Exercise 8

Business Analytics and Data Science WS16/17

Introduction

The random forest is a tree-based classifier that performs competitively in many applications. As explained in the lecture, it is a combination of decision trees (**forest**) grown on random subsets of the feature space (**random**). Simply speaking, a large number of decison trees are (almost) fully grown but at each split only a (usually small) number of variables is randomly chosen as candidates for the split to make the trees more different.
In this exercise, we will use package **caret** to train a random forest, but will also see one of the boundaries of the package. Consequently, we will explore the benefit of replacing caret by a little infrastructure of our own here.

Exercise 8.1: Random forest with caret

1. Install the **caret** and **randomForest** package and load the data set loans. Split it into training and test set with a ratio of 80:20 as done in the previous exercises.
2. Load package **caret** and create an object model.control which contains the setup for the model estimation framework. Check the help and set the following control parameters in **trainControl()**:
   - (Unrepeated) 5-fold cross-validation
   - Class probabilities should be calculated
   - **twoClassSummary** as **summaryFunction** as done in the previous excercise.
3. Specify the variable values compared during model selection in an object **rf.parms**. The standard random forest algorithm in **caret** can be tuned over the number of variables tried at each split (option **mtry**), e.g. 1 to 5
4. Tune the random forest and save the best model in a variable **rf**. Compare the performance of the candidate models. Recall: The **train()** function allows you to do model tuning in one call. Specify the random forest method and (as done in the previous exercise) ROC as metric to be used to compare on AUC values.
5. In line with the previous exercise, predict **yhat.rf** and evaluate the model performance **auc.rf** on the test set using the AUC.

Exercise 8.2: Random forest without caret

As you saw in the lecture and in the train call above, random forest models have another parameter to be specified: **ntree**, the number of trees in the combination. Following the intuition "more is better", caret does not consider this a tuning paramter for which candidate models can automatically be compared. We will build our own framework, or extend the cross-validation framework from earlier, to tune over both parameters.

1. Build a framework to tune the number of randomly sampled variables at each split **mtry** and the number of trees you need to grow **ntree**. To do this, initialize a grid **rf.parGrid** containing the values of **mtry** and **ntree** that you want to compare. Try 1 to 10 variables sampled for each split and between 100 and 1000 trees.
2. As in previous exercises, use the **cut()** function for gain a vector *folds* of indices that separate the training data into 5 folds of equal size.
3. Analogeously to the previous exercise, create two loops, one within the other, that perform the following:
   - For each mtry-ntree combination candidate n:
     - For each fold k:

    - Train a random forest model **rf** with **ntree** and **mtry** equal to the n-th candidate of the tuning grid on the folds not including fold k.
    - Predict the outcomes for the current validation fold, compute the AUC values on the validation fold, and save this value. Recall: Function **auc()** from package **pROC** does this quickly.

      Note: The above structure is called pseudo-code. Before you work on complicated pieces of code, always outline the structure of what you are planning to do in written form to make sure that you have a working plan on what you are trying to do and will not get lost in the middle of your work.
4. Calculate the average AUC value (over five folds) for each parameter combination. Find the best combination.
5. Train a random forest model with the best parameters on the full training sample and save the model to **rf.tuned**.
6. Compare the best performing trees found with caret to the best tree tuned over mtry *and* ntree. You can look at the overall results to investigate the impact of the number of trees on the model for this data set.