Machine Learning 2
Data Science
Winter Semester 2018/19

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Prof. Tim Downie

Workshop 1
Tree Models: Ensemble Methods

In this workshop you will need the packages `rpart`, `rpart.plot` and `randomForest`. Load these packages, installing them if necessary.

Exercise 1 Understanding bootstrap sampling

We will take the vector $1, 2, \ldots, n$ and investigate the properties of bootstrapping this vector.
Start with $n = 10$ so that we can easily observe what is going on. The code below does the following:

1. defines n and the vector to be bootstrapped,
2. takes a sample size n with replacement,
3. outputs the raw boot strap sample,
4. sorts the output,
5. creates a frequency table
6. outputs how many values were sampled once, twice, three times etc.:`table(table())`,
7. outputs many "unique values" are in this bootstrapped sample?[1]
8. and calculates how many observations are "out of bag" OOB.

Run the code, checking the output at each stage, and repeat a few times to understand what is going on.

```
n<-10
vec<-1:n
bs.samp<-sample(vec,size=n,replace=T)
bs.samp
sort(bs.samp)
table(bs.samp)
table((table(bs.samp)))
#numer of unique values in the bootstrapped sample
length(unique(bs.samp))
#number of the out of bag values
n.OOB<-n-sum(table((table(bs.samp))))
```

[1] Unique in this sense means "unterschiedlich"

Adapt the code in the following way to investigate the asymptotic behaviour.

- Let $n = 100$.
- Create a for loop to obtain $B = 100$ bootstrap samples.
- In each iteration store the number and proportion of out of bag values.
- Once the for loop is exited, plot the cumulative proportion (hint: `plot(,type="l")` and `cumsum()`).
- Increase the value of B to approximate the asymptotic proportion of out of bag observations.

Compare the proportion you are getting with the approximate proportion stated in Lecture 1 of a third.

Exercise 2 Bagging a regression tree "longhand"

In Workshop 1 you obtained a regression tree for the `Boston` housing data. You will now obtain bootstrapped samples of these data and create a tree, in order to get a bagged tree predictions, and find the *important* variables. You will learn the R function used in practice in Exercise 3.

Recreate the same training sample as in Workshop 1.

If you wrote down the MSE of the test set (pruned and non pruned) the find these values, or repeat the tree fitting part from last week.

Obtain one bagged (bootstrap) sample of the data

```
bag.samp<-sample(train,size=length(train),replace=T)
oob.samp<-train[-bag.samp]
tree.bag=rpart(medv~.,Boston,subset=bag.samp)
```

Store the fitted values and the out of bag predictions for this "bag". Also store the predicted values for the ~~training~~ data.

Put this in a loop and run $B = 100$ times, storing the above information for each iteration (the predictions should be stored in a matrix).

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.

Exercise 3 Bagging with R

The command `randomForest` is the R-Command to fit a bagged tree directly. The important argument is `mtry = 13`, which specifies that all 13 of the possible explanatory variables are considered, so generating bagged trees.

```
bag.boston = randomForest(medv~., data= Boston , subset = train ,
                          mtry = 13, importance =TRUE)
bag.boston
```

Calculate the MSE for the test data. Comparing this figure with the bagged test MSE in Exercise 2, `randomForest` gives a noticeably better result. The tree fitting options have been optimised for bagging, which for simplicity we did not do in Exercise 2.

Exercise 4 Random forest regression trees "longhand"

Copy and paste your bagging code from Exercise 2

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 $p = 13$ explanatory variables and the rule of thumb is to take $m = \sqrt{p}$ and round m down[2], giving $m = 3$.

In each bag, sample four variables and include the 14th variable which is `medv`. We want to use sampling without replacement, because sampling the same variable twice is not helpful. The code to do this is:

```
var.samp<-c(sample(1:13,size=3,replace=FALSE),14)
```

Adapt your rpart data argument to use the bootstrap samples and `var.samp`

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

Run the loop, obtain the forest predictions and calculate the MSE as in Exercise 2

Exercise 5 Random forest using R

You have probably guessed that the R command for a random forest is `ramdomForest` with the `mtry` argument set to 3.

Copy and past your code from exercise 3, to fit and evaluate the ramdom Forest with $m = 3$.

Exercise 6 Variable importance

Variable importance of a tree or ensemble model is described in Lecture 1.

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

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

Compare the variable importances for the bagged and random forest models.

[2]In Lecture 1 I said round up, but inspection of the R code suggests to round down

`rpart` also calculates the variable importance and can be output using `tree.boston$variable.importance`. Note that this includes variables that do not appear in the final tree, but were tried out as part of the tree fitting algorithm.

Exercise 7 Homework: Calculating variable importance

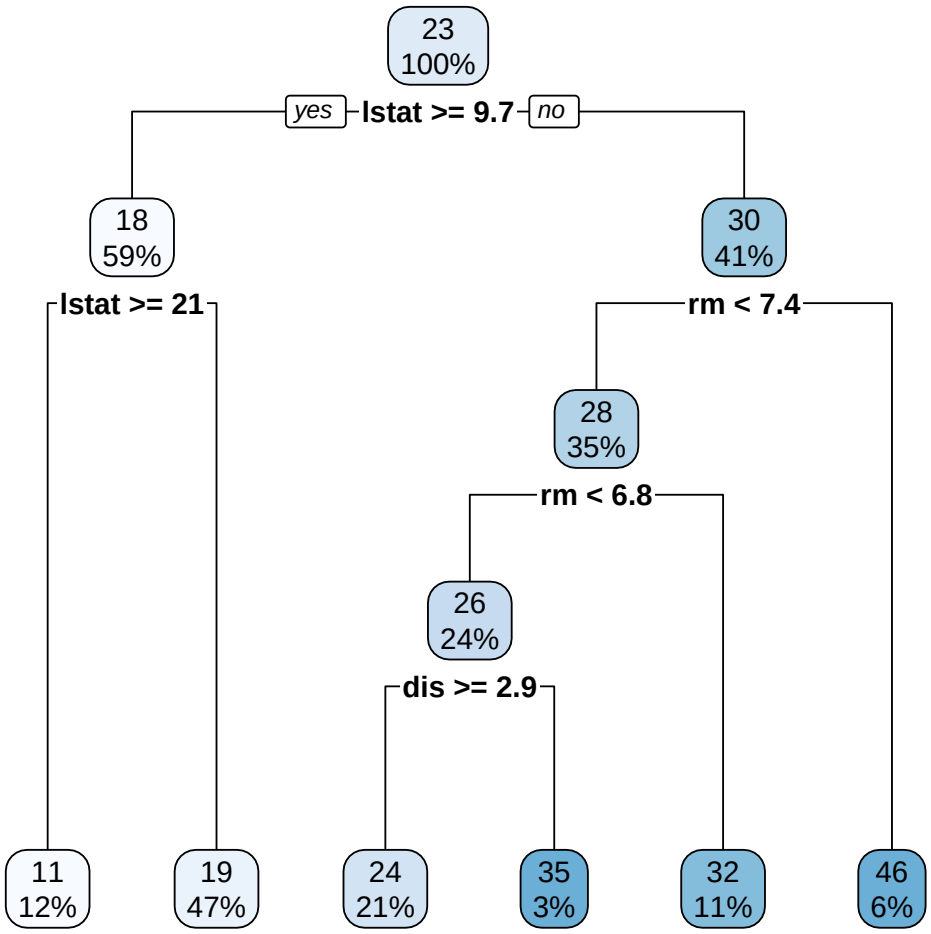
Below is a tabular summary and overleaf a plot of a simple rpart tree for the Boston data. The cp value has been chosen specifically to keep the exercise calculations simple but non-trivial.

Each node in the tree has a row in the table. The standard numbering is applied, 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[3] for the parent node and subtract the deviance in the two child nodes. Assing 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.

Calculate the variable importance for the 3 variables in the tree.

```
> tree.boston=rpart(medv~.,Boston,subset=train,cp=0.03)
> rpart.plot(tree.boston)
> tree.boston$frame
      var   n  wt        dev     yval complexity ncompete nsurrogate
1   lstat 253 253 20894.6572 22.67312 0.46257558        4          5
2   lstat 150 150  3464.7147 17.55133 0.07461842        4          4
4  <leaf>  30  30   311.8897 11.10333 0.03000000        0          0
5  <leaf> 120 120  1593.6987 19.16333 0.02162884        0          0
3      rm 103 103  7764.5843 30.13204 0.20467339        4          1
6      rm  89  89  3310.1604 27.57640 0.03919129        4          5
12    dis  61  61  1994.6223 25.52131 0.03208187        4          5
24 <leaf>  54  54   544.0083 24.32778 0.03000000        0          0
25 <leaf>   7   7   780.2743 34.72857 0.03000000        0          0
13 <leaf>  28  28   496.6496 32.05357 0.03000000        0          0
7  <leaf>  14  14   177.8436 46.37857 0.03000000        0          0
```

[3]Correction: In Lecture 1, the MSE was used to calculate the importance, but actually the Squared Error or deviance is used.


23
100%
yes
lstat >= 9.7
no
18
59%
30
41%
lstat >= 21
rm < 7.4
28
35%
rm < 6.8
26
24%
dis >= 2.9
11
12%
19
47%
24
21%
35
3%
32
11%
46
6%