Exercise 4

Business Analytics and Data Science WS16/17

Introduction

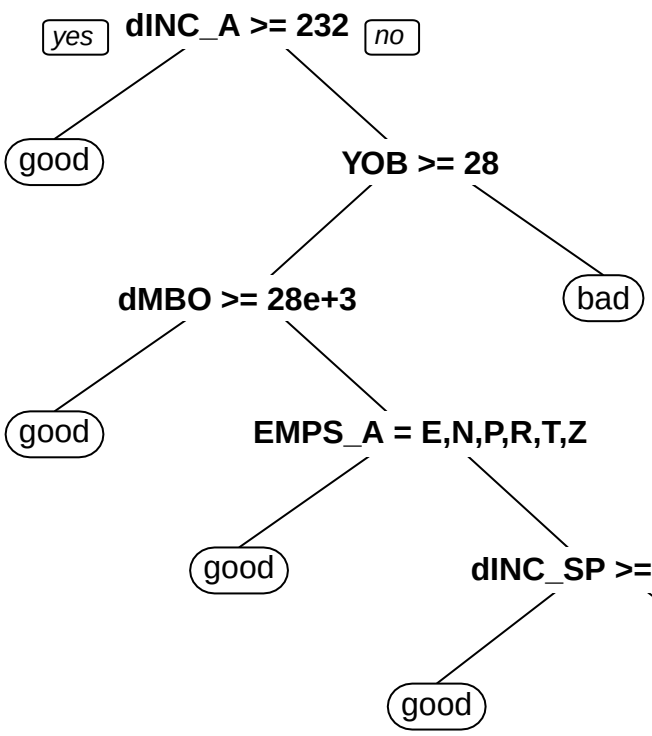
In this exercise, you will look deeper into a very popular machine learning model called "Decision tree" that can be used for classification and regression problems. Later in the course, we will see them again as part of the very popular random forest ensemble model. Trees are very interpretable and can, at least in principle, accomodate missing values in new observations to be predicted.

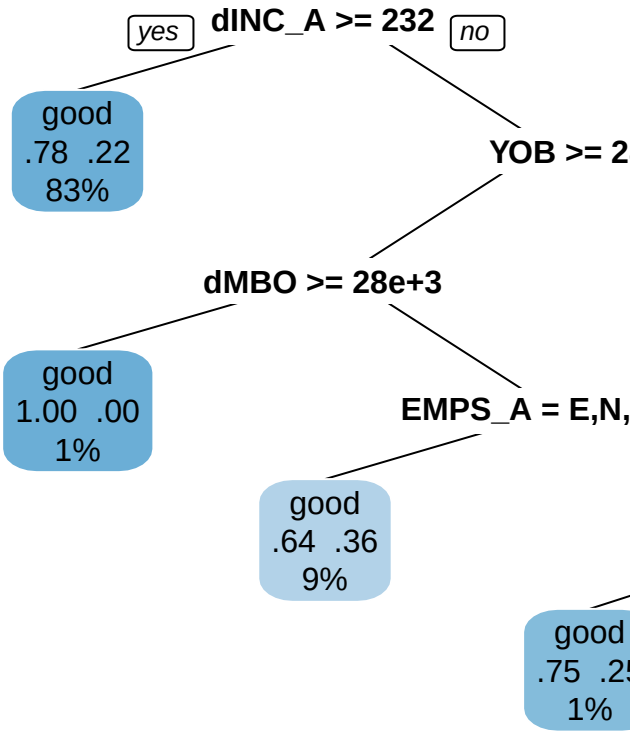
Classification and regression trees work by partitioning the data into smaller, more homogeneous groups. Based on some measure of homogeneity, e.g. the Gini index/impurity (in the two class case $p_1(1-p_1)+p_2(1-p_2)$) or overall sums of squared errors (SSE), the algorithm looks for the variable and split for this variable that most increases homogeneity in the resulting partitions. This splitting process continues within the newly created partions until a pre-defined maximum depth or minimum number of observations in each node is reached. Predictions can then be calculated based on the category probabilities in the terminal nodes (for classification) or a model trained on each subgroup (for regression).

Decision Trees

Decisions trees are among the most basic machine learning algorithms used for classification and regression.

1. Load the loan data set using your custom function from Exercise 4.
2. Use function **rpart()** in the package with the same name to build a decision tree on the data with the default options.
3. As before, predict the the default probability (i.e. BAD == 1) and compare the performance to the models from exercise 4 using the brier score.
4. Use package **rpart.plot** to visualize your decision tree. Have a look at the options to optimize it, e.g. to look at how many observations fall into each node.
5. What credit risk does a 55-year old man without income (no response on employment) with three childen and no outstanding mortgage whose wife earns $60.000 a year pose according to the model tree?






```
##       YOB nKIDS nDEP PHON dINC_SP EMPS_A dINC_A RES dHVAL  dMBO dOUTM dOUTL
## 40     45     0    0    1   30000      W     NA   O 21248 14464     0     0
## 40.1   45     0    0    1   30000      W      0   O 21248 14464     0     0
## 41     68     0    0    1       0      T     NA   P     0     0     0     0
## 41.1   68     0    0    1       0      T   6189   P     0     0     0     0
## 42     54     2    0    1       0      P     NA   O 43392     4     0     0
## 42.1   54     2    0    1       0      P  55068   O 43392     4     0     0
## 43     45     4    0    1       0      E     NA   O 28928 34464   988     0
## 43.1   45     4    0    1       0      E  42000   O 28928 34464   988     0
## 44     13     0    0    0     850      R     NA   U     0     0   520     0
## 44.1   13     0    0    0     850      R  37500   U     0     0   520     0
## 45     58     0    0    1       0      P     NA   O 26464 64000   520     0
## 45.1   58     0    0    1       0      P  40500   O 26464 64000   520     0
## 46     66     0    0    1       0      P     NA   P     0     0     0     0
## 46.1   66     0    0    1       0      P  21750   P     0     0     0     0
## 47     59     1    0    1   16000      V     NA   O 24928 54464   960   180
## 47.1   59     1    0    1   16000      V  15000   O 24928 54464   960   180
## 48     68     0    0    1       0      T     NA   P 48928  6464     0     0
## 48.1   68     0    0    1       0      T      0   P 48928  6464     0     0
## 49     64     0    0    1       0      R     NA   O 34464     4     0     0
## 49.1   64     0    0    1       0      R      0   O 34464     4     0     0
## 50     45     4    0    1   20000      W     NA   F     0     0     0     0
## 50.1   45     4    0    1   20000      W      0   F     0     0     0     0
##       dOUTHP dOUTCC  BAD YOB_missing
## 40         0      0 good           0
## 40.1       0      0 good           0
## 41         0      0 good           0
## 41.1       0      0 good           0
## 42         0      0 good           0
## 42.1       0      0 good           0
```

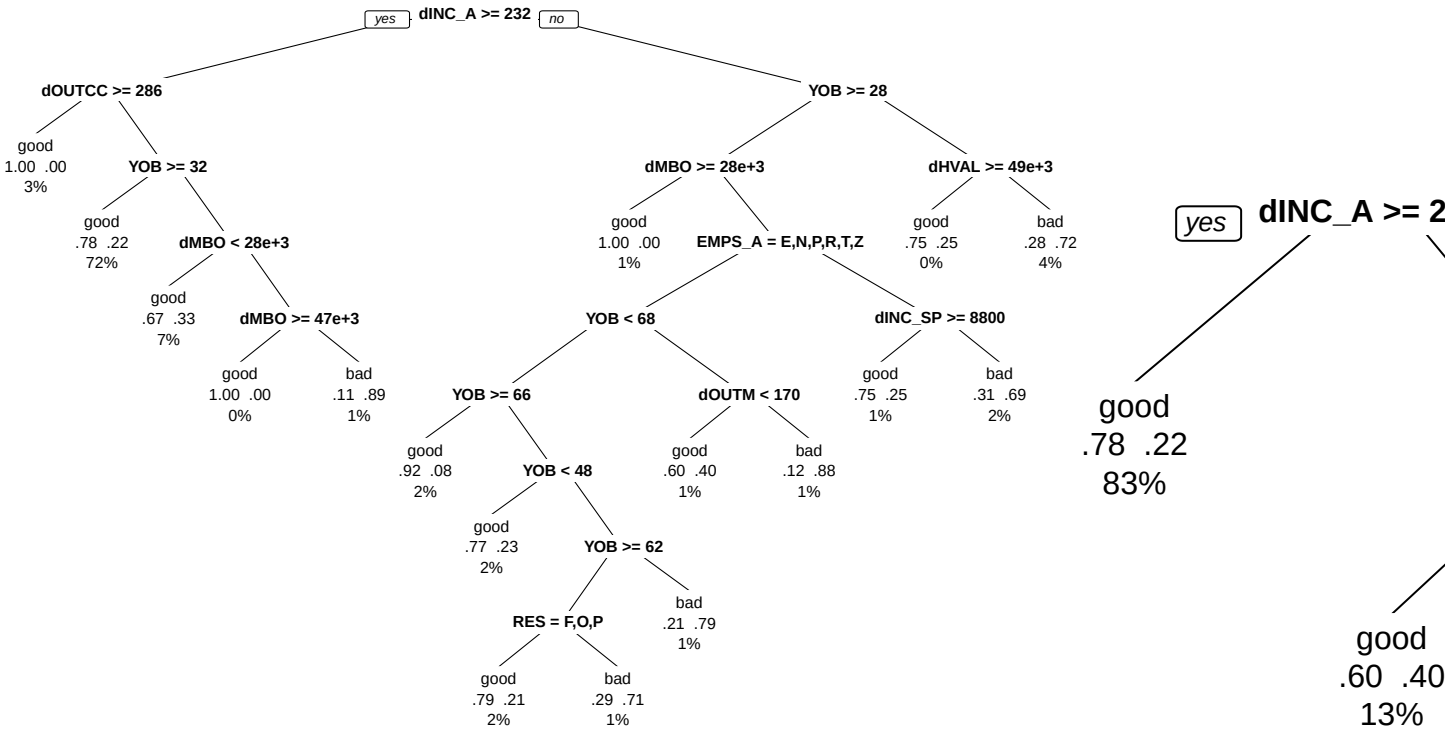
```
## 43        0      0 good           0
## 43.1      0      0 good           0
## 44        0      0 good           0
## 44.1      0      0 good           0
## 45        0      0 good           0
## 45.1      0      0 good           0
## 46        0     60  bad           0
## 46.1      0     60  bad           0
## 47        0      0 good           0
## 47.1      0      0 good           0
## 48        0      0 good           0
## 48.1      0      0 good           0
## 49        0      0 good           1
## 49.1      0      0 good           1
## 50        0      0 good           0
## 50.1      0      0 good           0
```

```
##        40      40.1        41      41.1        42      42.1        43
## 0.2500000 0.2500000 0.3603604 0.2227674 0.2227674 0.2227674 0.2227674
##      43.1        44      44.1        45      45.1        46      46.1
## 0.2227674 0.2227674 0.2227674 0.2227674 0.2227674 0.2227674 0.2227674
##        47      47.1        48      48.1        49      49.1        50
## 0.2227674 0.2227674 0.3603604 0.3603604 0.2227674 0.3603604 0.2500000
##      50.1
## 0.2500000
```

Controlling complexity (or A deeper look into shallow trees)

In theory, we can grow a tree by adding splits until every observation in the training set is classified correctly. In practice, we don't do this because the trees very complex and will overfit the training data, a problem that we will discuss in more detail during the next exercise. The process of stopping early or discarding possible splits is called *pruning* the tree. There are several stopping criteria that can be set to achieve this.

1. Build another decision tree on the data and save it as **dt.full**, but this time override the default option settings. Set a complexity parameter value of 0 and a minimum number of observations that must exist in a node in order for a split to be attempted of 3. Look at the tree structure. Note: This tree will be large, so plotting it may take a while.
2. Let's see what happens when we build trees that are a little more or a little less complex than the default trees. Check the default complexity values by reading **?rpart**. Build two more decision tree and save them as **dt.prunedLess** and **dt.prunedMore**, with settings cp = 0.005, minbucket = 4 and cp = 0.02, minbucket = 8. Plot the trees.
3. As before, predict the the default probability (i.e. BAD == 1) and compare the performance of all the trees using the brier score. Which tree would you recommend to the bank?




```
##       full prunedLess    default prunedMore
## 0.01702284 0.16670617 0.18008512 0.18340471
```