Machine Learning

Marica Valente[1]

June 25, 2024

[1]Assistant Professor, University of Innsbruck (marica.valente@uibk.ac.at)

Outline

Chapter 1: Tree-based ML (Non-parametric ML)

CART

Classification and Regression Trees fit a model into each leaf after pushing data down their branches (Breiman et al. 1984)

Nonparametric methods for HD data

Tree-based methods: Supervised learning approaches

- Algorithm predicts y from x using decision trees that is classifier s mapping $s : R^p \to y$
- Non-parametric, flexible - non-linearities without explicit modeling
- Technically easy to implement, relatively easy to interpret
- Useful for description, exploration, and prediction of data

Regression trees (RT)

- Continuous dependent variables
- $Y = \mu(X) + \varepsilon$ with $E[\varepsilon|X] = 0$
- $\hat{Y} = \hat{\mu}(X) = \hat{E}[Y|X = x]$

Classification trees (CT)

- Discrete dependent variables
- μ is Bayes (probabilistic) classifier
- $\hat{\mu}(X) = \hat{Pr}(Y = k|X = x)$

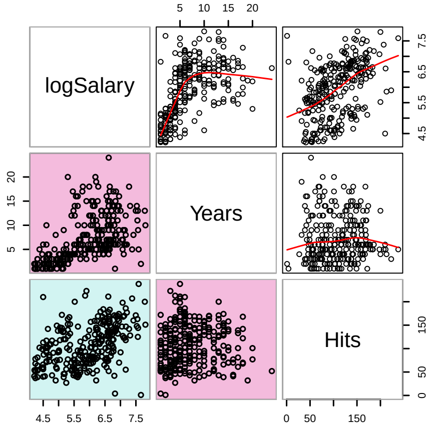
Deriving prediction with trees

$\hat{\mu}$ derived with training data - prediction accuracy with test data

1. Use training set to divide predictor space, i.e., the set of possible values for $X_1, ..., X_p$ into J distinct regions $R_1, ..., R_J$
2. For every observation of the test set that falls into R_j, we make the same prediction $\hat{y}_{R_j}$
    1. Predictor ($\hat{\mu}$) in regression tree: **Average outcome ($\bar{y}$) in** R_j
    2. Quality of prediction: $MSE = \frac{1}{n}\sum_i^n (y_i - \hat{y}_i)^2$
    3. Comparison of prediction accuracy of different trees: MSE for test set

Example: Predicting baseball player's salary

- Major League Baseball Data, seasons 1986-87, 322 players
- Response (dependent variable): log of salary in thousand $
- Predictors: experience (years in major leagues) and hits (hits last year)

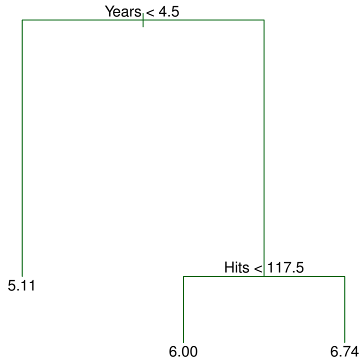


Standard linear regression results

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	4.2751	0.1184	36.11	0.0000
Hits	0.0087	0.0009	9.85	0.0000
Years	0.0982	0.0083	11.85	0.0000

- Salary increases with years of experience and hitting success, both highly significant

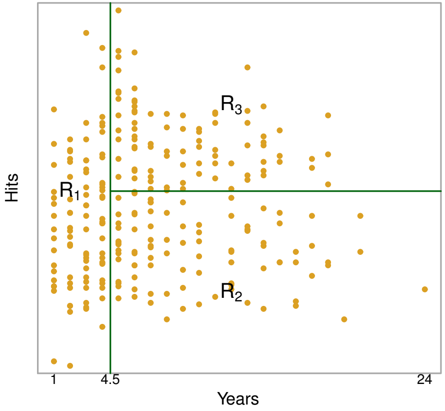
Regression tree



Interpretation

- If less than 4.5yrs experience, mean log income is 5.11$
- If more than 4.5yrs experience, matters! Mean log income is:
  - 6.0 if $<$ 117.5 hits last year
  - 6.74 if $>$ 117.5 hits last year

- Tree partitions players into three regions
  $R_1 = \{Y|yrs < 4.5\}$
  $R_2 = \{Y|yrs > 4.5, h < 117.5\}$
  $R_3 = \{Y|yrs > 4.5, h > 117.5\}$
- Each region has obs. with similar income $\hat{y}_{R_1} = 5.11$ $\hat{y}_{R_2} = 6.00$ $\hat{y}_{R_3} = 6.74$



Interpretation of results

Standard regression model

- Salary increases with years of experience and hitting success, both highly significant

Tree-based model

- Year: most important factor determining salary, salary increases with experience
- For less experienced players, "hits" are of no relevance for salary
- For players with more than 5yrs experience, the hits in the last period positively influence the salary

How to derive a regression tree

How to derive $R_1, ..., R_p$?

- Intuition: Consider R_j as rectangles that partition the X space depending on values of X_j
- Find $R_1, ..., R_J$ that minimize $RSS = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y_{R_j}})^2$

  $\rightarrow$ Recursive partitioning

Recursive partitioning - Top-down splitting

1) Start at top of tree $\rightarrow$ single space R

2) Partition R: $R_1(j,s) = \{X | X_j \leq s\}$ and $R_2(j,s) = \{X | X_j > s\}$

- Identify predictor X_j and split point s that minimizes RSS

$$\min_{j,s} \sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

3) Identify RSS-minimizing (j, s) to partition R_1 (or R_2) into R_3 and R_4

. . .

J) Identify RSS-minimizing (j, s) to partition R_j into R_J and R_{J+1}

Recursive partitioning

Overfitting: if J(nr. leaves)$= n$(nr. obs) $\rightarrow y_i = \hat{y}_{R_i}$

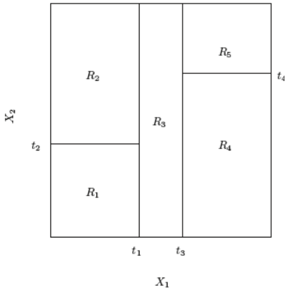
- Complex trees: bad predictions in test set
- Smaller trees: smaller variance, better interpretation, but bias

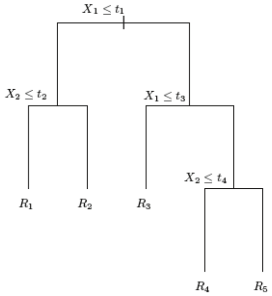
Avoiding too complex trees - stopping criteria

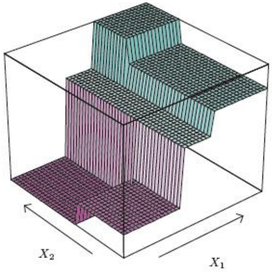
- Minimum number of observations in R_j
- Maximum number of final regions R_j (max. tree depth)
- RSS reduction greater than threshold

Example for recursive partitioning

Binary splitting - Corresponding tree - Contour of prediction surface







Pruning

Shorten a complex tree T_0 to obtain a subtree

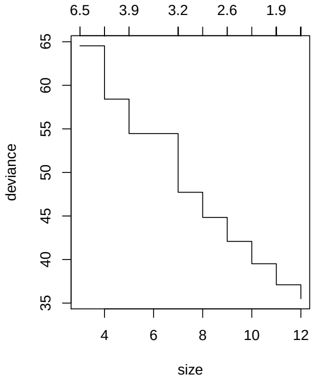
- Penalize tree depth: define number of terminal nodes T ($< T_0$)
- For a given α, identify X_j and split point s that minimizes RSS:

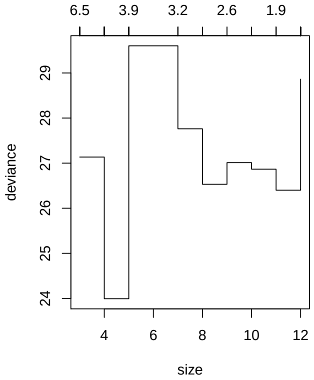
$$\min_{j,s} \sum_{i: x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2 + \alpha T$$

- α controls trade-off: less complexity (more bias in training set, less variance) to better fit test data
- $\alpha = 0$ delivers T_0; $\alpha > 0$ increases the RSS $\rightarrow$ penalizes complexity IS
- Breiman (1984): for each α there is a unique T_α minimizing the above

Example: Salaries in baseball - Pruning

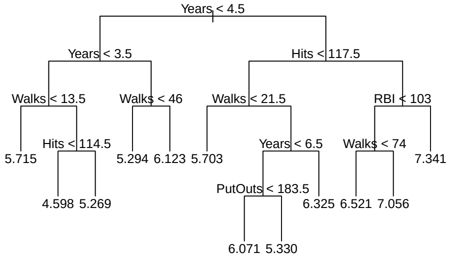
- x-axis: resulting tree size for $\alpha = 0.1, 0.2, ..., 7$
- y-axis: RSS (deviance) $\sum_i (y_i - \hat{y}_i)^2$
- Left: Training set, Right: Test set



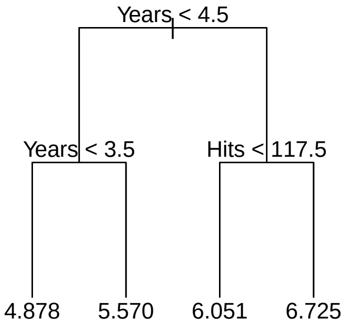


Example: Salaries in Baseball - Pruning

Unpruned tree (MSE: 0.370)



Pruned tree (MSE: 0.300)



Implementation in R

R Packages

- rpart + rpart.plot
- party
- caret
- tree

Regression tree example with rpart

Build a tree

```
tree <- rpart(logSalary ~ . ,      Estimation
              data=train,                         Dataset to use
              control=rpart.control(cp=0))  Complexity Parameter
```

In R, tuning parameter (stopping criterion) is called cp. Higher cp, shorter tree. Cp=0 means full tree (of maxdepth=30 by default in R). Cp is minimum explained variance by a split to occur

Prune the tree

```
pruned_tree <- prune(tree, cp=0.01)     Pruning with cp=0.01
```

Cp=0.01 means perform the split if explained variance is at least 1%

Plot the tree: Compare full tree with pruned tree

```
rpart.plot(tree)
rpart.plot(pruned_tree)
```

Cross validation (CV)

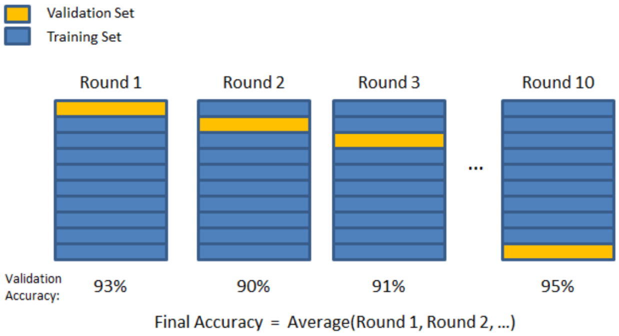
We can tune α to produce the best OOS prediction
k-fold cross validation:

1. Split data into k subsets (folds) of similar size
2. For each k
    1. Use fold k as test set - remaining $k-1$ folds are one training set
    2. Pick a value for the tuning parameter (α)
    3. Fit model using the $k-1$ folds other than fold k
    4. Calculate MSE_k for observations in k (test set)
3. Among the values of α pick the one associated to the lowest MSE_k

$\rightarrow$ Usually $k = 10$ or $k = 5$ is used (R automatically sets $k = 10$)

k-fold CV in detail

- Split data into k *mutually exclusive* subsets of *equal size*
- E.g. 10-fold CV, N=101: 9 folds of 10 items and 1 fold of 11 items
- k=10 for small dataset, 5 otherwise (best bias-variance trade-off empirically:
- k too small high variance (full data and folds too similar) vs. k too big high bias (small folds more likely to be noisy)
- k train/validation splits for k evaluation rounds:



Cross validation for model tuning

- Why? How we train/test split datasets (if small or medium sized) can make a significant difference: the results can vary tremendously
- With CV we technically don't even need a test set
- Still holding out a test set allows to **evaluate different algorithms** on the **same held-out data**
- **Tree:** Prune tree using CV to reduce tree size optimally (pick optimal complexity parameter α)
- Importantly, even without CV, trees are often pruned in R: maxdepth=30 by default, and min. nodesize=5 is common in RT
- **Random forest (coming soon):** No need of CV because each tree is unpruned

Example for classification tree (Titanic data)

- In the night of 14 April through to the morning of 15 April 1912, the RMS Titanic sank on its maiden voyage from Southampton to New York. Of the estimated 2,224 people on board, more than 1,500 found death in the North Atlantic Ocean.
- Our R session aims to identify the variables that best predict survival and death of the passengers using tree-based methods.
- Our task is to train algorithms to predict survival/death of passengers accurately.
- We use a dataset of 1,000 passengers of the Titanic that contains passenger's information, e.g., age, name, and family status, the ticket price that was paid, and a binary variable that equals 1 if the passenger survived.
- Deriving further features/predictors from the given data is your task as it may increase predictive power of the algorithm considerably.

The Titanic contest

- Derive further features/predictors from the given data - this may increase predictive power of the algorithm considerably!
- To check the accuracy of the trained algorithm, another dataset of around 190 passengers is used as ultimate test data. This dataset contains the same information as the main dataset - except for the survival dummy variable.
- We will run some algorithms together, then at home please add/create new variables and try to improve the OOS fit.
- Then I will make available the true survivors data and you will see how your algorithm performed.
- Voluntary

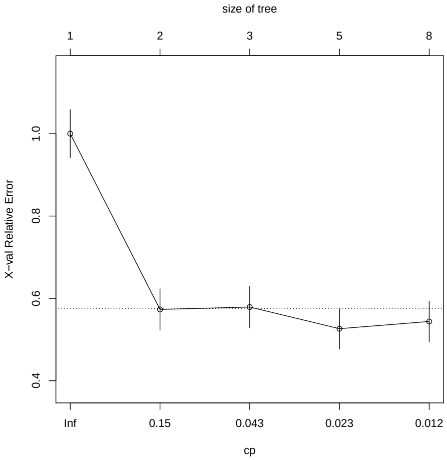
Example for classification tree (Titanic data)

Pruned tree

- Misclassification error stable (y-axis, in R called xerror)
- Optimal size already with 6 leafs
- This corresponds to $\alpha = 0.023$

Optimal pruned tree (opt. α)

- Training set misclassification = 43% (unpruned: 32%)
- Test set misclassification = 17% (unpruned: 27%)



Classification tree example with rpart

Build a tree

```
model.rpart <- rpart(survived ~ .,   Estimation
                  data = titanic.train,        Dataset to use
                  method = "class",               Classification
                  control = rpart.control(cp=0))  Complexity
```

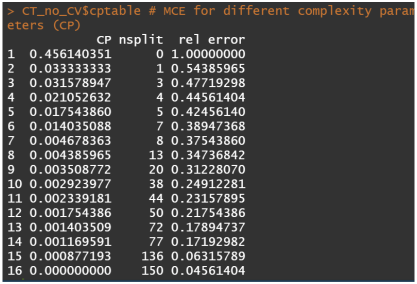
Prune the tree

```
model.pruned <- prune(model.rpart,cp=0.023) Pruning with cp=0.023
```

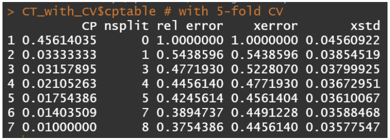
Plot the tree

```
rpart.plot(model.pruned)
```

- “CP” (“complexity”): % reduction in classification error with that split
- 1% is the default limit for deciding when to stop splitting (CP=0.01)
- At CP=0.01 we prune off splits which do not improve the fit by at least 1%
- “rel error”: standardized classification error in % (divided by error with no splits)
- With CV (e.g. xval=5) “xerror” is the CV error, “xstd” its variability across folds
- CP* at min(xerror)



```
> CT_no_CV$cptable # MCE for different complexity param
eters (CP)
             CP nsplit  rel error
1  0.456140351      0 1.00000000
2  0.033333333      1 0.54385965
3  0.031578947      3 0.47719298
4  0.021052632      4 0.44561404
5  0.017543860      5 0.42456140
6  0.014035088      7 0.38947368
7  0.004678363      8 0.37543860
8  0.004385965     13 0.34736842
9  0.003508772     20 0.31228070
10 0.002923977     38 0.24912281
11 0.002339181     44 0.23157895
12 0.001754386     50 0.21754386
13 0.001403509     72 0.17894737
14 0.001169591     77 0.17192982
15 0.000877193    136 0.06315789
16 0.000000000    150 0.04561404
```



```
> CT_with_CV$cptable # with 5-fold CV
          CP nsplit rel error    xerror       xstd
1 0.45614035      0 1.0000000 1.0000000 0.04560922
2 0.03333333      1 0.5438596 0.5438596 0.03854519
3 0.03157895      3 0.4771930 0.5228070 0.03799925
4 0.02105263      4 0.4456140 0.4771930 0.03672951
5 0.01754386      5 0.4245614 0.4561404 0.03610067
6 0.01403509      7 0.3894737 0.4491228 0.03588468
7 0.01000000      8 0.3754386 0.4456140 0.03577547
```

- nsplit=0 means 100% error (CP=1), 1 split means 45% improved fit (CP=0.45)
- nsplit=150 leads to 4.5% error: in R splits are blocked when maxdepth=30

CV in R - example for Titanic data

Build full tree

```
CT_CV5 <- rpart(survived ~ .,                              Estimation
                data=titanic.train,                       Data
                method="class",                       Classification
                control=rpart.control(xval=5))
                 Starting from full complexity (deep tree), 5-fold
                 Default is xval=10, 10-fold CV
```

Identify optimal pruning parameter

```
cp_opt <- CT_CV5$cptable[which.min(CT_CV5$cptable[,4]),1]
```

Prune tree with optimal parameter

```
CT_CV5 <- prune(CT_CV5,cp=cp_opt)
```

Summary on tree-based methods

Advantages

- Easy to interpret, easy to explain - useful graphical representation
- Useful for data description, exploration, and prediction
- Outperforms LM if relationships are non-linear - without modeling
- Endogenous variable selection
- Outlier robustness - splits are not at the tails of the data: Since, extreme values or outliers, never cause much reduction in RSS, they are never involved in split.
- No dimensionality problems (handles very large X)
- Categorical predictors can be included without additional dummies
- All variables are allowed to interact

Disadvantages

- Variable selection is data-driven, not theory-driven - Interpretation can be difficult
- Too much information: "Paralysis of the analysis"

Bagging - Bootstrap aggregation

Motivation

- Deeper trees: Variance increases, bias decreases
- But: repeating procedure and averaging results reduces variance

Bagging procedure

1. Split data set B times in training sets (bootstrap random samples)
2. Build **deep** tree on b^{th} data set to obtain $\hat{f}^{*b}(x)$
3. Average predictions across B trees (in-sample):

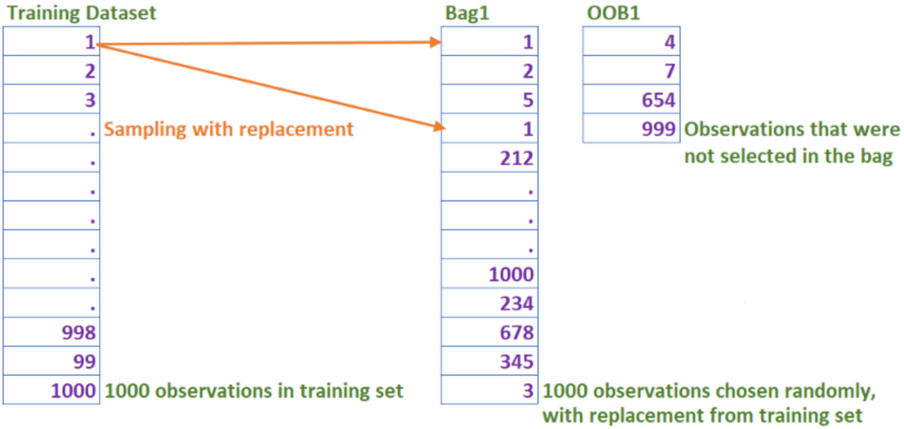
$$\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(x)$$

How to assess performance? $\rightarrow$ Out-of-bag (OOB) error

- Each b contains sample not used to train $\hat{f}_b$ - the Out-of-bag sample $\rightarrow$ Each obs. will be predicted multiple times (in different trees)
- Use trees that do not contain i
- For each i OOB, average predictions across trees, then estimate MSE
- Prediction are obtained as $\hat{f}_i = \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b^{-i}(x)$
- Choice of B: "high enough" - no risk of overfitting (trees are deep!)

Creation of Bag and Out-Of-Bag (OOB)

Bootstrap: for every bag, draw at random 1000 obs. (1 obs. 1000 times). Each draw is made with replacement. Meaning, when the 1st sample is chosen, there are 1000 options to chose from.

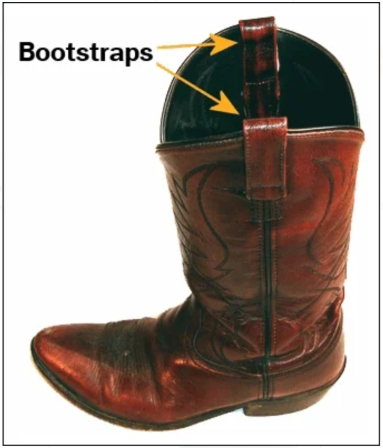

Training Dataset
1
2
3
.
.
.
.
.
.
.
998
99
1000
Sampling with replacement
1000 observations in training set
Bag1
1
2
5
1
212
.
.
.
1000
234
678
345
3
1000 observations chosen randomly, with replacement from training set
OOB1
4
7
654
999
Observations that were not selected in the bag

How Random can the Forest be?

Fraction of obs. in the bag is around 63.2% (with bootstrap).
Proof:

- n. of obs. in training set = n
- Size of each bag = n
- Probability of an obs. being selected in a draw = 1/n
- Probability of an obs. not being selected in a draw = 1-(1/n)
- Number of draws = Size of Bag = n
- For each bag, the probability of an obs. not being selected at all = $[1-(1/n)]^n$
- As the value of n increases, this value tends to 36.8%
- Probability of an obs. making it into the bag after n draws
- = 1 – 36.8%
- = 63.2%

The history of bootstrap



The history of bootstrap

- *History.* The term is often attributed to Rudolf Erich Raspe's story The Surprising Adventures of Baron Munchausen
- He is riding around on his horse in a forest and suddenly gets stuck in mud. He screams for help but there is no one around who hears his voice! Luckily our hero does not give up and gets a great idea: "what if I just pull myself out of this mud?". He grabs the straps of his boots and pulls himself loose. Fantastic - he just invented bootstrapping.
- *Nowadays.* Bootstrapping means "to pull yourself up by your bootstraps": to help yourself without the aid of others; use your own resources (e.g. shall I bootstrap my startup? = build my business with my own resources without external investment or with minimal external capital)

Combining random trees into a forest

Inject randomness into trees, and build a forest to reduce variance and improve OOB predictions (Breiman, 2001)

Random Forests (RF)

Motivation

- If there is a strong predictor, all bagged trees will look similar - using this strong predictor for the first split
  $\rightarrow$ Predictions from the bagged trees are highly correlated

RF decorrelates the B trees

- ´**At each split**, tree b uses only a **random subset** m of the p predictors $\rightarrow$ Trees are more different from each other

- Default in R: $m = \sqrt{p}$ (or smaller if predictors highly correlated)

Random Forest Procedure

B trees build a forest. For b in $1, ..., B$

1. Draw bootstrap sample of the training data
2. Grow tree with recursive partitioning until minimum node size is reached. Thereby
   1. At each split select a random subset of predictors m
   2. Use the optimal split using only the m predictors
3. The ensemble of trees is a sequence of trees
4. Each observation i is IS for some trees, and OOS in other trees
5. To make predictions for each observation i, take the average outcome (or majority rule) using only trees where i is OOS
6. Prediction are obtained as $\hat{f}_i = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b^{-i}(x)$

RF Bias?

- Each tree b is a weak learner - uses less information than a standard tree
- To achieve low bias each tree is grown to maximum depth (considering that in R maxdepth=30 splits)
- Choice of B: "high enough" - test set error converges, no risk of overfitting

Bagging in R: bagging()

Bagging parameters

- Choice of B (*nbagg*)
- Use OOB sample to estimate the missclass. error or RSS (*coob*=TRUE)

And as in standard CT / RT

- Min. number of obs. to split (rpart: *minsplit*)
- Min. number of obs. in leaves (rpart: *minbucket*)
- Tree complexity (rpart: *cp*)
- Maximum tree depth (rpart: *maxdepth*)
- ...

Leo Breiman (1996), Bagging Predictors. Machine Learning 24(2), 123-140.

Random Forest parameters

- Number of trees B (randomForest: *ntree*)
- Subsample size in b (randomForest: *sampsize*)
- Number of variables considered for splitting m (randomForest: *mtry*)

And as in standard CT / RT

- Min. number of obs. in leaves (randomForest: *nodesize*)
- Maximum tree depth (randomForest: *maxnodes*)
- ...

Random Forest example with randomForest

Build a Random Forest

```
RF_tree <- randomForest(AHD ~ .,          Estimation
                 data=train,              Dataset to use
                 mtry=2,                  m
                 ntree=500,               Number of trees
                 maxnodes=4)              Maximum nr. of nodes
```

Prediction

```
pruned_tree <- predict(RF_tree, newdata=test)
```

Plot RF OOB training error

```
plot(RF$err.rate[,1], type="l")
```

RF algorithm: Summary

1. **Random Item Selection**: Each tree is trained on about 2/3 of full data (exactly 63.2%). Obs. are drawn at random with replacement from the original data (=bootstrap). The bootstrap sample is the training set for growing the tree.

2. **Random Variable Selection**: Some predictor variables (mtry, typically 1/3) are selected at random out of all the predictor variables and the best split on these mtry is used to split the node

   Note: The value of mtry is held constant during the forest growing.

   Recall: In a standard tree, each split is created after examining every variable and picking the best split from all the variables (not just mtry of them)

3. For each tree, using the leftover (36.8%) data, calculate the misclassification or MSE rate - **out of bag (OOB) error rate**. Aggregate error from all trees to determine overall OOB error rate for the classification. in R:

```r
RF # random forest from running randomForest()
print(RF) # gives overall OOB error rate
```

   If we grow 500 trees, on average an obs. will be OOB for about .37*500=185 trees

Frequently Asked Questions

- How “deep” is actually each tree?

$\rightarrow$ **randomForest(... nodesize=)** Min size of terminal nodes. Setting this number larger causes smaller trees and thus takes less time. Default values differ for classification (1) and regression (5)

- Can we reduce variance by increasing **nodesize**?

$\rightarrow$ The whole idea of RF is to build deep trees and reduce variance by *averaging predictions across trees*: if OOB too high (because RF variance is high, i.e. you are overfitting) then increase **ntree** to reduce variance and/or reduce **mtry** to reduce noise due to correlated trees

- What is the role of **mtry** in growing a tree?

$\rightarrow$ **mtry**: n. of variables randomly sampled as candidates **at each split.** The algorithm searches the optimal split using mtry variables only. Default values differ for classification: sqrt(p) where p is number of predictors vs. regression p/3

- How likely it is that we do not have an OOB sample?

$\rightarrow$ For **each** bag of size n, the probability of an obs. not being selected at all = $[1\text{-}(1/\text{n})]^n >>> 0$ (already >36% for each tree with n=100)

$\rightarrow$ If n small and ntree small, bags and OOB may be too correlated/noisy: Increase the number of bags (trees)!

- In practice, if the OOB error is high, which are the most important parameters to change?

→ Increase **ntree** / tune **mtry** using **tuneRF()**

```
tuneRF(x, y, mtryStart, ntreeTry=50, stepFactor=2,
improve=0.05, doBest=FALSE, ...)
```

- x,y = matrix of predictors, vector of outcomes
- mtryStart = starting value of mtry. Default is mtry in randomForest
- ntreeTry = number of trees used at the tuning step (smaller als ntree)
- stepFactor=at each split, mtry is inflated (or deflated) by this value
- improve = the (relative) improvement in OOB error must be by this much for the search to continue
- doBest= whether to run a forest using the optimal mtry found

Interpretation

Problem: How to interpret B trees?

$\rightarrow$ **Variable importance**

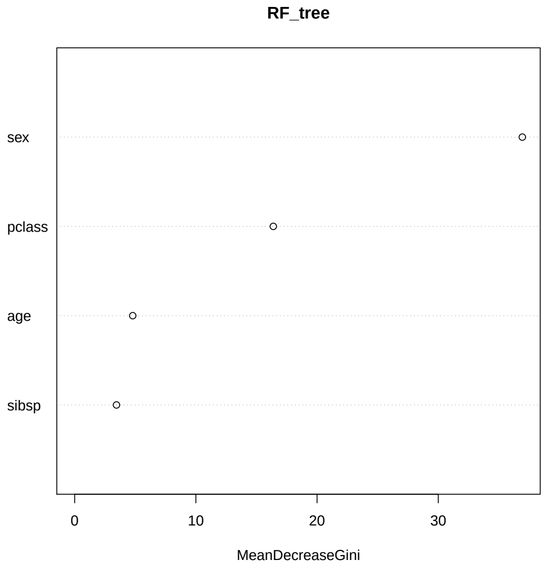
$\rightarrow$ **Variable marginal effects**

Problem: How to interpret B trees? $\rightarrow$ Variable importance

- Record for each tree how much RSS reduces due to a split for a predictor (or missclassification error for CT)
- RSS gain averaged over B is a measure for variable importance $\rightarrow$ Large RSS gain: important predictor
- For classification: **Gini Index** used instead of misclassification error
- If, after a split, each node becomes more pure = informative split!
- min. Gini Index is 0 = pure node = all elements in the node are of one unique class: this node will not be split again! Thus, the optimum split is chosen by the features with less Gini Index
- Decrease in Gini = lower variance WITHIN node = higher variance ACROSS nodes $\rightarrow$ Large Gini decrease: important predictor

```
varImpPlot(RF_tress) # computes variable importance in R
```

Random Forest: Variable Importance Plot



Problem: How to interpret B trees? $\rightarrow$ Variable marginal effects

- How to compute the marginal effect of x_1 on y?
- Is the relationship between y and x_1 linear, monotonic or more complex?
- Linear regression models: relationship is linear
- Compute marginal effect of x = Compute Partial Dependence Functions (and plot them!)
- Visualization is one of the most powerful interpretational tools
- Partial Dependence Plots (PDP) help interpret models produced by any “black box” prediction method

RF: Estimating the marginal effect of x_1

For **regression** problems (y continuous):

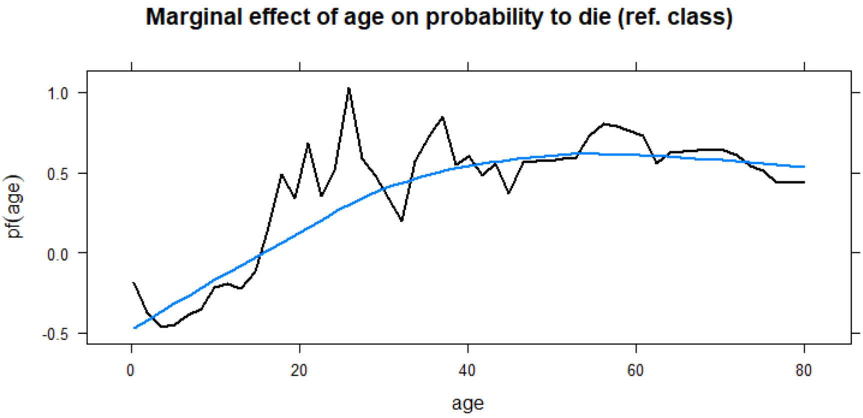
1. Estimate the forest, then define a specific set of values for x_1
2. For a given value of x_1, predict outcome for all individuals
3. **Average** n **predictions for that level of** x_1
4. Function mapping x_1 to average predictions is a **“partial function”**
5. Plot **avg prediction on y-axis**, corresponding value of x_1 **on x-axis**

RF: Estimating the marginal effect of x_1

For **classification** problems (y discrete):

1. Instead of averaging predictions for a given value of x_1, **get the share of "votes" for each class and compute partial function**
2. E.g. $age = 3$ and $n = 100$: Assume 20 obs. are predicted to die (0) and 80 are predicted to survive (1), so $p_0 = 0.2$ and $p_1 = 1 - p_0 = 0.8$
3. Thus, $age = 3$ is associated to low probability to die
4. Take "die" as reference class, plot partial function (pf) on y-axis: $pf(age = 3) = log(0.2) - \frac{1}{2}log(0.8) = -1.5$
5. This is the **relative logit contribution** of $age = 3$ on prob. to die
6. Negative values on y-axis = low probability to die **ceteris paribus**
7. Positive values on y-axis = high probability to die **ceteris paribus**
8. Zero value on y-axis **= zero marginal effect**

RF: Partial Dependence Plot



R Code for Partial Dependence Plot (PDP)

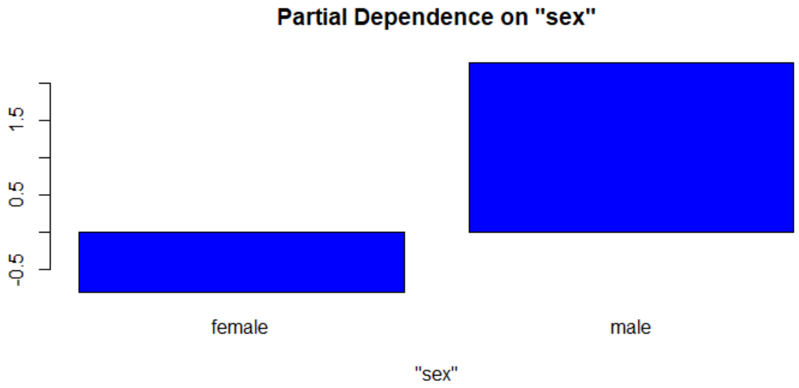
```r
library(pdp) # pdp is one the many packages for PDP
library(dplyr) # for the %>% operator

RF  # object (forest) estimated with randomForest package

RF %>%  # the %>% operator is read as "and then"
partial(pred.var = "age") %>% # estimate pf(age)
plotPartial(smooth = TRUE, # fit a smooth blue line
ylab = expression(pf(age)), # y label and title
main ="Marginal effect of age on probability to die")
```

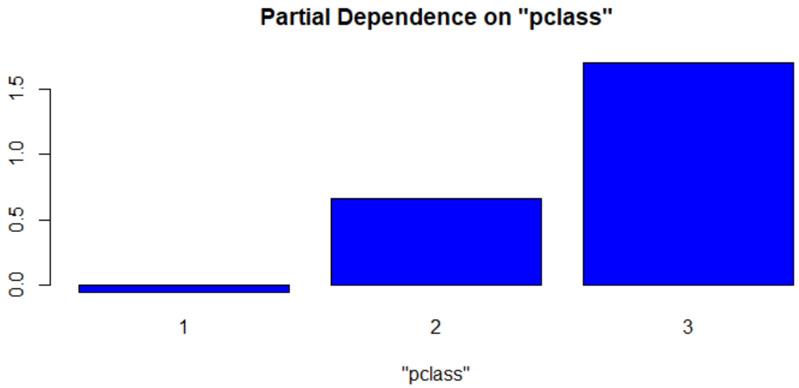
RF: Partial Dependence Plot

```r
partialPlot(RF, pred.data = trdata, x.var="sex")
```



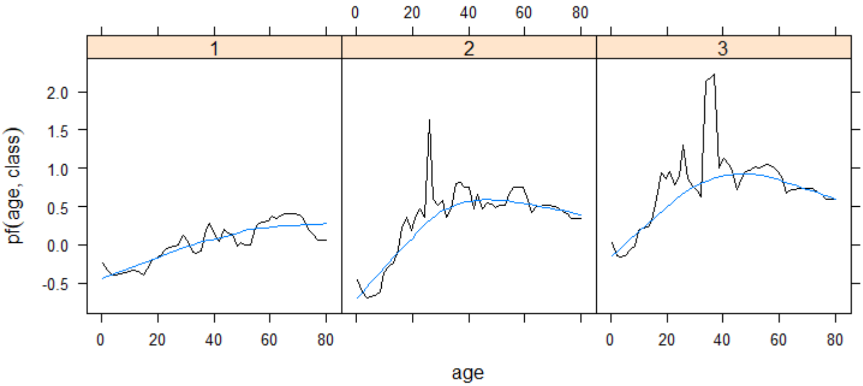
RF: Partial Dependence Plot

```r
partialPlot(RF, pred.data = trdata, x.var="pclass")
```



Partial dependence plot of 2 variables at once

```
RF %>%
partial(pred.var = c("age", "pclass")) %>%
plotPartial(smooth=TRUE, ylab = expression(pf(age, class)))
```

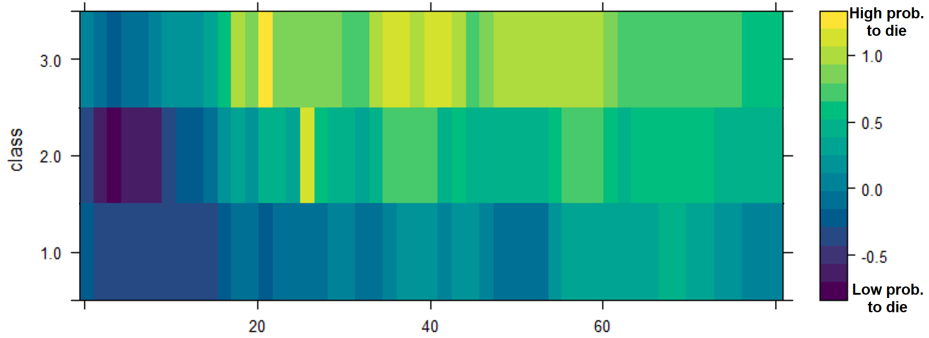


Partial dependence plot of 2 variables at once (heatmap)

```
dataheat <- RF %>%  partial(pred.var = c("age", "pclass"))

ggplot(data = dataheat , aes(x = age, y = pclass, fill = yhat)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", high = "yellow", midpoint = 0
  labs(title = "Partial Dependence Heatmap for age and class")
```

Interpret: Titanic 3rd class die more than 1st class, old die more than young, most deaths for aged 40 in 3rd class, least deaths for kids in 2nd class


class
High prob. to die
Low prob. to die