Prof. Dr. Steffen Wagner
Angewandte Statistik
Fachbereich II
Berliner Hochschule für Technik

Exercise 11: Ensemble Methods

Machine Learning I – SoSe 24

This workshop covers hierarchical clustering and soft clustering. At the end of the worksheet there are a couple of written exercises for you to do at home, which should be good practice for the exam.

1 Preparations

1.1 RStudio Project

1. Open your Machine Learning 1 RStudio Project
2. Create an R Script file to perform this exercise

1.2 Required Packages

For this exercise you require the following additional R packages. Please make sure that you have installed them on your computer before coming to the workshop session for the case that Eduroam is not working.

```r
# check if packages can be loaded, i.e. they are already installed
library(ISLR2)
library(rpart)
library(rpart.plot)
library(randomForest)  # what could it be? Hm?
library(gbm)           # Generalized Boosted Regression Modeling
```

If you get an error at this stage, you need to install the packages.

2 Bagging

2.1 Bagging a regression tree by hand

You already fitted a regression tree to the *Boston* data from James et. al. in an earlier workshop session. These data are part of the `ISLR2` package that comes with the 2nd edition of James.

1. To access the data set, first load the package and read about the data set using.

You will again look at fitting regression trees to the Boston data using `medv` (median value of owner-occupied homes in $1000s) as the outcome variable. This time you start with the pruned tree for the basic tree, and then look at the results of "bagging" this tree.

2. Start by loading the `rpart` libraries, creating a training set, and fitting the tree to the training data using a complexity parameter of 0.019, the value found in Workshop 8 for the pruned tree.

```
require(rpart)
require(rpart.plot)

set.seed(1)
train <- sample(1:nrow(Boston), nrow(Boston)/2)  # row numbers training data
test <- (1:nrow(Boston))[-train]

tree.boston <- rpart(medv ~ ., Boston, subset = train, cp = 0.019)
print(???)
rpart.plot(???)
pred.train <- predict(???, newdata = Boston[train, ])
mean((Boston$medv[train] - pred.train)^2)
pred.test <- predict(???, newdata = Boston[test, ])
mean((Boston$medv[test] - pred.test)^2)
```

   What is the difference between the third last and the last line? Is the observed difference plausible?

Still using the `Boston` housing data, you will now obtain bagged samples of these data and create a tree, in order to get bagged tree predictions. In Exercise 2.2 you will use R code to do this directly.

3. Obtain one bagged (bootstrap) sample of the training data, and the out of bag elements.

4. Fit a *full* tree for this bag. Store the fitted values for the full data set, selecting the right rows, calculate the in-bag, the OOB and the test data MSE.

```
tree.bag <- rpart(medv ~ ., Boston, subset = bag.samp, cp = 0)
preds <- predict(tree.bag, newdata = Boston)

inbagMSE <- mean((Boston$medv - preds)[bag.samp]^2)
inbagMSE

oobMSE <- mean((Boston$medv - preds)[oob.samp]^2)
oobMSE
```

```
test.MSE <- mean((Boston$medv-preds)[test]^2)
test.MSE
```

Notice that the cp-value is set to zero. When bagging, a full tree is grown giving low bias-high variance trees. The averaging out of all the bagged trees reduces the variance.

5. Create a loop so that B=100 bagged trees are fitted.

```
B <- ???
inbagMSE <- oobMSE <- test.MSE <- rep(NA, B)
preds <- matrix(NA, nrow = nrow(Boston), ncol = B)
for (i in ???){
  bag.samp <- ???
  oob.samp <- ???

  tree.bag <- ???

  preds[,i] <- ???
  inbagMSE[i] <- ???
  oobMSE[i] <- ???
  test.MSE[i] <- ???
}
```

6. After running the loop calculate the overall predictions for the training data, and for the test data (mean over each bag). Calculate the overall OOB MSE, and the test data MSE and compare the results with those for the single tree.

```
plot(inbagMSE)
```

```
mean(inbagMSE)
plot(oobMSE)
```

```
mean(oobMSE)
plot(test.MSE)
```

```
mean(test.MSE)
```

Notice that each individual tree doesn't do any better in terms of MSE than one single tree.

7. Now average the predicted values (using `apply()`) in order to get the bagging model predictions, and compare the results.

```
all.preds <- apply(preds, 1, mean)

# calculate the bagged prediction MSE
mean((Boston$medv[train] - all.preds[train])^2)
mean((Boston$medv[test] - all.preds[test])^2)
```

These are much better values than for a single tree model.

2.2 Bagging using the R function `randomForest`

The command `randomForest` is the R command to fit a bagged tree directly. The important argument is `mtry = 12`, which specifies that all 12 of the possible predictor variables are used, which is the requirement for a bagged tree.

```
library(randomForest)
bag.boston <- randomForest(medv ~ ., data = Boston , subset = train ,
                           mtry = 12, ntRee = B, importance = TRUE)
bag.boston
```

1. Calculate the MSE for the test data. Comparing this figure with the bagged test MSE in Exercise 2.1, `randomForest` gives a slightly better result. The tree fitting options have been optimised for bagging (e.g. pruning), which for simplicity we did omit in exercise 2.1.

2. The following two functions returns/plots the so called *variable importances* for an object created by `randomForest`.

   ```
   importance(bag.boston)
   varImpPlot(bag.boston)
   ```

3. Which are the five most important variables? Rerun `rpart()` to get the default tree (`cp = 0.01`), and plot the tree. Compare the “important” variables with those used in the default single tree?

3 Random Forests

3.1 Random forest regression trees by hand

The structure of this exercise is very similar compared to exercise 2.1. Only minor adjustments to this existing code are necessary.

1. Copy and paste your bagging code from Exercise 2.1, commenting your source file so you know where this exercise starts. Load the required packages. Create the same training and test data sets as you did last time.

A random forest is the same algorithm as bagging, but in each iteration, only a small subset of explanatory variables are made available for the tree. In the Boston data there are $p=12$ explanatory variables and the rule of thumb is to take $m=\lfloor\sqrt{p}\rfloor$ giving $m=3$.

2. In each bag (inside the for-loop), sample three predictor variables *and* include the 13th variable which is the outcome variable `medv`. This time you should use sampling *without* replacement, because sampling the same variable twice is pointless. The code to do this is:

```
variable.samp <- c(sample(1:12, size = floor(sqrt(ncol(Boston) -1)),
                            replace=FALSE),
                   13)   # Why the 13?
```

3. Adapt your `rpart` data argument to use the bootstrap samples and `variable.samp`:

```
tree.rf <- rpart(medv ~ . , data = Boston[ , ???], subset = train, cp = 0.01)
```

4. Obtain one tree and plot it to see how the variables in the tree are different from before.

5. Run the loop, obtain the forest predictions and calculate the MSE as in Exercise 2.1.

We find that these results are not as good as the bagging results, but somewhat better than just using one tree. Try increasing the number of variables selected in each iteration to 6. This gives noticeably lower MSE values, showing that the $m=\lfloor\sqrt{p}\rfloor$ rule is just a rule of thumb.

3.2 Random forest using the command `randomForest`

You already know from exercise 2.2 that the R command for a random forest is `ramdomForest`. Changing the `mtry` argument to a value smaller than the full number of all available explanatory variables (in this case 3) is the only thing we have to do.

1. Copy and past your code from Exercise 2.2, to fit and evaluate the random Forest with `mtry`=3.

2. Obtain the variable importances for the random forest model. Compare the variable importances for the two methods.

3. As in Exercise 3.1, change `mtry` from 1 to 12 and investigate how the test MSE changes.

4 Boosting

4.1 Investigating a boosted tree model

You will use the R package `gbm` to fit a boosted regression tree. The main aim of this exercise is to slowly step through the stages of boosting tree models, in order to see how the *slow learning* in the boosting algorithm develops.

You will start off by running just one iteration of the boosting algorithm, and then adding an extra iterations and investigate the changes. The data set you will use is the `mtcars` data set to predict the variable `mpg` – fuel consumption in miles per gallon. This data set is small enough to be able to investigate in detail, but no so small that it has no interesting features.

In Moodle you can find the R code for this exercise in the file `Ex_11_IntroBoosting.R`.

A description of the steps in the source file are given below. Work through the source file slowly at each stage referring to the details below.

1. Load the `gbm` package and set the seed
2. One iteration
   a. Fit a boosted tree to the `mtcars` Data with just one iteration (`n.trees=1`). The shrinkage parameter $\lambda = 0.1$ (large). Note that the default `interaction.depth` is 1, so each iteration fits a tree with just one split. The other parameters will be explained in the next exercise.
   b. Output summary information. Not so interesting after just one iteration.
   c. Compute the MSE of the null model ($\hat{y}_i = \bar{y}$) and compare it to the MSE after one iteration.
   d. Plot the fitted values against the observed values, and add two reference lines $y = \bar{y}$ and $y = x$. A perfect fit puts all the points onto the diagonal line.
   e. This command gives the effect of the variable called `cyl`. The fitted split is `cyl>5`.
3. Fit a boosted tree with two iterations. Obtain the MSE and again plot the fitted against the observed values. Note that exactly the same split has been fitted, but the separation between the two fitted value levels has got bigger.
4. Fit a boosted tree with three iterations.
   a. Obtain the MSE and again plot the predicted against the observed values. Now there is a third level in the fitted values, the new level contains just two observations.
   b. `summary(mtcars.boost)` gives the “relative influence” of each variable, which is a similar measure to the variable importance of a bagged-tree/random-forest.
   c. The split at the 3rd iteration is on the variable `displ`. The marginal effect of the split can be inspected graphically using `plot(mtcars.boost,i.var="displ")`
5. Fit a boosted tree with 10 iterations,
   a. First inspect the model after 4 iterations. A new split occurs, what is this new split?
   b. After 5 iterations the `cyl>5` is reinforced.
   c. The `matplot` command plots the fitted values against iteration number. Many of the fitted values are the same, so what we see are the different levels in the fitted values.
   d. Inspect the model after 10 iterations.
6. Using `gbm.more()`, add another 90 iterations to the model and investigate. This function means that the first 10 iterations do not need to be re-run.
7. Add another 900 iterations to the model and investigate.

8. Use another shrinkage rate λ and compare how quickly the model fits with different values of λ.

4.2 Boosting with the Boston data

Work through the short subsection 8.3.4 in James et al on pages 359/360.

5 Written exercises

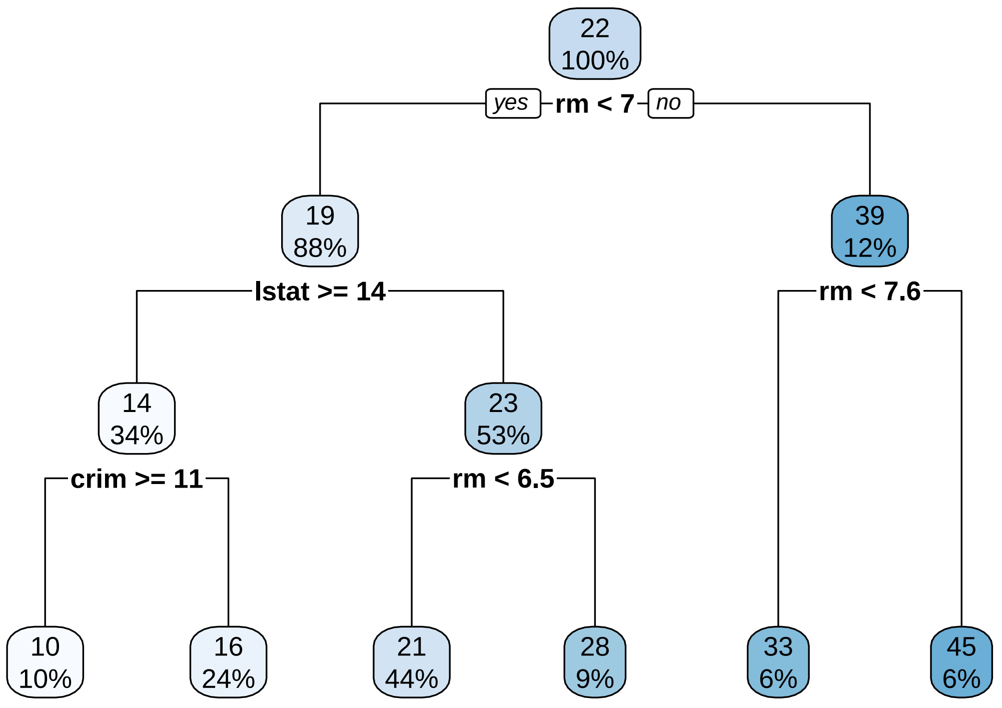
5.1 Calculating Variable importance

Given is a tabular summary and a plot of a simple `rpart` tree for the Boston data. The cp value has been chosen specifically to keep the exercise calculations simple but non-trivial.

```
set.seed(1)
train <- sample(1:nrow(Boston), nrow(Boston)/2)
tree.boston <- rpart(medv ~ ., data = Boston, subset = train, cp = 0.03)
tree.boston$frame$dev <- round(tree.boston$frame$dev, 0)
tree.boston
```

```
## n= 253
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 253 19448 21.78656
##    2) rm< 6.9595 222  6794 19.35360
##      4) lstat>=14.405 87  1554 14.46092
##        8) crim>=11.48635 26   303 10.31538 *
##        9) crim< 11.48635 61   614 16.22787 *
##      5) lstat< 14.405 135  1816 22.50667
##       10) rm< 6.543 111   763 21.37748 *
##       11) rm>=6.543 24   256 27.72917 *
##    3) rm>=6.9595 31  1929 39.20968
##      6) rm< 7.553 16   505 33.42500 *
##      7) rm>=7.553 15   317 45.38000 *
```

```
rpart.plot(tree.boston)
```



Each node in the tree has a row in the numeric output. Each line in the output corresponds to one note giving the values
`node), split, n, deviance, yval`.
A $*$ at the end of the row denotes terminal node. The standard numbering for nodes is applied[1], so that node 3 splits into nodes 6 and 7.

If the node is split further then the splitting variable is given. `n` is the number of elements in that node. The column `dev` outputs the the Squared Error, often called the *loss* or *deviance*. To calculate the “increase in node purity” for one node, take the deviance for the parent node and subtract the deviance in the two child nodes. Assessing this number to the splitting variable. If a variable is used for more than one split then the increase in node purities are added for each of the relevant splits.

[1] for details see Slides *Regression trees*, slide 30.