Machine Learning

Lecture 3: Regularization, Decision trees, Random Forests

Mohamad GHASSANY

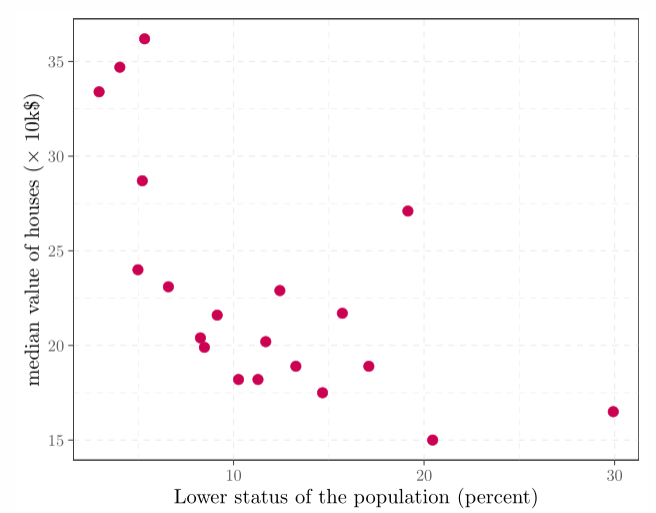
EFREI Paris

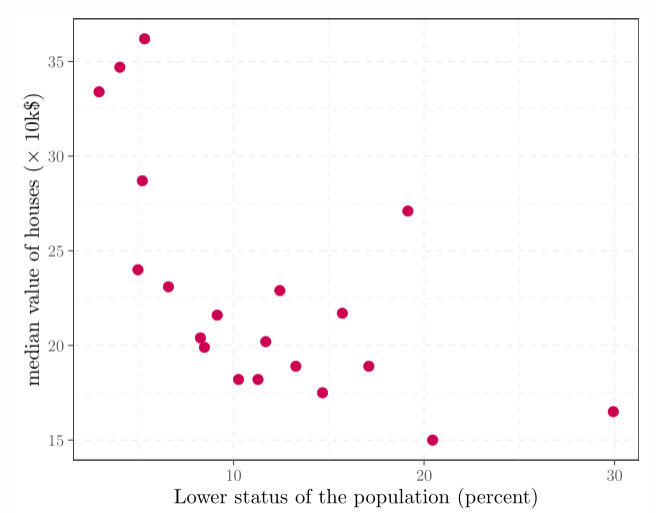
Regression and Classification

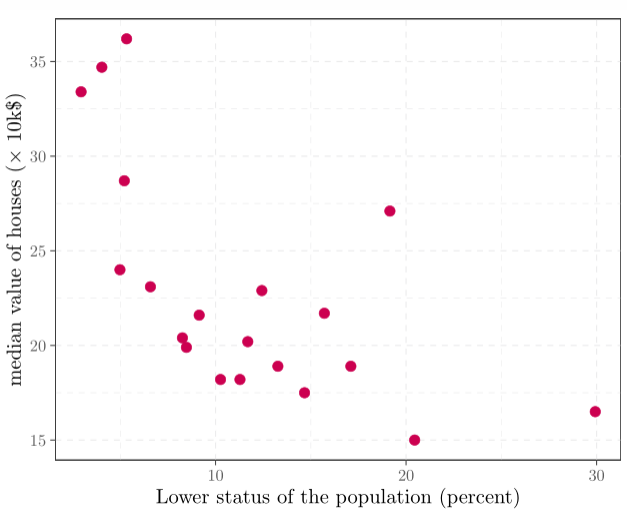
- Supervised learning
- Target variable type
- Hypothesis
- Cost function
- Optimization
- Sampling

The problem of overfitting

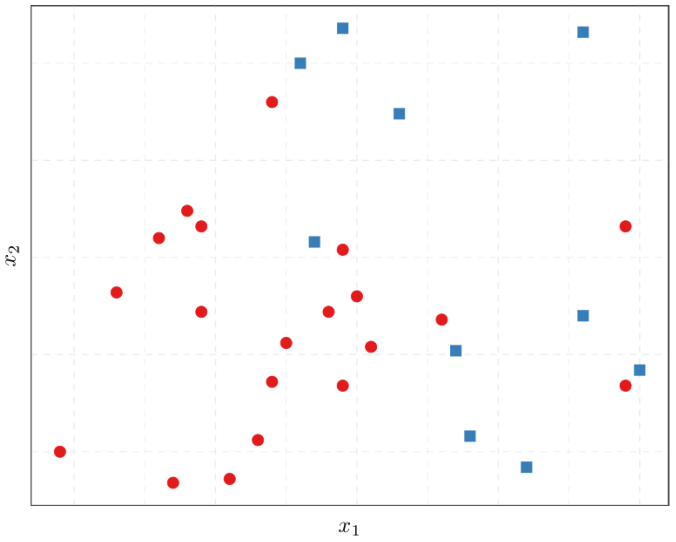
Overfitting: Regression example

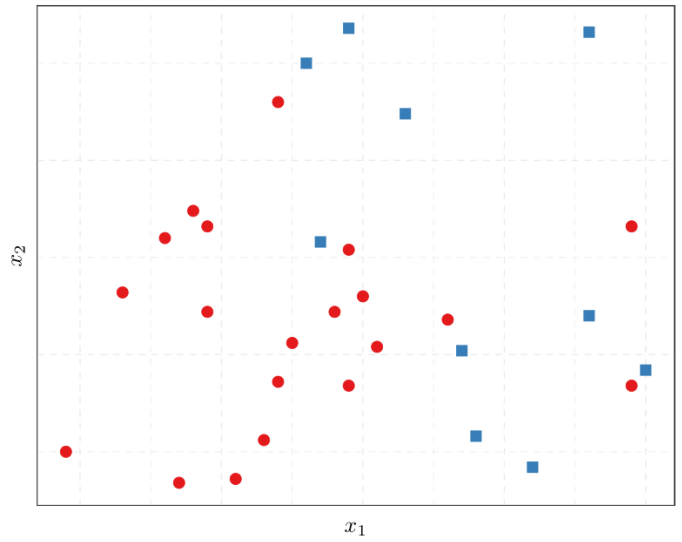


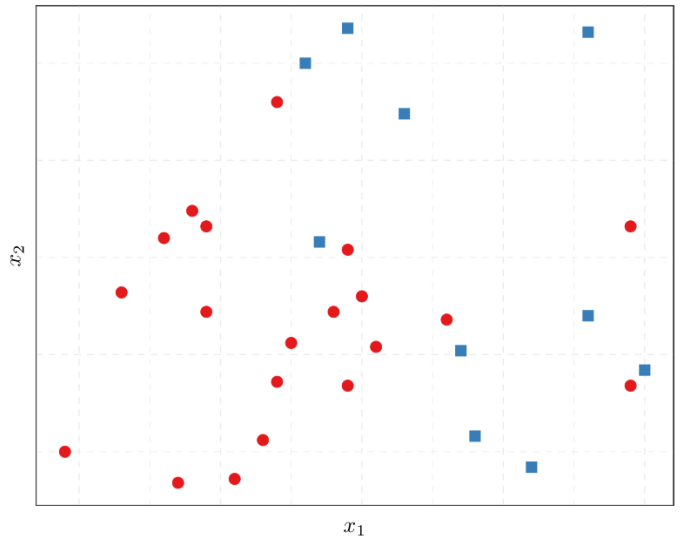




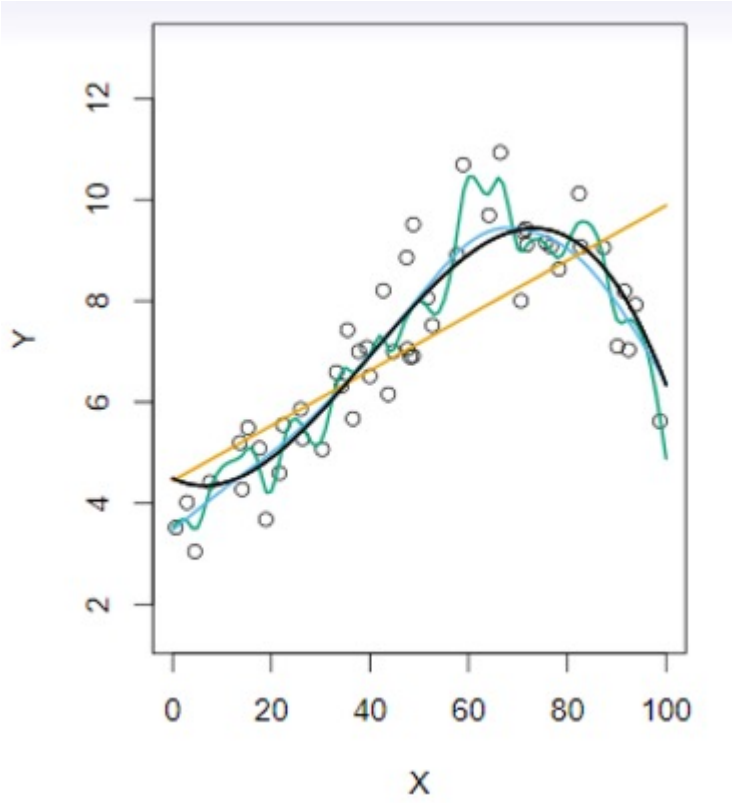
Overfitting: Logistic Regression example



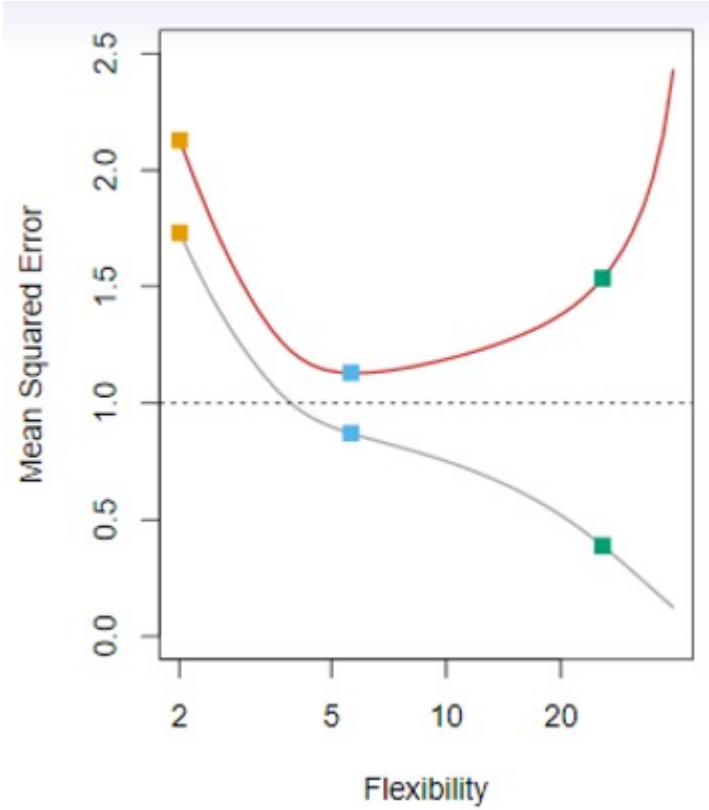




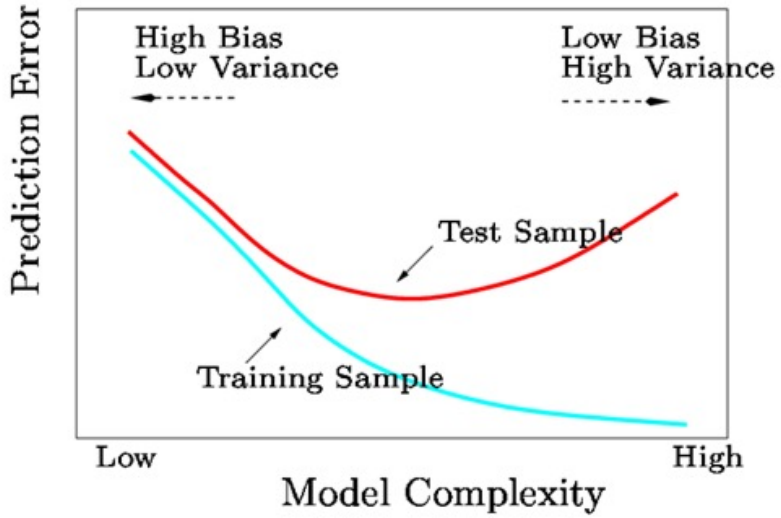
Underfitting/Overfitting Trade off



Black: Truth
Orange: Linear Estimate
Blue: smoothing spline
Green: smoothing spline (more flexible)



RED: Test MES
Grey: Training MSE
Dashed: Minimum possible test MSE (irreducible error)



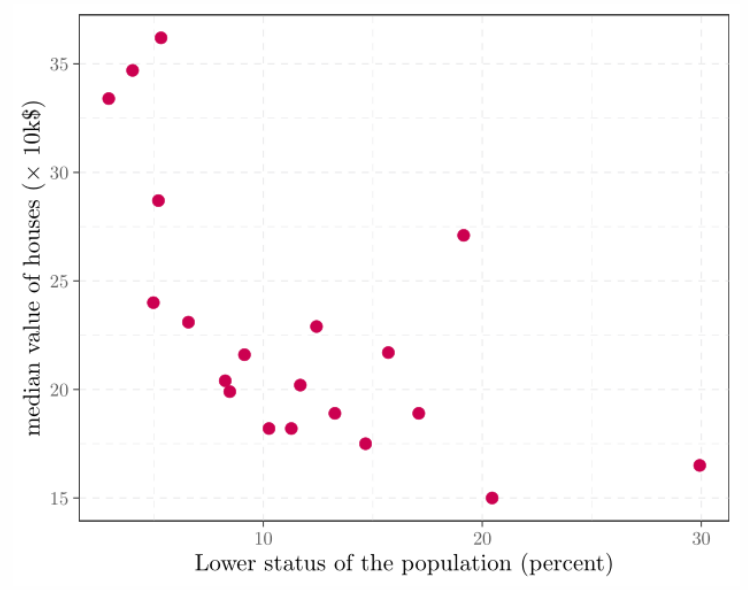
We must always keep this picture in mind when choosing a learning method. More flexible/complicated is not always better!

Adressing overfitting

- Options:

1. **Reduce number of features**
   - Manually
   - Model selection
2. **Regularization**
   - Keep all the features, but reduce magnitude/values of parameters ω_j
   - Works well when we have a lot of features, each of which contributes a bit to predicting y
   - It may not be immediately obvious why such a constraint should improve the fit, but it turns out that shrinking the coefficient estimates can significantly reduce their variance

Regularization: Cost function

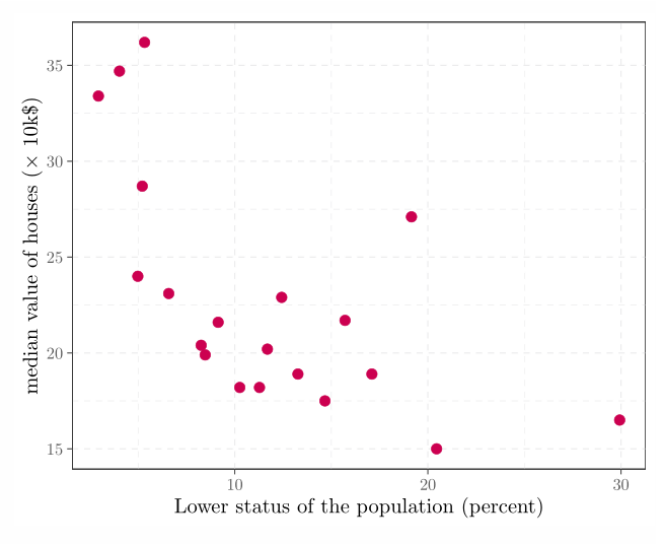


$$J(\omega) = \frac{1}{2n}\sum_{i=1}^{n}\left(f_\omega\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Choice of λ

- $J(\omega) = \frac{1}{2n}\left[\sum_{i=1}^{n}\left(f_\omega\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda \sum_{j=1}^{p} w_j^2\right]$

- Choice of λ
  - What happens if λ is large ?



Ridge Regression and Lasso

- Ridge Regression

$$J(\omega) = \frac{1}{2n}\sum_{i=1}^{n}\left(f_\omega\left(x^{(i)}\right) - y^{(i)}\right)^2$$

- Lasso

$$J(\omega) = \frac{1}{2n}\sum_{i=1}^{n}\left(f_\omega\left(x^{(i)}\right) - y^{(i)}\right)^2$$

- Neither ridge regression nor the lasso will universally dominate the other
- In general, one might expect the lasso to perform better when the response is a function of only a relatively small number of predictors.
- However, the number of predictors that is related to the response is never known *a priori* for real data sets.
- A technique such as cross-validation can be used in order to determine which approach is better on a particular dataset.
- **Cross-validation**: we choose a grid of λ values, and compute the cross-validation error rate for each value of λ. We then select the value for which the cross-validation error is smallest.

Regularization for Linear Regression

Regularized Linear Regression: GD

- $J(\omega) = \frac{1}{2n}\left[\sum_{i=1}^{n}\left(f_\omega\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda \sum_{j=1}^{p} w_j^2\right]$
- $\min_{\omega} J(\omega)$

- Using **GD**:
  - `initialize` ω_j `randomly`
  - `repeat until convergence{`
    - $\omega_0^{new} = \omega_0^{old} - \alpha \frac{1}{n}\sum_{i=1}^{n}\left(f_\omega\left(x^{(i)}\right) - y^{(i)}\right)$
    - $\omega_j^{new} = \omega_j^{old} - \alpha \ \frac{1}{n}\sum_{i=1}^{n}\left(f_\omega\left(x^{(i)}\right) - y^{(i)}\right).x_j^{(i)}$ simultaneously for every $j = 1, \dots, p$

  `}`

- $\omega_j^{new} = \omega_j^{old}\left(1 - \alpha\frac{\lambda}{n}\right) - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(f_\omega\left(x^{(i)}\right) - y^{(i)}\right).x^{(i)}$

- $\left(1 - \alpha\frac{\lambda}{n}\right)$ will always be less than 1

Regularized Linear Regression: OLS

- $J(\omega) = \frac{1}{2n}\left[\sum_{i=1}^{n}\left(f_\omega\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda \sum_{j=1}^{p} w_j^2\right]$
- $\min_{\omega} J(\omega)$

- $\omega = (X'X \qquad\qquad)^{-1}X'y$

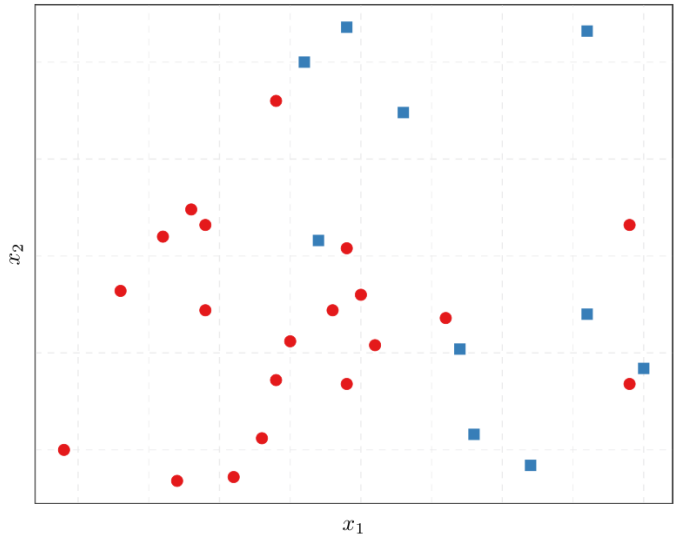
- Using regularization takes care also of non-invertibility problem

Regularization for Logistic Regression

Regularized Logistic Regression

- $J(\omega) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} \log f_\omega\left(x^{(i)}\right) + \left(1 - y^{(i)}\right)\log\left(1 - f_\omega\left(x^{(i)}\right)\right) + \frac{\lambda}{2n}\sum_{j=1}^{p} \omega_j^2$

- $\min_\omega J(\omega)$



- Using **GD**:
  - `initialize` ω_j `randomly`
  - `repeat until convergence{`
    - $\omega_0^{new} = \omega_0^{old} - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(f_\omega\left(x^{(i)}\right) - y^{(i)}\right)$
    - $\omega_j^{new} = \omega_j^{old} - \alpha[\frac{1}{n}\sum_{i=1}^{n}\left(f_\omega\left(x^{(i)}\right) - y^{(i)}\right).x^{(i)} + \frac{\lambda}{n}\omega_j]$ simultaneously for every $j = 1, \dots, p$

  `}`

Data modelling vs Algorithmic modelling

Data modelling vs Algorithmic modelling

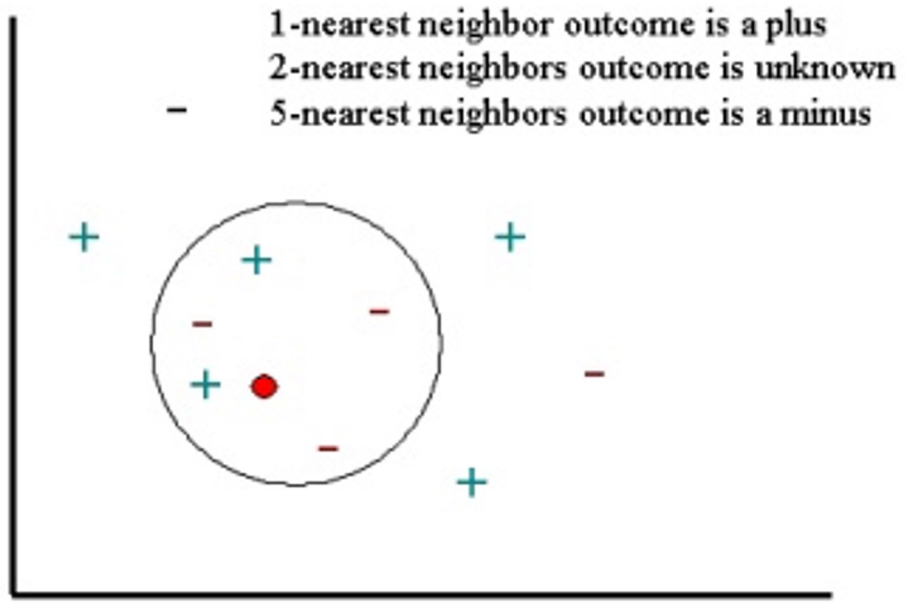
In Statistical Learning:

- Data modelling
  - Prior knowledge of f
  - E.g. $f(x) = \omega' x$
  - E.g. Multiple Regression Logistic Regression, Discriminant Analysis, etc..
- Algorithmic modelling
  - No constraints
  - A set of algorithmic instructions that relate the independent and dependent variables
  - E.g. , K-Nearest Neighbours (kNN), Decision Trees (DT), Random Forests (RF), NN..

K-Nearest Neighbors (kNN)

K-Nearest Neighbors (kNN)

- A non parametric method used for regression and classification.
- The kNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.



- Need to choose only the number of nearest neighbors and a distance

kNN Algorithm

- For **Regression**:
  - For every test sample, see the neighbors labels and apply a vote!
  - Input: Traning set and Test set, a distance d, number of neighbors k
  - **For** every test sample **do**:
    - Calculate the distance to every training sample
    - Choose the k nearest neighbors w.r.t. distance d
    - Assign the sample to the **average of the target values** of the k neighbors
  - **End for**
- For **Classification**:
  - For every test sample, see the neighbors labels and apply a vote!
  - Input: Traning set and Test set, a distance d, number of neighbors k
  - **For** every test sample **do**:
    - Calculate the distance to every training sample
    - Choose the k nearest neighbors w.r.t. distance d
    - Assign the sample to the **most frequent class** of the k neighbors
  - **End for**

kNN pros and cons

- Pros:
  - Simple
  - Do like neighbors
  - Sometimes effective
- Cons:
  - Need to choose k
  - Heavy calculations if n or p are large

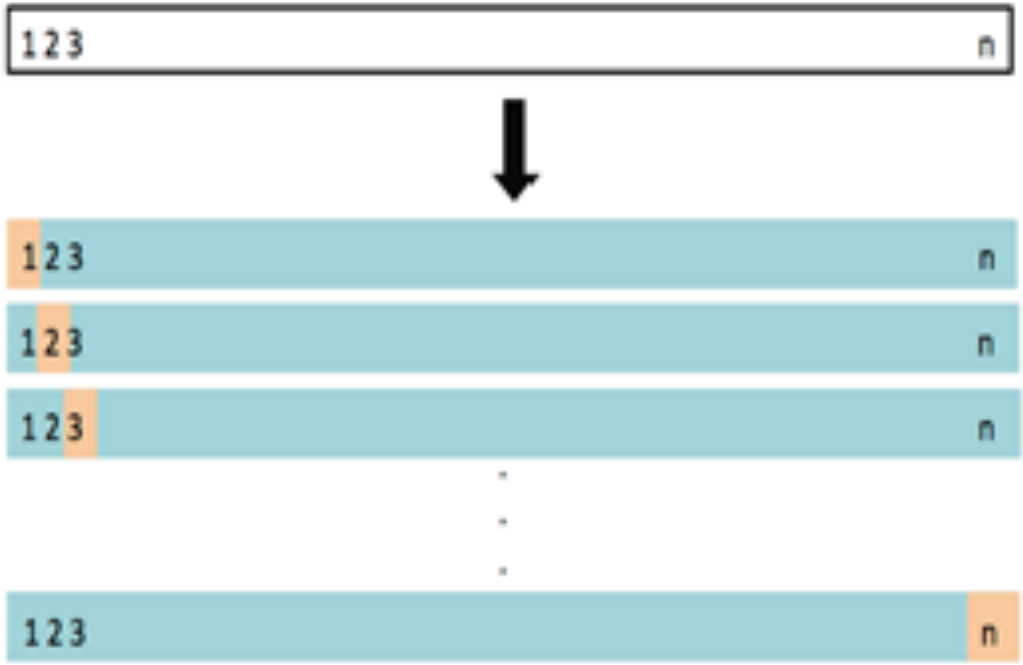
Cross Validation

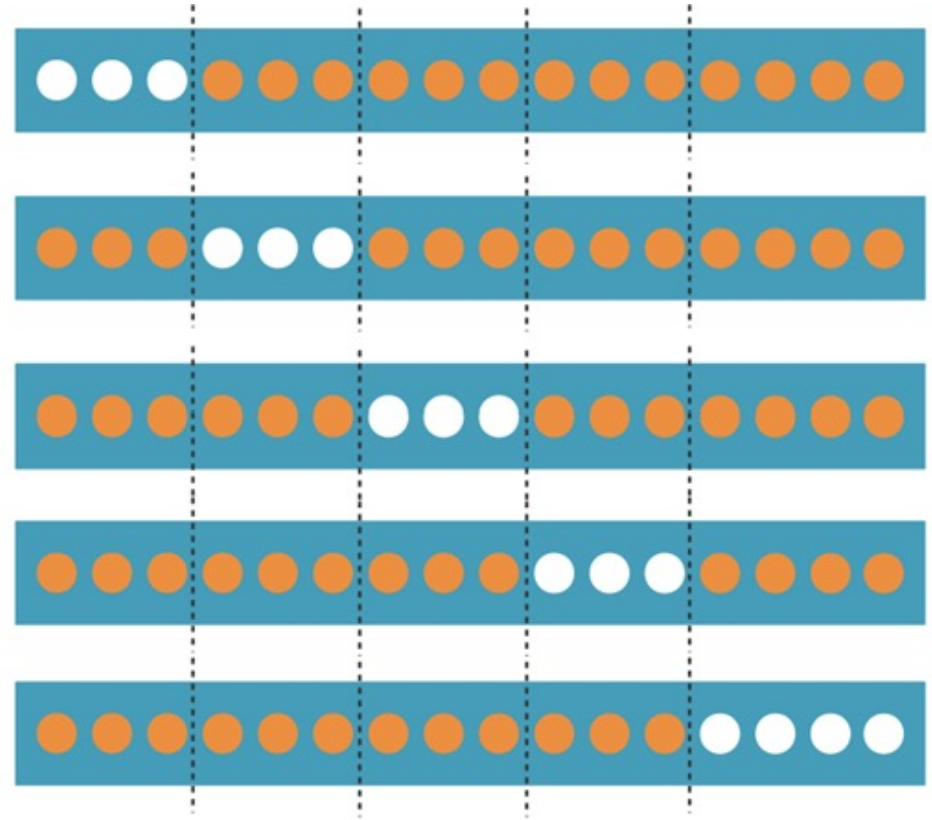
Cross Validation

- Cross-validation is **a resampling procedure used to evaluate machine learning models on a limited data sample**. ... That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

- The general procedure is as follows:
  - Shuffle the dataset randomly.
  - Split the dataset into k groups
  - For each unique group:
    - Take the group as a hold out or test data set
    - Take the remaining groups as a training data set
    - Fit a model on the training set and evaluate it on the test set
    - Retain the evaluation score and discard the model
  - Summarize the skill of the model using the sample of model evaluation scores

Cross Validation



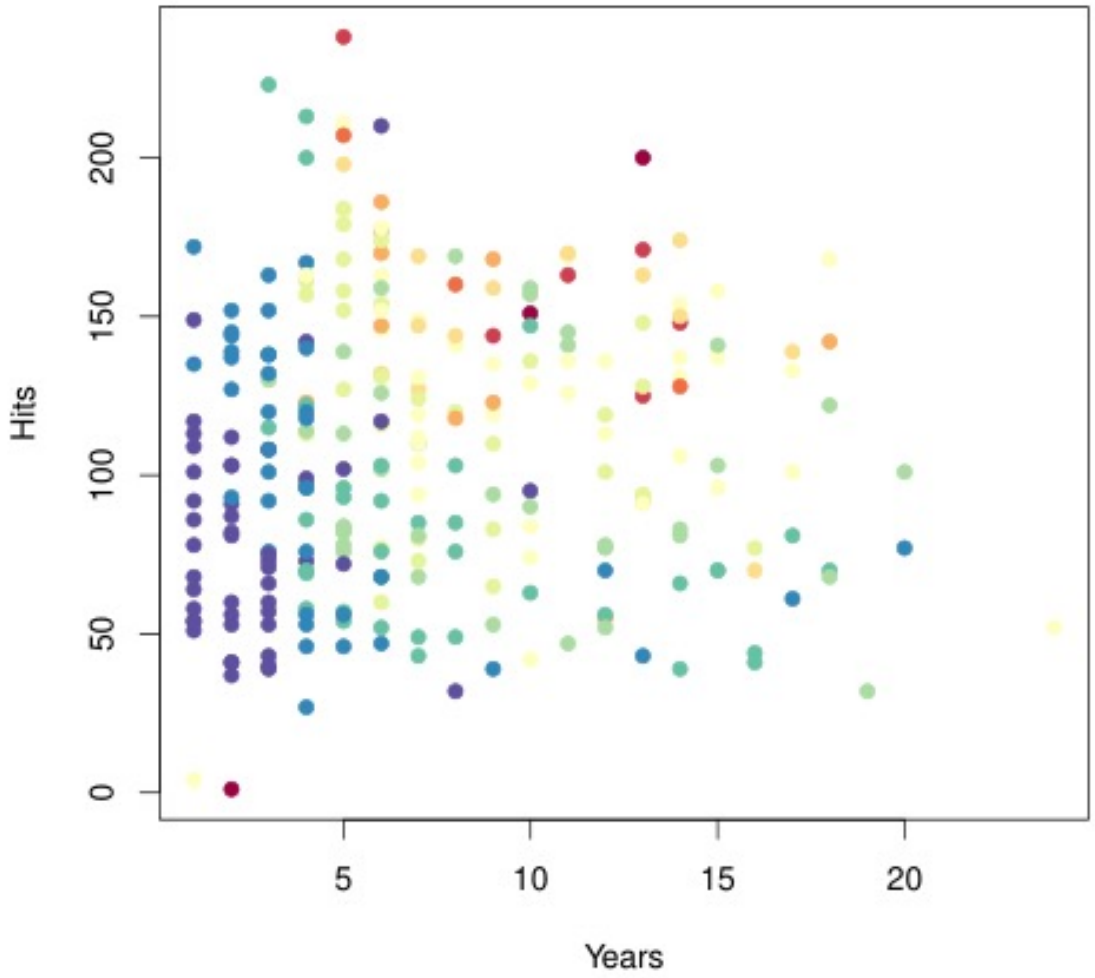


Decision Trees

Decision Trees

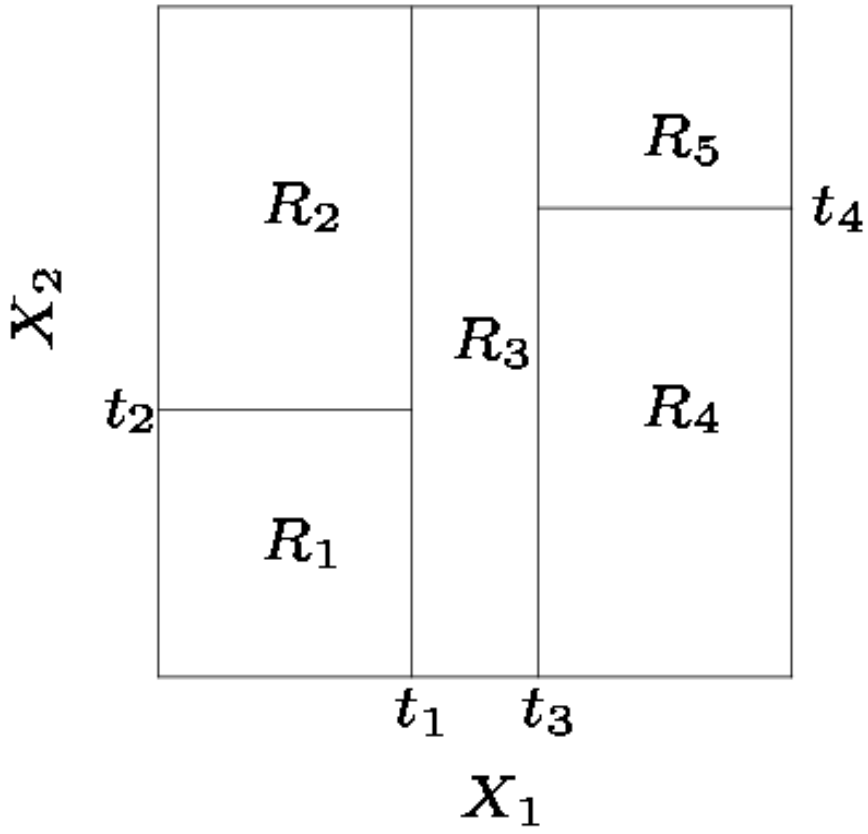
- Principle: think dimension by dimension. Maybe the nearest to human way of decision taking
  ⇒ *stratifying* or *segmenting* the predictor space into a number of simple regions
- Trees can be used in both: Regression and Classification
- Features can be continuous or discrete
- Trees are easily interpretable, but typically they are not competitive with the best supervised learning approaches in terms of prediction accuracy
- Hence we also discuss **bagging**, **random forests**, and **boosting**. These methods grow multiple trees which are then combined to yield a single consensus prediction ⇒ performance improvement but loss in interpretation.

Decision Trees: Intuition

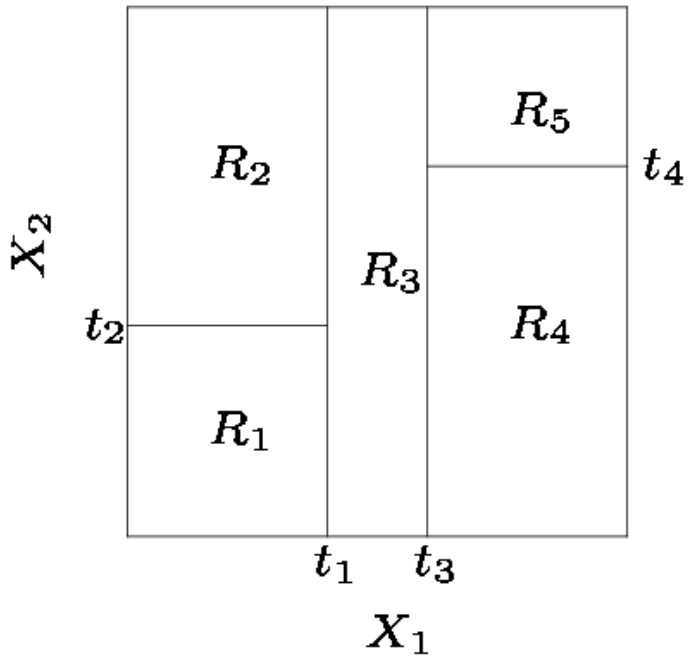


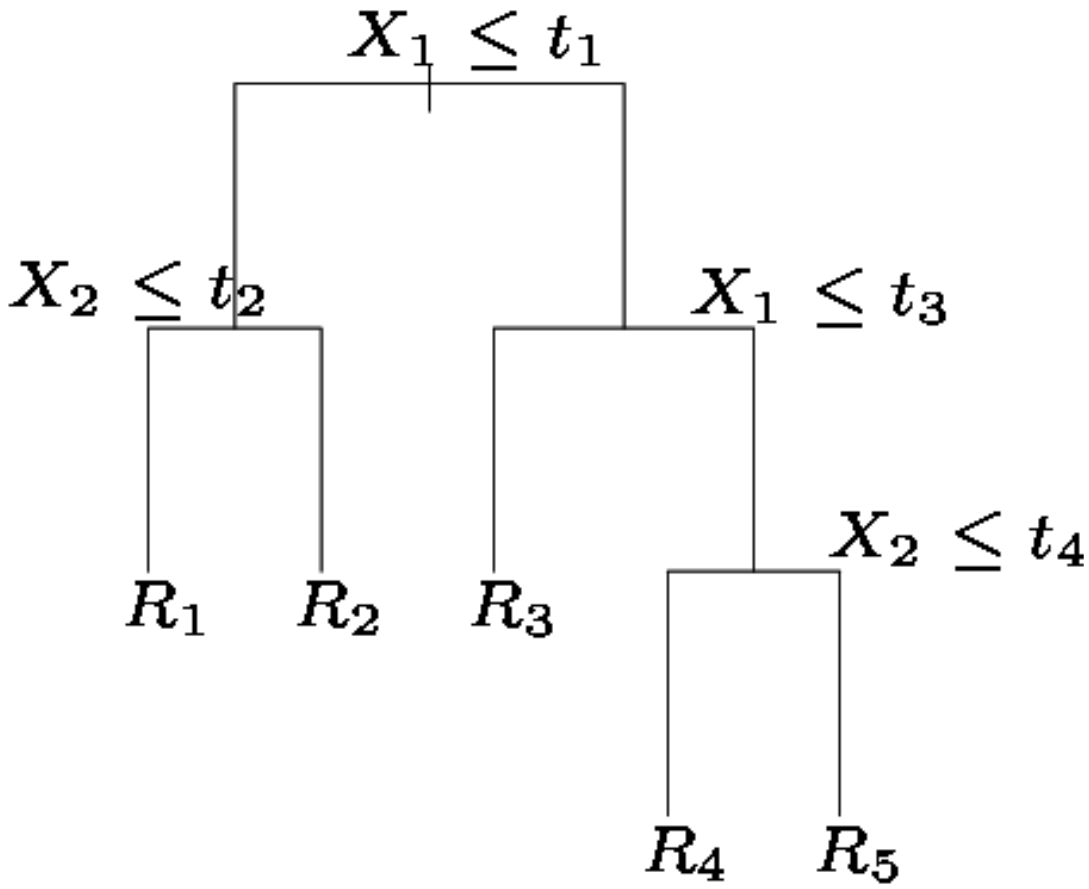
Decision Trees: principle

- Generally we create the partitions by iteratively splitting one of the X variables into two regions
  1. First split on X1=t1
  2. If $X_1<t_1$, split on $X_2=t_2$
  3. If $X_1>t_1$, split on $X_1=t_3$
  4. If $X_1>t_3$, split on $X_2=t_4$



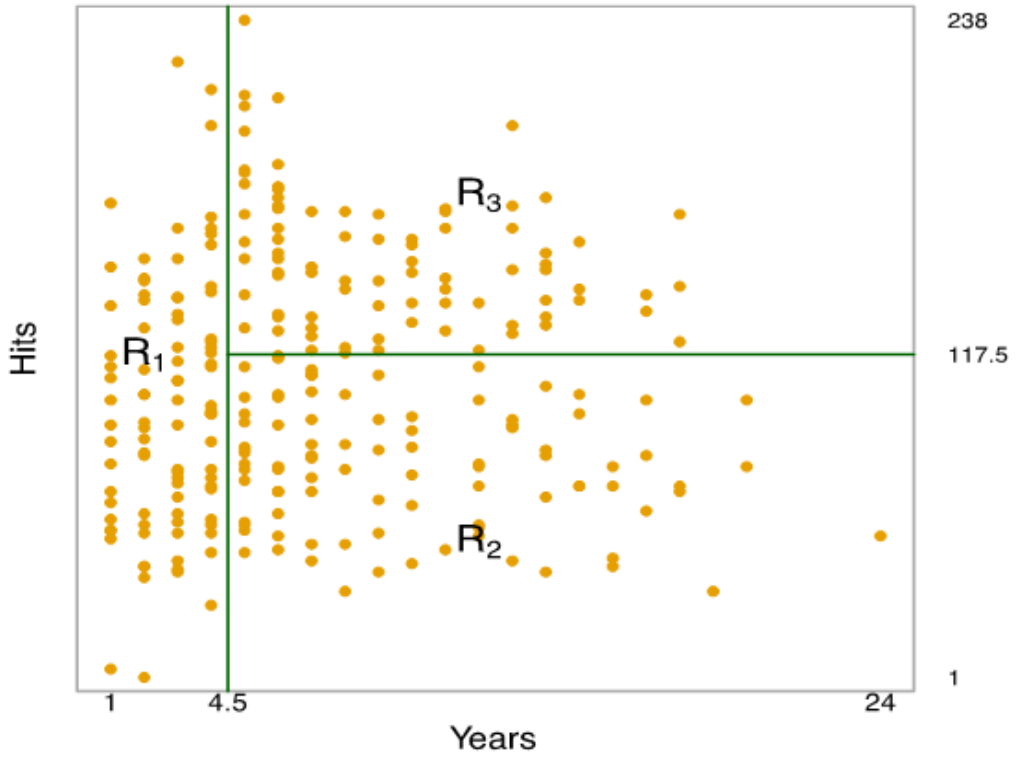
Decision Trees: principle

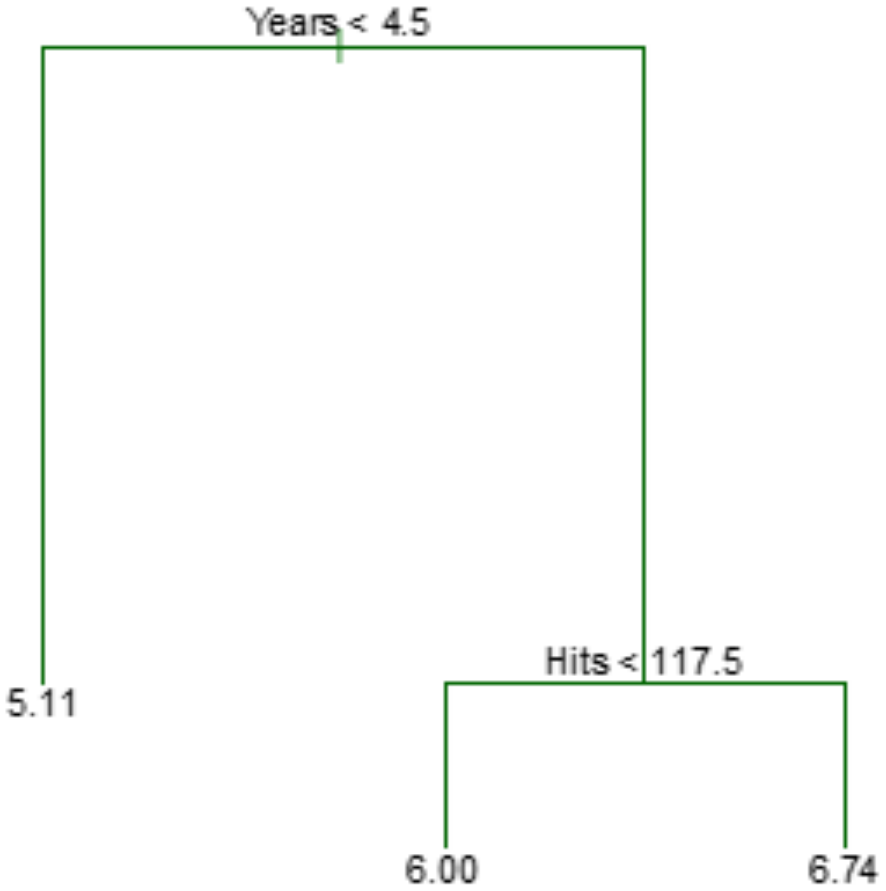




- When we create partitions this way we can always represent them using a tree structure.
- This provides a very simple way to explain the model to a non-expert i.e. your boss!

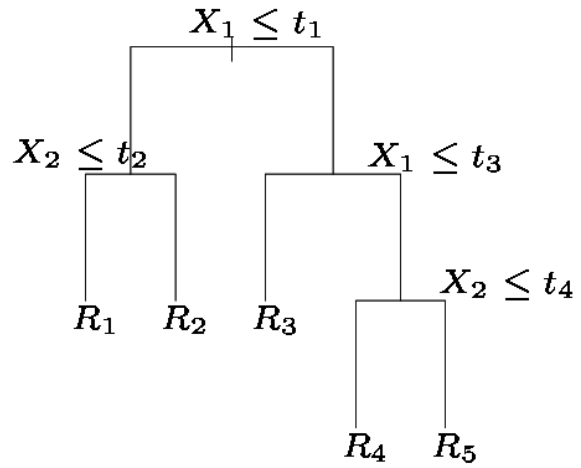
Decision Tree for Regression

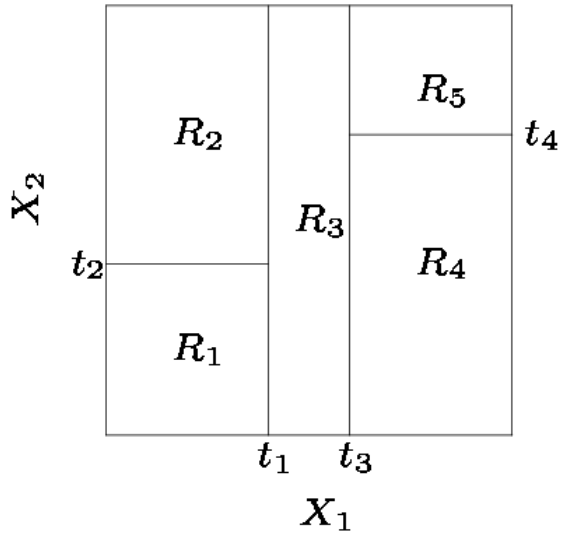




Decision Trees: Terminology

- In keeping with the *tree* analogy, the regions R1, R2, and R3 are known as *terminal nodes*
- Decision trees are typically drawn *upside down*, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as *internal nodes*





Important questions

- What are the predicted values ?
- How do we cut ?

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model.
- The goal is to find boxes R1, . . . , RJ that minimize $\sum_{\{j=1\}}^{J} \sum_{\{i \in R_j\}} \left(y_i - \hat{y}_{R_j} \right)^2$ where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

- It is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a *top-down, greedy* approach that is known as recursive binary splitting.

- Predictions: We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.

Pruning

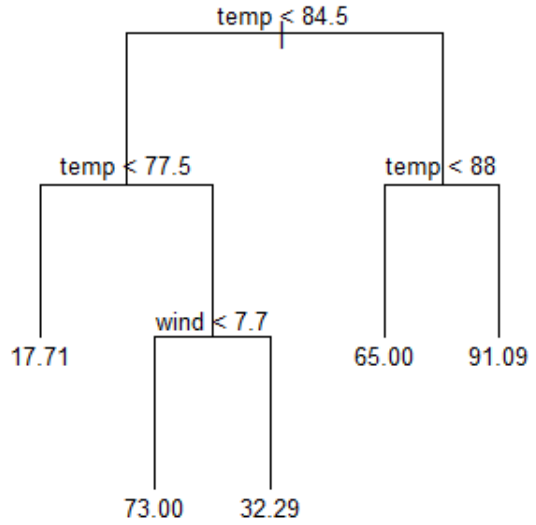
- The process described before may produce good predictions on the training set, but is likely to overfit the data
- ⇒ A smaller tree with fewer splits (that is, fewer regions R1, . . . , RJ)
- Strategy: grow a very large tree T_0 , and then *prune* it back in order to obtain a *subtree*
- *Cost complexity pruning* is used to do this

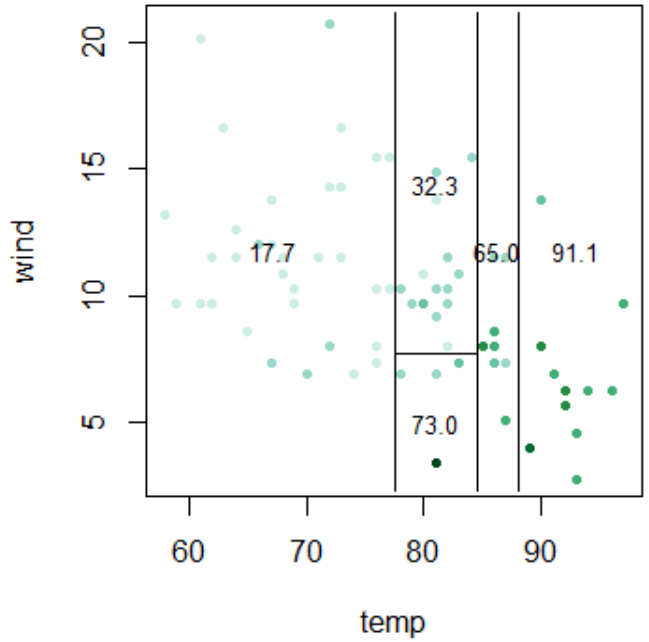
- A tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data
- We select an optimal value of α using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to the optimal value of α

Pruning

- Consider the `ozone` data set[1]. The data set consists of 111 observations on the following variables:
  - `ozone` : the concentration of ozone in ppb
  - `radiation`: the solar radiation (langleys)
  - `temperature` : the daily maximum temperature in degrees F
  - `wind` : wind speed in mph

```
##   ozone radi temp wind
## 1    41  190   67  7.4
## 2    36  118   72  8.0
## 3    12  149   74 12.6
## 4    18  313   62 11.5
## 5    23  299   65  8.6
```

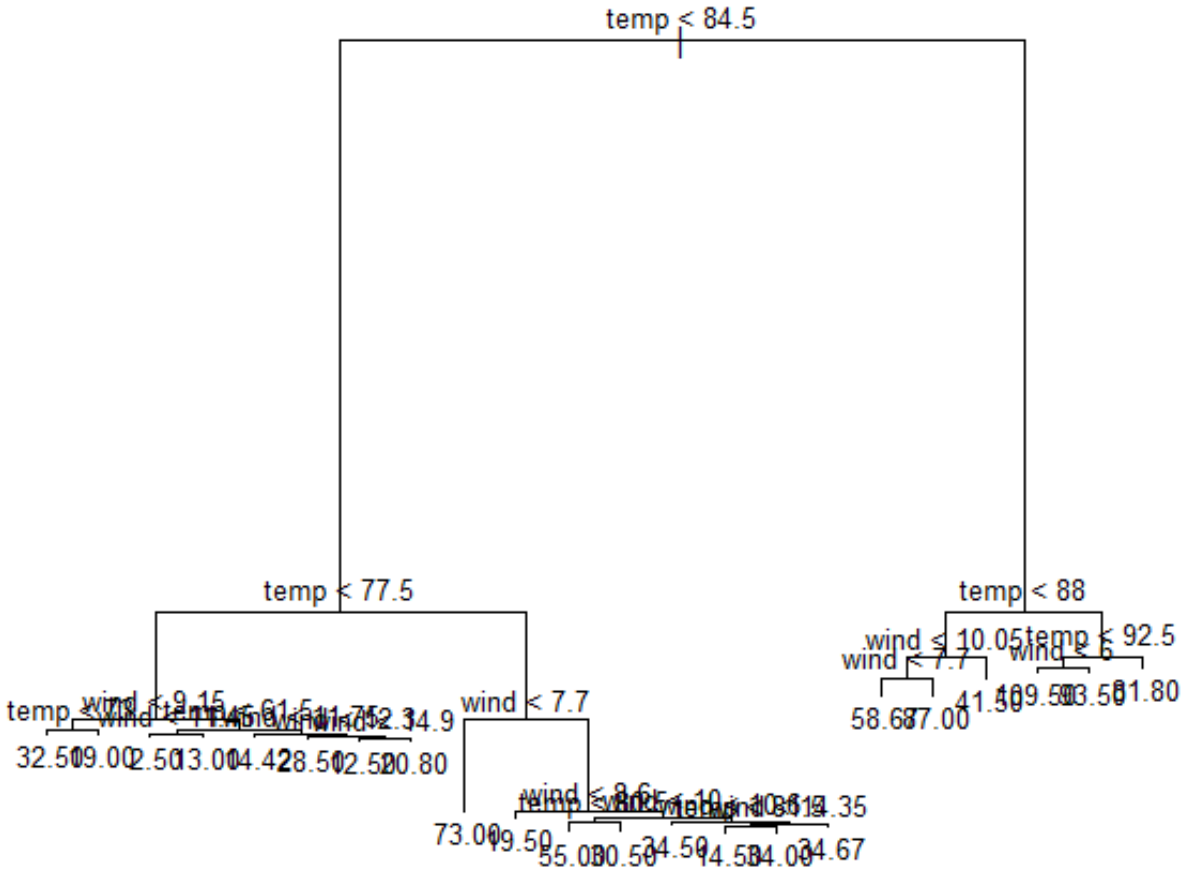




- We see that `temperature` is the "most important" predictor for predicting the ozone concentration. Observe that we can split on the same variable several times.

1. https://web.stanford.edu/~hastie/ElemStatLearn/datasets/ozone.info.txt

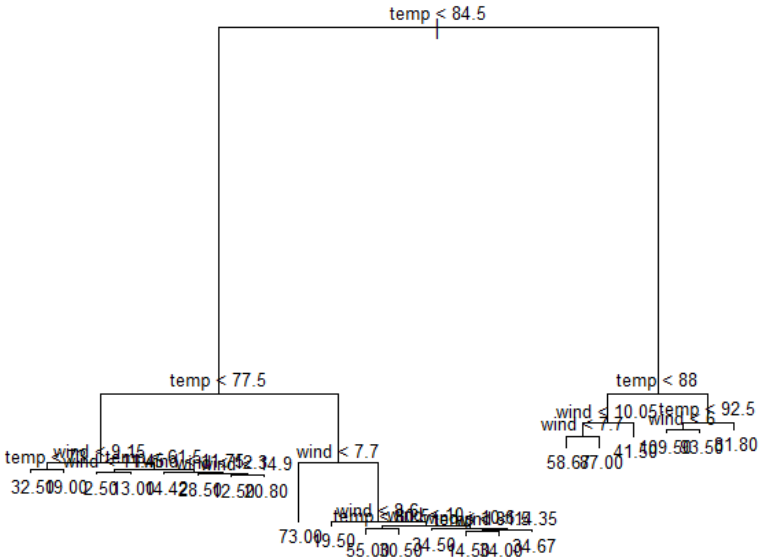
Deeper Tree: overfitting?

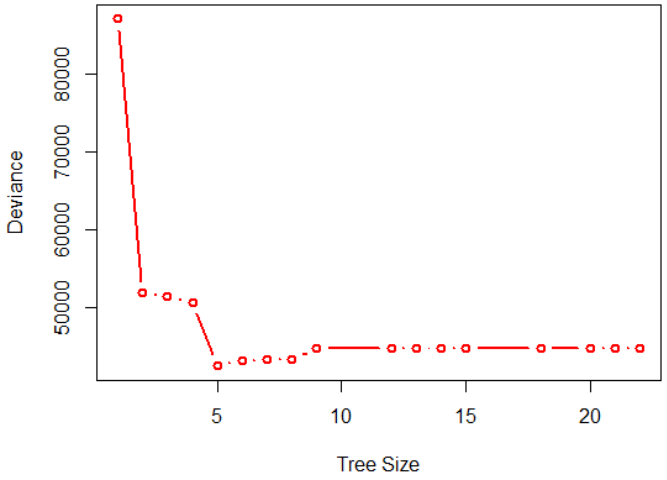


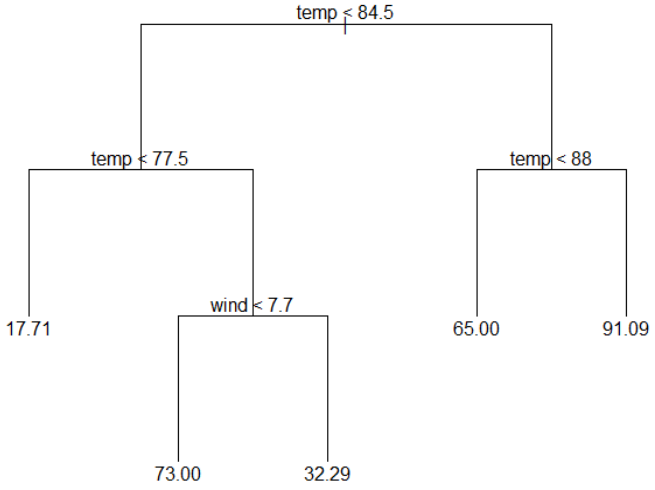
Building a regression tree: Algorithm

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. That is, divide the training observations into k folds. For each $k = 1, \dots, K$:
   - Repeat Steps 1 and 2 on all but the kth fold of the training data.
   - Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.
   - Average the results for each value of α, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Pruning the ozone tree







Decision Trees for Classification

Decision Trees for **Classification**

- Now allow for $K \geq 2$ number of classes for the response.
- Building a decision tree in this setting is similar to building a regression tree for a quantitative response, but there are two main differences: *the prediction* and the *splitting criterion*.

1) The prediction:

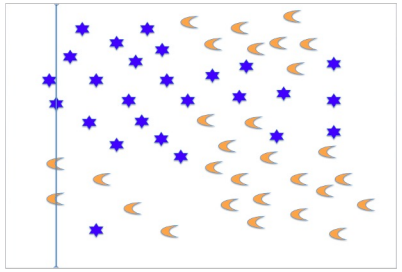
- In the *regression case* we use the mean value of the responses in R_j as a prediction for an observation that falls into region R_j.
- For the *classification case*, however, we have two possibilities:
  - **Majority vote**: Predict that the observation belongs to the most commonly occurring class of the training observations in R_j
  - Estimate the **probability** that an observation x_i belongs to a class k, $\hat{p}_{jk}(x_i)$, given as the proportion of class k training observations in region R_j.

    Region j has N_j observations and with n_{jk} observations lying in class k:

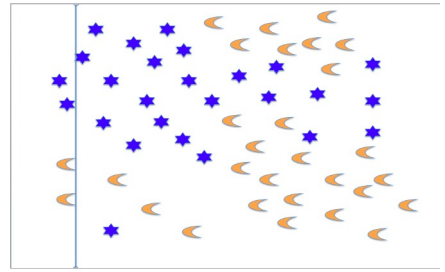
$$\hat{p}_{jk} = \frac{1}{N_j} \sum_{i:x_i \in R_j} I\left(y_i = k\right) = \frac{n_{jk}}{N_j}.$$

2) The splitting criterion: We can use some *measure of impurity* of the node. For leaf node j and class $k = 1, \dots, K$:

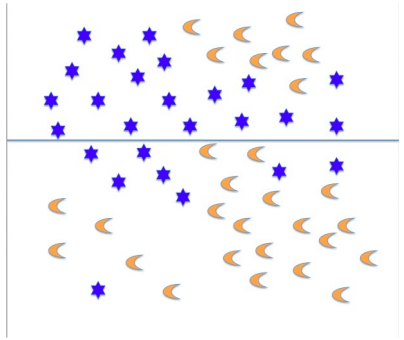
- **Gini index**:
  - $G = \sum_{k=1}^{K} \hat{p}_{jk} \, (1 - \hat{p}_{jk})$,
  - which is small if all of the $\hat{p}_{jk}$'s are close to 0 or 1.



Gini = 0,94



Gini : 4/55 × 2/26 × 2/29 + 51/55 × 24/26 × 27/29 = 0,79



Gini = 0,24

- **Cross entropy**:
  - $D = -\sum_{k=1}^{K} \hat{p}_{jk} \log \hat{p}_{jk}$.
  - Since $0 \leq \hat{p}_{jk} \leq 1$, it follows that $0 \leq -\hat{p}_{jk} \log \hat{p}_{jk}$, with values near zero of $\hat{p}_{jk}$ is close to 0 or 1.

When making a split in our classification tree, we want to minimize the Gini index or the cross-entropy.

From trees to forests

Advantages (+)

- Trees automatically select variables.
- Tree-growing algorithms scale well to large n, growing a tree greedily.
- Trees can handle mixed features (continuous, categorical) seamlessly, and can deal with missing data.
- Small trees are easy to interpret and explain to people.
- Some believe that decision trees mirror human decision making.
- Trees can be displayed graphically.

Disadvantages (-)

- Large trees are not easy to interpret.
- Trees do not generally have good prediction performance (high variance).
- Trees are not very robust, a small change in the data may cause a large change in the final estimated tree.

What is next?

Several of the listed disadvantages can be addressed by

- **Bagging**: grow many trees (from bootstrapped data) and average - to get rid of the non-robustness and high variance by averaging.
- **Random forests**: inject more randomness (and even less variance) by just allowing a random selection of predictors to be used for the splits at each node.
- **Boosting**: make one tree, then another based on the residuals from the previous, repeat. The final predictor is a weighted sum of these trees.

In addition:

- **Variable importance** - to see which variables make a difference (now that we have many trees).

Bagging

(bootstrap aggregation)

Bagging

- Decision trees often suffer from high variance. By this we mean that the trees are sensitive to small changes in the predictors: If we change the observation set, we may get a very different tree.
- Another way to understand "high variance" is that, if we split our training data into two parts and fit a tree on each, we might get rather different decision trees.
- To reduce the variance of decision trees we can apply *bootstrap aggregating (bagging),* invented by Leo Breiman in 1996.

  - Assume we have B *i.i.d.* observations of a random variable X each with the same mean and with variance σ^2. We calculate the mean $\bar{X} = \frac{1}{B}\sum_{b=1}^{B} X_b$. The variance of the mean is

$$\text{Var}(\bar{X}) = \text{Var}(\frac{1}{B}\sum_{b=1}^{B} X_b) = \frac{1}{B^2}\sum_{b=1}^{B} \text{Var}\,(X_b) = \frac{\sigma^2}{B}.$$

  - By averaging we get reduced variance. This is the basic idea!
  - For decision trees, if we have B training sets, we could estimate $\hat{f}_1(\mathbf{x}), \hat{f}_2(\mathbf{x}), \ldots, \hat{f}_B(\mathbf{x})$ and average them as

$$\hat{f}_{avg}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b\,(\mathbf{x})\,.$$

However, we do not have many independent data set - so we use *bootstrapping* to construct B data sets.

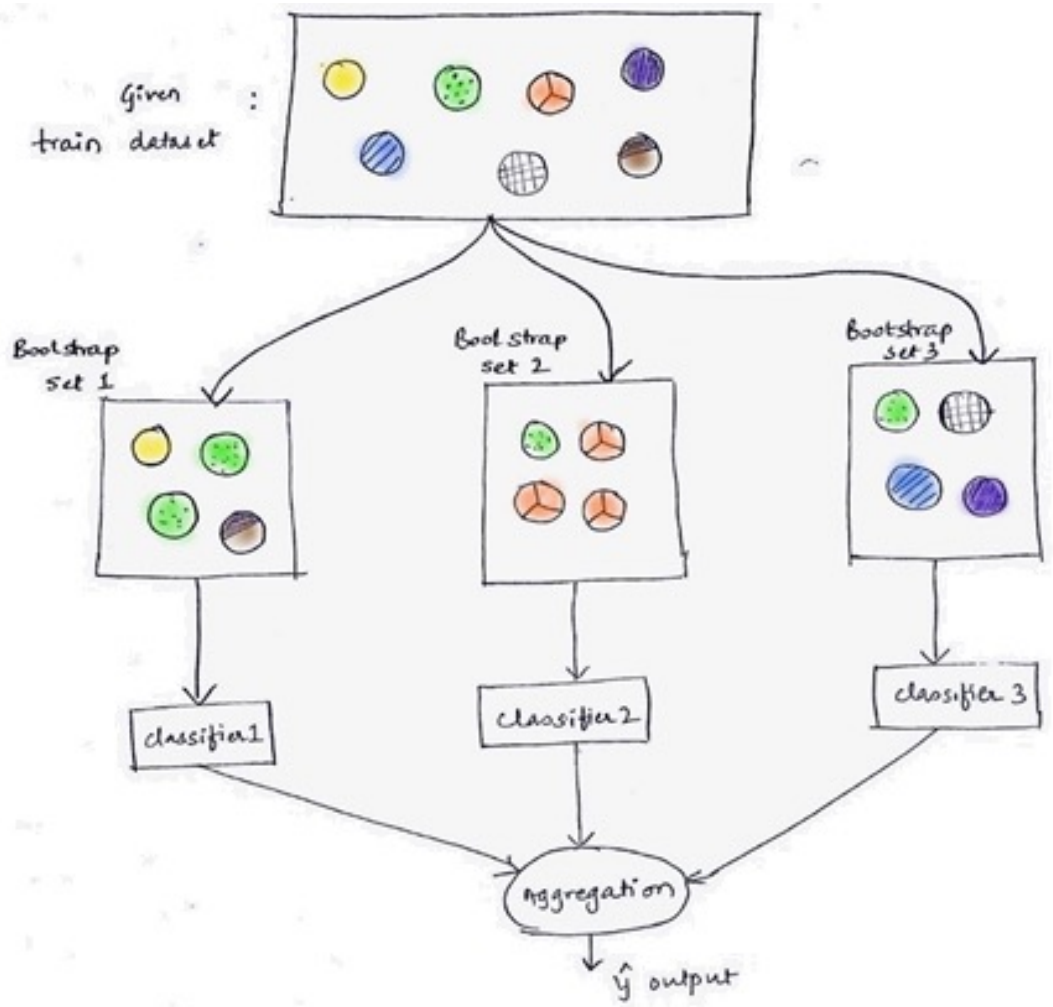
Bagging

- Bootsrapping: Draw *with replacement* n observations from our sample - and that is our first *bootstrap sample*.
- We repeat this B times and get B bootstrap samples - that we use as our B data sets.
- For each bootstrap sample $b = 1, \dots, B$ we construct a decision tree, $\hat{f}^{*b}(x)$.

- For a *regression tree*, we take the average of all of the predictions and use this as the final result:

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

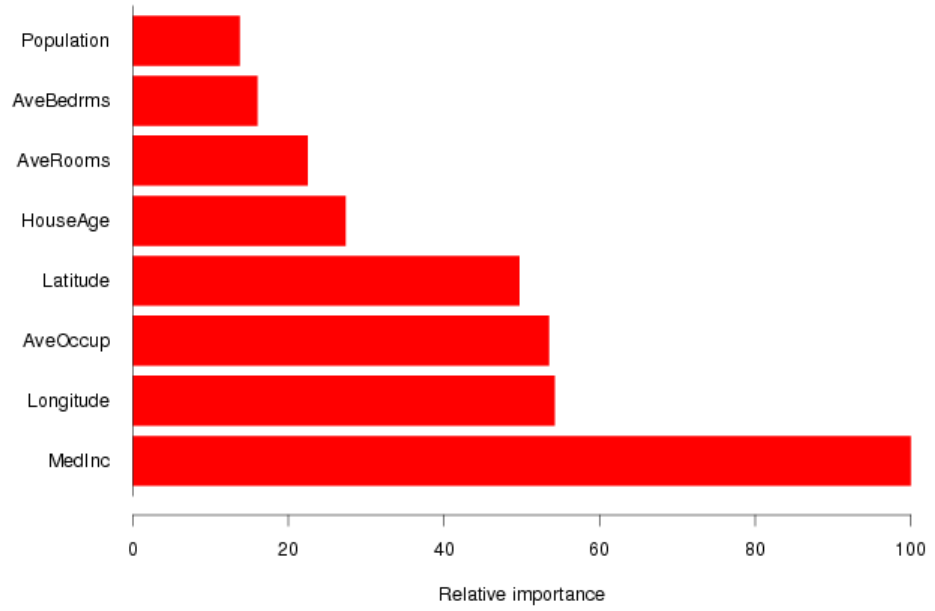
- For a *classification tree* we record the predicted class (for a given observation x) for each of the B trees and use the most occurring classification (*majority vote*) as the final prediction.



Source: link

Variable importance plots

- Drawback of bagging: It becomes difficult to interpret the results. Instead of having one tree, the resulting model consists of many trees (it is an ensemble method).
- *Variable importance plots* show the relative importance of the predictors: the predictors are sorted according to their importance, such that the top variables have a higher importance than the bottom variables.
- There are in general two types of variable importance plots:
  - variable importance based on decrease in node impurity.
  - variable importance based on randomization.



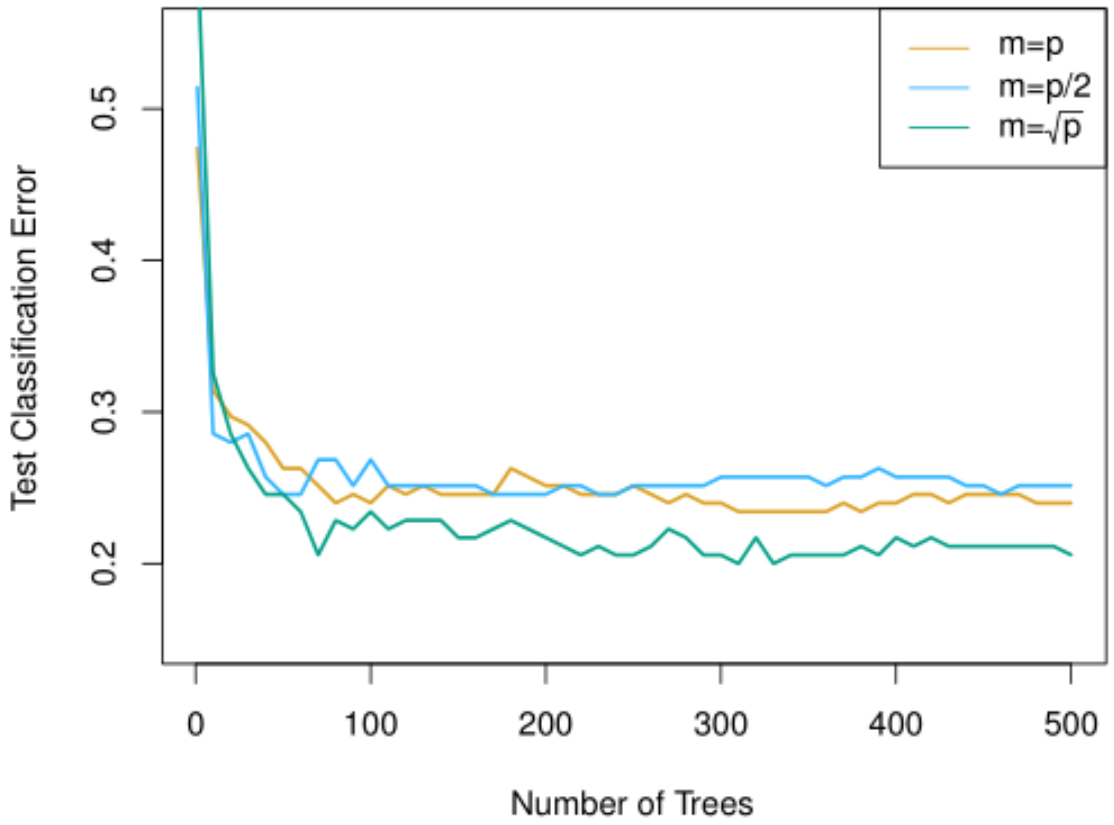
Random Forests

Random Forests motivation

- If there is a strong predictor in the dataset, the decision trees produced by each of the bootstrap samples in the bagging algorithm becomes very similar: Most of the trees will use the same strong predictor in the top split.
- Not optimal, as we get B trees that are highly correlated. → No large reduction in variance by averaging $\hat{f}^{*b}(x)$ when the correlation between the trees is high.

- Random forests provide an improvement over bagged trees by a small tweak that *decorrelates the trees*. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But each time a split in a tree is considered, a random selection of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.
- A fresh selection of m predictors is taken at each split, typically
  - $m \approx \sqrt{p}$ (classification),
  - $m = p/3$ (regression).
- The general idea is that for very correlated predictors m is chosen to be small.

Comparison


m=p
m=p/2
m=√p
Test Classification Error
0.2
0.3
0.4
0.5
0
100
200
300
400
500
Number of Trees

Boosting

Boosting

Boosting is an alternative approach for improving the predictions resulting from a decision tree. We will only consider the description of boosting regression trees (and not classification trees) in this course.

In boosting the trees are grown *sequentially* so that each tree is grown using information from the previous tree.

- First build a decision tree with d splits (and $d+1$ terminal notes).
- Next, improve the model in areas where the model didn't perform well. This is done by fitting a decision tree to the *residuals of the model*. This procedure is called *learning slowly*.
- The first decision tree is then updated based on the residual tree, but with a weight.
- The procedure is repeated until some stopping criterion is reached. Each of the trees can be very small, with just a few terminal nodes (or just one split).

Algorithm 8.2: Boosting for regression trees

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, ..., B$, repeat:
   a) Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data.
   b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$

   c) Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$

3. The boosted model is $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$.

Boosting has three tuning parameters which need to be set (B, λ, d), and can be found using cross-validation.

Boosting

Tuning parameters

- **Number of trees** B. Could be chosen using cross-validation. A too small value of B would imply that much information is unused (remember that boosting is a slow learner), whereas a too large value of B may lead to overfitting.
- **Shrinkage parameter** λ. Controls the rate at which boosting learns. λ scales the new information from the b-th tree, when added to the existing tree $\hat{f}$. Choosing a small value for λ ensures that the algorithm learns slowly, but will require a larger B. Typical values of λ is 0.1 or 0.01.
- **Interaction depth** d: The number of splits in each tree. This parameter controls the complexity of the boosted tree ensemble (the level of interaction between variables that we may estimate). By choosing $d = 1$ a tree stump will be fitted at each step and this gives an additive model.