Statistical learning for data science: Handout 6

Stefan Zohren

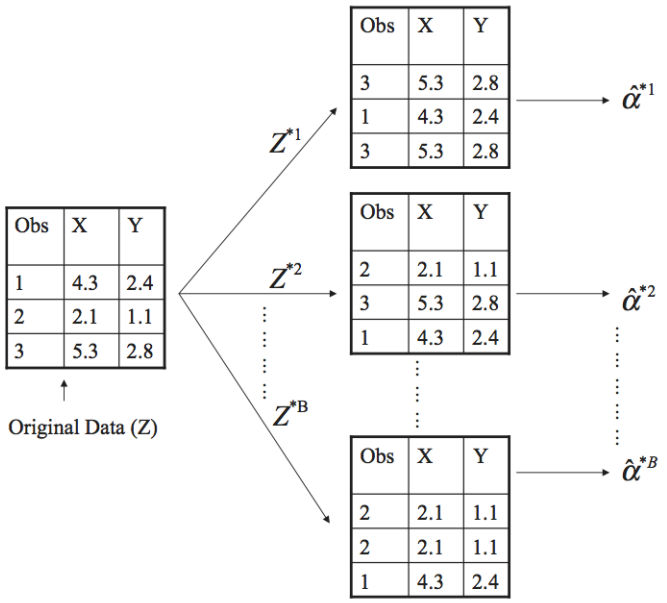
9 May 2016

The content of this lecture is mainly based on Chapter 8 of An Introduction to Statistical Learning.

A short introduction to the bootstrap

The bootstrap was introduced by Efron in 1979. The name refers to Baron Munchhausen's plan for getting himself out of a swamp by pulling himself out by his bootstraps. This analogy will become clearer later. The bootstrap is a powerful method, but at the same time it is computationally expensive which substantially delayed its adoption.

The idea of the **bootstrap** is basically simple, but can be confusion if you see it for the first time. Imagine we have n observation labelled i = 1,. . . ,n. We could create a new 'synthetic' data set by **sampling from the data with replacement**. You can think of sampling with replacement from the data as if the observations in the data were represented by balls in a bag. You take a ball out of the bag look at its label, write down which one it was (sample) and then put it back into the bag (with replacement) before repeating the whole procedure. This is illustrated in the left-hand-side of the following figure taken from An Introduction to Statistical Learning:



You see that the original data-set has n=3 observation labelled i = 1,2,3. We now sample from the data with replacement. In the first sampling Z_1 we selected third observation, then the first and then the third. You see that observation 2 is not present in the new 'synthetic' data set while observation 3 is represented twice. The whole procedure is repeated B times to generate B synthetic or bootstrapped data sets $Z_1, ..., Z_B$.

We observe that some observations are included more than once in a given new 'synthetic' data set, while others are not present, e.g. in Z_1, observation 3 appears twice, while observation 2 is not present. If the size of the original data set is large then on average approximately 2/3 of the observation will be present in a given 'synthetic' or bootstrap data set.

Exercise (optional): Prove this.

In statistical learning we generally assume that our data comes from a given distribution f, often written as an underlying true model plus noise. Each data set is then a sample from the distribution. In the cases where we simulated our own we actually know what the underlying distribution is and we can sample as many data sets as we like. In real-life situations, we cannot simply go and get more data. However, we can construct from our data an estimator of our distribution f, namely empirical distribution $\hat{f}$ obtained from our data point. Even though we do not know f we can use $\hat{f}$ to sample new data. This is exactly the bootstrap, as resampling from the data with replacement is nothing else than sampling from the empirical distribution $\hat{f}$.

The bootstrap has several important usages. For example:

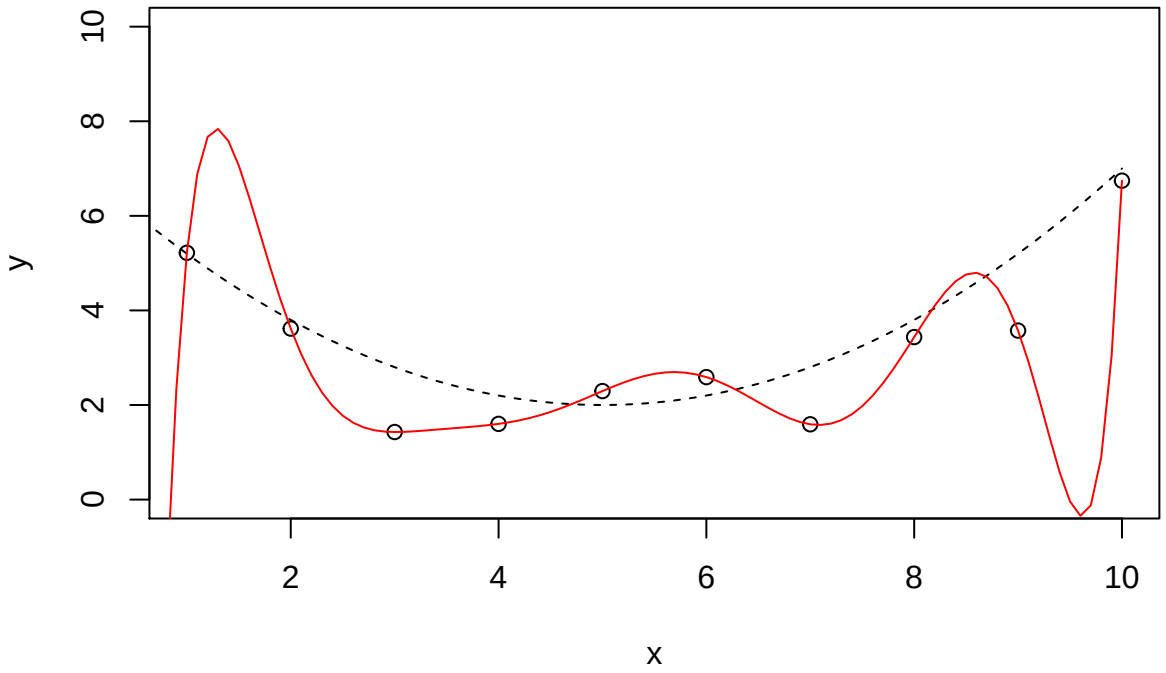
- Using bootstrap techniques one can obtain **confidence intervals of estimators** or even the entire distribution over estimators. For example in linear regression, the `lm` function provides you with confidence intervals on the estimated parameters (intercept,slopes). The result for the confidence interval is based on an analytical formula which relies on a normality assumption of the errors. In some case we do not want to make such an assumption or a similar analytical calculation might not be possible at all. In those cases we can still use the bootstrap. We simple resample a number of bootstrap data sets (often 100-1000). We then run our learning algorithm on each each of the data sets, which results in one estimate for each set, see e.g. the right-hand-side of the above figure. The we can simply calculate the standard error of those estimates to construct a confidence interval. We could even make a histogram to obtain an estimate of the entire distribution.
- Another application of the bootstrap is in prediction and in particular to **reduce the variance of the predictor by averaging various predictors**. We can train a model on each of the bootstrap data sets and then make a prediction using this model. This leads to B different predictions. We can then for example take the mean of the B predictions to obtain our final prediction. In particular, such a technique can be used to reduce the variance of the predictor. Is is also important to note that to use the bootstrap to reduce the variance of our predictor we should have a very flexible model with low bias and high variance.

The second use case will be important for the models we will be discussing in this lecture. Let us illustrate it briefly.

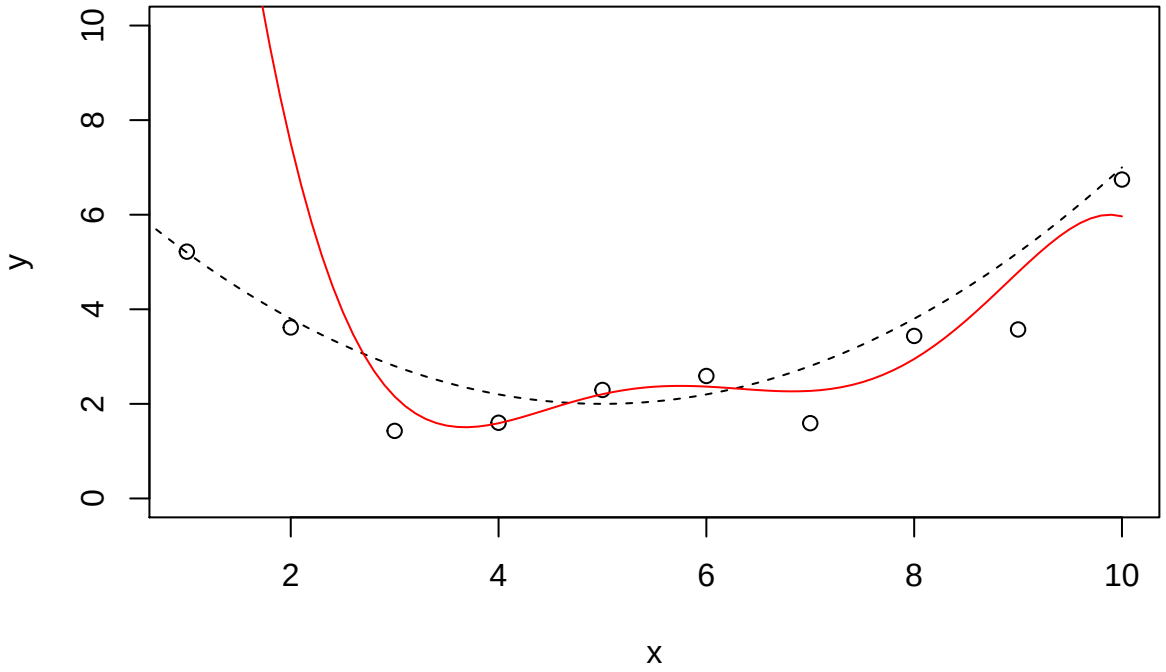
For the illustration we return to the toy example from last class:

```
set.seed(10)
x <- 1:10
f <- function(x) { 2 + 0.2*(x-5)^2 }
y <- f(x)+rnorm(10)
```

Here we plot the data, the underlying function, as well as the fit to a polynomial of order 10:



Using bootstrap techniques we can now resample from the data and repeat the above training B=100 times. The resulting predictions are averaged yielding the following plot:



The above led to a reduction in the variance of our predictor. Note however that this is only a mild improvement, which has in part to do with the fact that our original data set is only very small.

Exercise: This exercise has to do with the first use case of the bootstrap. Execute the above commands to generate the data. Using the `lm` function fit the a linear regression with y being the response variable and x the predictor. Inspect the summary output of your fit. Read off the confidence interval of the intercept and slope. Your task is to obtain the confidence interval using bootstrap techniques. Creating B=100 bootstrap data sets. You can write this procedure by hand or you use `boot()` function from the `boot` package (You will have to wrap the entire procedure you want to repeat B times into a function). Also make a histogram of your 100 estimates for the intercept and slope respectively. You can find more information in chapter 5.2 of

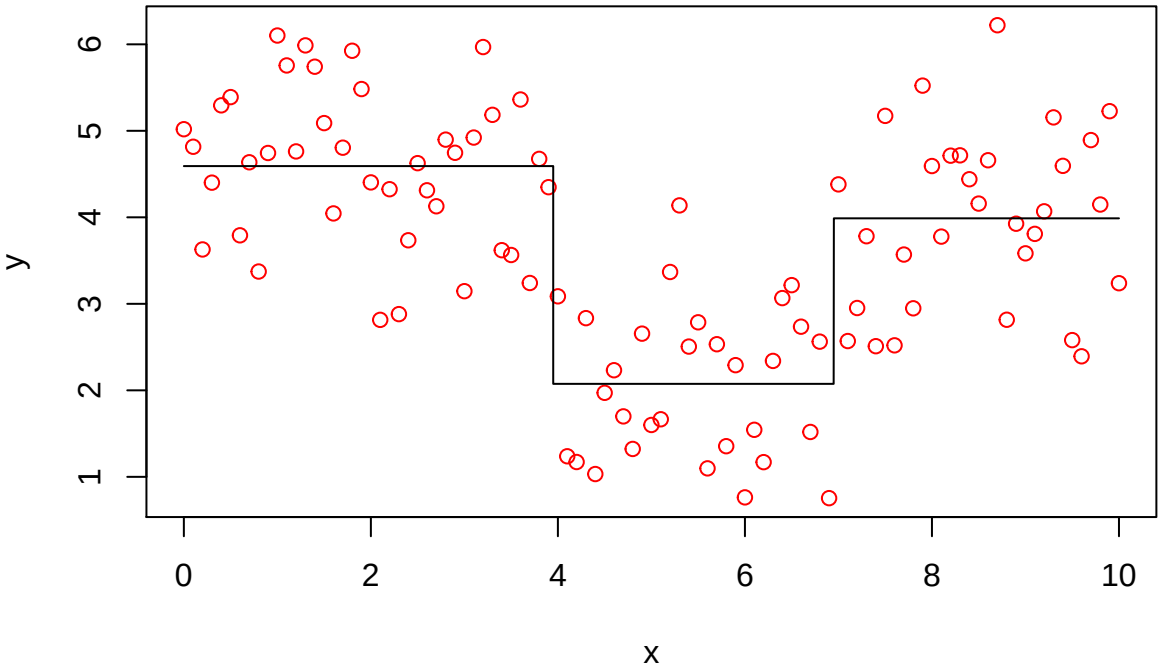
An Introduction to Statistical Learning. (**Variation:** Instead of the generated toy data, you can also use the California Housing Data.)

Bagging, random forests and boosting

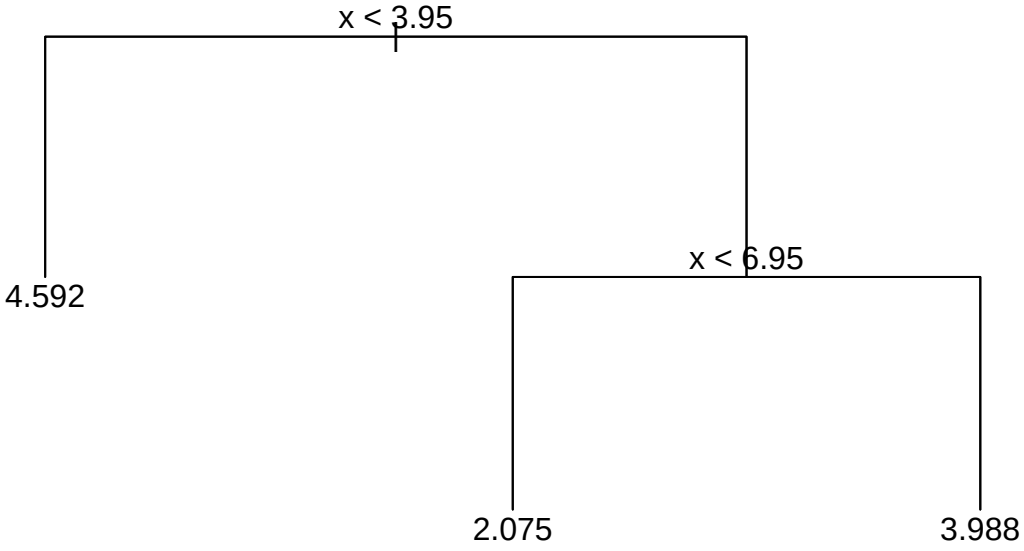
Short revision decision trees

Let us briefly revise the idea of decision trees, as all model we discuss in this lecture are tree-based models. Recall that all a tree does is fitting constant value for the response variable to different partitions of the predictor space. This is done in a greedy fashion, where we first determine the best binary split of the predictor space, then, given this split, determine the next best split etc.

Recall the example from last class which used a one-dimensional predictor space and two splits.



The resulting function can be represented by a binary decision tree as we discussed last class:



Decision trees have many advantages:

- They are very **simple** and the way prediction is done is very **clear**. This can be an advantage, especially when used in legal context, where the way the decision/prediction was made can easily be explained to a judge. This is for example used by credit card companies to determine default probabilities and potentially refuse credit based on this decision.
- Another very strong point of decision trees we have seen is that they **can be used with heterogeneous data**. We can use essentially the same model for regression and classification and furthermore the predictors can be of various types including numerical variables, categorial variables, etc.

Disadvantages of decision trees are mainly:

- Decision trees are **limited in their predictive power**.

We now see how we can use ensemble methods to improve the predictive power of decision trees, while maintaining some of their advantages.

Bagging

Bagging is nothing else than applying the idea of using the bootstrap to reduce the variance predictors to decision trees.

Mathematically speaking the estimator for the response variable obtained from a decision tree with M leaves is:

$$\hat{y} = \hat{f}(x) = \sum_{i=1}^{M} \hat{a}_i 1_{x \in \hat{R}_i}$$

Here $\hat{R}_1, ..., \hat{R}_m$ is the optimal partition of the predictor space and $\hat{a}_i = \text{mean}(\{\text{all } y_j \text{ s.t. } x_j \in R_i\})$ is the predicted response in this partition. In the tree shown above we have $\hat{R}_1 = \{x|x < 3.95\}$, $\hat{R}_2 = \{x|3.95 < x < 6.95\}$, $\hat{R}_3 = \{x|x > 6.95\}$, as well as $\hat{a}_1 = 4.592$, $\hat{a}_2 = 2.075$, $\hat{a}_3 = 3.988$.

We now repeat the training of a decision tree on B bootstrap data sets, resulting in B predictors $\hat{f}^b(x)$, $b = 1, ..., B$. We can then calculate the final predictor by **averaging**:

$$\hat{f}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x)$$

The above determines the final predictor when applying **bagging to regression trees** where the response variable is numerical allowing us to average it. In the case of **bagging for classification trees** we would instead obtain B predictions for the label at a given point which generally we would aggregate using **majority voting**.

There is a very nice aspect of the bootstrap in the context of bagging, that we have not yet stressed. We have seen earlier that on average only 2/3 of the data are present in a given bootstrap data set. We can use this fact in a very nice way, namely, we can use the remaining 1/3 of the data and treat it as a cross-validation set for this bootstrap sample. The cross-validation error calculated using this set is called the **out-of-bag-error**. The nice thing is that we get it basically for free!

We now illustrate with the California Housing Data how to do bagging in `R`. We import the data and split it in training and test set as follows:

```
set.seed(10)
HousingData <- read.csv("California-Housing.txt")
n <- nrow(HousingData)
trainIndices <- sample(1:n, n*0.8)
HousingData.train <- HousingData[trainIndices,]
HousingData.test <- HousingData[-trainIndices,]
```

In the previous class we fitted a regression tree to predict `MedianHouseValue` as a function of all predictors:

```
library(tree)
reg.tree <- tree(MedianHouseValue ~. , data = HousingData.train)
```

The test MSE was obtained as:

```
pred.tree = predict(reg.tree, newdata = HousingData.test)
mean((HousingData.test$MedianHouseValue-pred.tree)^2)
```

```
## [1] 0.5040067
```

Bagging is implemented in the package `randomForest` (together with random forests which we will discuss next). We now fit a bagging model using B=100 bootstrap data sets, each of which will have a regression tree trained on it:

```
library(randomForest)
# To do bagging we miust set mtry = numer of predictors, 8 in our case (more later)
reg.bag = randomForest(MedianHouseValue ~. ,data=HousingData.train, mtry=8,
                       importance =TRUE,ntree=100)
```

We can now calculate the test error:

```
pred.bag = predict(reg.bag, newdata = HousingData.test)
mean((HousingData.test$MedianHouseValue-pred.bag)^2)
```

```
## [1] 0.2123544
```

Exercise: Repeat the above analysis. In particular, also look at the outputs and summaries of each of the expressions, e.g. inspect the output of `pred.bag` and `summary(pred.bag)` etc.

We see the the improvement of the test MSE which was about 0.5 for a single regression tree and 0.4 for the lasso and rigid regression.

Random forests

Random forests are very similar to bagging. We can think of them as bagging with a small tweak. As we have seen above, in bagging we use the bootstrap to generate several bootstrap sample of the data and train a decision tree on each of the bootstrap data sets (a forest of trees, one for each bootstrap sample). We then average the predictors to get the final predictor. A potential problem is that the **individual predictors in bagging can be highly correlated**. For example, if there are a few predictors which explain the response variable very well, we will always make splits using those predictors first and thus the upper part of our tree will most likely be the same each time. This creates a high degree of correlation in our predictors

(decision trees) which tend to be very similar. We are also let to focus highly on the dominant predictors and ignore the less dominant predictors. Since the decision trees are correlated it is harder to reduce the variance by averaging. One goal is thus to make the individual decision trees trained on the bootstrap data sets less correlated. This is done in **random forests** as follows: instead of using all predictors each time we train a decision tree on a bootstrap data set, we allow each individual training procedure to only use a smaller random set of the predictors. In practice, each individual tree is only trained using a fraction of the size (number of predictors used) $= \sqrt{(\text{total number of predictors})}$ sampled from the entire set of predictors. When the number of predictors used is equal to the total number of predictors, then the random forest is equivalent to bagging. The number of predictors used is set in the `randomForest` function using the attribute `mtry`. We already used this function and the attribute above when doing bagging, only that we set `mtry=8` which was the total number of predictors. We can now train a random forest in the exact way as we did before, by simply reducing the number of predictors used.

```
reg.forest = randomForest(MedianHouseValue ~. ,data=HousingData.train, mtry=3,
                          importance =TRUE,ntree=100)
pred.forest = predict(reg.forest, newdata = HousingData.test)
mean((HousingData.test$MedianHouseValue-pred.forest)^2)
```

```
## [1] 0.2033348
```

We see a slight improvement in the error as compared to the MSE for bagging. This reduction can be more pronounced when having more predictors.

- Note that both in bagging as well as in random forests we should not prune the trees. Instead we train large flexible trees with individual low bias and high variance and use the averaging procedure to reduce the variance when going to the predictor.
- Considering the above comment, we observe that bagging and random forests do not require sophisticated fine-tuning of hyper-parameters: We would always want to use as many predictors as possible, as many trees as is computational reasonable and the optimal number of predictors used in random forests is the square root of the total number of predictors.

Boosting

Boosting (when applied to trees) is a method similar to bagging, where we train many decision trees on different variations of the original data set. However, in boosting we do not use the bootstrap to create new data sets, instead, **in boosting new data sets are created in a sequential manner**.

The detailed algorithms is explained in Algorithm 8.2 of An Introduction to Statistical Learning. We only explain the basic idea behind the algorithm here. The hyper-parameters of the model are the number of splits d and a shrinkage parameter λ.

The boosting algorithm then works as follows:

- Fit a decision tree to the data with the given number of splits
- Update the current predictor by adding the new predictor weighted by the shrinkage
- Update the data by 'subtracting' the variance explained by the new tree weighted by the shrinkage
- Repeat

When doing the above algorithm one usually choose a small value for d. The idea is that one learns many times a little at a time, always updating the data. However, even though each individual tree is small (sometimes only a single stem) the resulting **ensemble** of trees is a powerful predictor.

In R boosting is implemented in the `gbm` package through the `gbm()` function. The number of splits is given by the attribute `interaction .depth` and the shrinkage λ by the attribute `shrinkage`. Here we show an implementation of boosting in R. In the example we simply set $d = 3$ and $\lambda = 0.2$. In real-life you should use cross-validation to obtain the best values.

```
library(gbm)
reg.gbm <- gbm(MedianHouseValue ~. ,data=HousingData.train,
               distribution= "gaussian", n.trees=2000,
               interaction.depth = 3, shrinkage = 0.2, verbose =F )
pred.gbm = predict(reg.gbm, newdata = HousingData.test,n.trees=2000)
mean((HousingData.test$MedianHouseValue-pred.gbm)^2)
```

```
## [1] 0.183273
```

We see that the error is yet another improvement over the best previous error obtained using random forests.

As a final remark, note that boosting can also be applied to other models apart from decision trees, see The Elements of Statistical Learning for more details.

Case study: Heart data (continued)

We continue our case study on the Heart data set. Recall that the data set has 303 observations. The response variable is `AHD`, heart disease, yes or no. There are 13 predictors include factors such as `Sex`, `Age` etc.

We repeat the sequence of commands from last lecture to import the data, process it and split it in training and test set:

```
heart <- read.csv("Heart.csv")
heart$X <- NULL
heart$AHD <- factor(heart$AHD)
heart$Sex <- factor(heart$Sex)
heart$Thal <- factor(heart$Thal)
heart$ChestPain <- factor(heart$ChestPain)
set.seed(10)
n <- nrow(heart)
trainIndices <- sample(1:n, n*0.8)
heart.train <- heart[trainIndices,]
heart.test <- heart[-trainIndices,]
n <- nrow(heart.train)
trainIndices <- sample(1:n, n*0.8)
heart.train.tr <- heart.train[trainIndices,]
heart.train.cv <- heart.train[-trainIndices,]
```

Exercise: Train a bagging model, a random forest as well as a boosting model on the data. Compare the models in performance. For boosting you will have to to proper cross-validation to fine-tune the hyper-parameters. We already split off a CV set above.

Exercise: Create a plot of MSE as a function of number of trees used for each of the three different learning algorithms. Compare them using a single plot.