forest have a similar degree of sensitivity, but random forest performs better than bagging both for what concerns the test misclassification rate and the specificity. That is why we consider random forest to be the optimal choice among the three.