60611 Advanced Statistical Learning
Tree-Based Methods and Random Forests (RF)

Aurélie Labbe (based on D. Larocque course notes)

HEC Montréal
Sciences de la décision

Agenda

Introduction

1. **Introduction to Tree-Based Methods**
2. Example: Breast Cancer Data
3. Basic Random Forest
4. Forest with Other Tree Building Algorithms
5. Ames Data (continued)
6. German Credit (continued)
7. Variable Importance with Random Forests

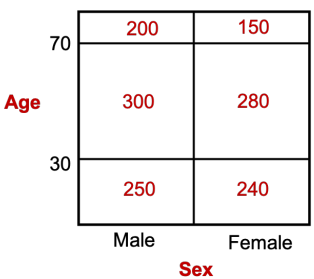
Introduction

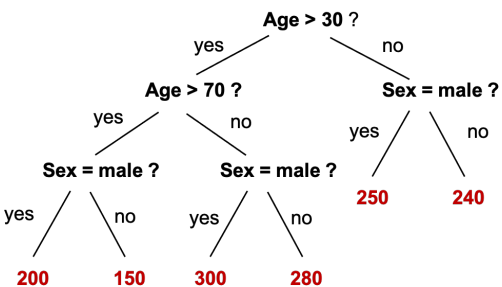
- As before, we have a target variable Y and many covariates $X_1, X_2, \ldots, X_p$.
- The covariates can be of any types: continuous, categorical (binary, nominal, ordinal).
- We assume we have a sample of size n from the target population and use the notation Y_i and X_{ji} to denote the value of Y and X_j for the i^{th} observation in the sample, $i = 1, \ldots, n$ and $j = 1, \ldots, p$.

Tree-Based Methods

- Trees became popular with the classical book "Classification and Regression Trees" (Breiman, 1984).
- The basic idea is to obtain a data-driven partition of the covariates space in order to locally fit the data.

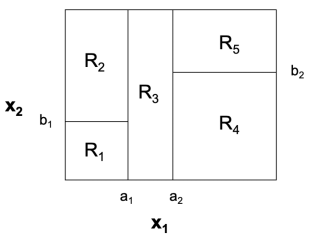
Prediction of salary based on age and sex





Tree-Based Methods (cont')

- Suppose we have a partition of the covariate space into m regions $R_1, \ldots, R_m$.
- By a partition, we mean that each possible value of the vector $\mathbf{X} = (X_1, \ldots, X_p)$ falls in one and only one region.



Tree-Based Methods (cont')

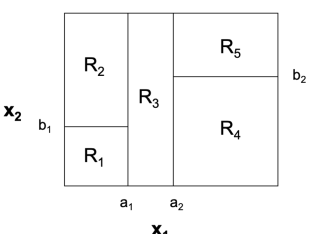
- A possible model to predict Y is

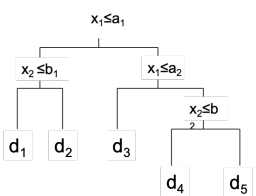
$$\hat{f}(\mathbf{X}) = \sum_{j=1}^{m} d_j I(\mathbf{X} \in R_j),$$

  where $I(\mathbf{X} \in R_j) = 1$ if the condition is true and 0 otherwise.

- Consequently, for a given $\mathbf{X}$, since $R_1, \ldots, R_m$ is a partition, one and only one of the indicator function in the sum will be 1 and all others will be 0.

- Assume that $\mathbf{X}$ is in region R_k, then the prediction, for that $\mathbf{X}$ is d_k.





Tree-Based Methods (cont')

- Three elements need to be specified for such a model:
  - i) the number of regions in the partition m
  - ii) the partition $R_1, \ldots, R_m$
  - iii) the values $d_1, \ldots, d_m$
- It is computationally impossible to search through all possible partitions to find the "best" one.
- Tree-based methods provide a solution to this problem, and in fact, to the problem of specifying all three elements.
- All of them will be estimated from the data, following a recursive partitioning algorithm.

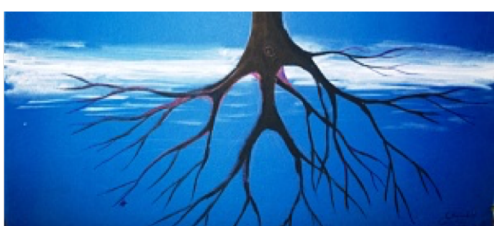
Agenda

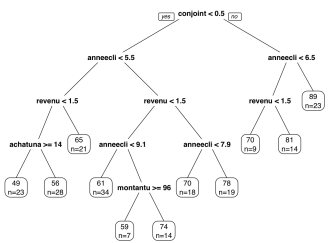
Introduction

1. **Introduction to Tree-Based Methods**
   (a) CART
   (b) Conditional Inference Trees
2. Example: Breast Cancer Data
3. Basic Random Forest
4. Forest with Other Tree Building Algorithms
5. Ames Data (continued)
6. German Credit (continued)
7. Variable Importance with Random Forests

Classification and Regression Trees (CART)

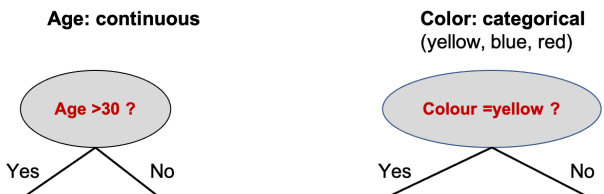
- To fix ideas, we will present the original CART (Classification and Regression Trees) algorithm to build a regression tree, that is a tree for a continuous outcome Y.
- CART proceeds in a greedy and local fashion to try to improve the model at each split.
- Suppose we are at a node t and we want to split it (this node is the parent node), only the observations in the parent node are used.
- Although more general splits are possible, most algorithms restrict themselves to binary splits.





CART: splits depend on covariates

- There are two types of splits depending on the covariate:
  - If a covariate X is continuous (or at least ordered), the possible splits take the form $I(X > c)$ where c is a constant.
  - If the covariate is categorical and unordered (nominal variable), then the possible splits take the form $I(X \in \{c_1, \ldots, c_r\})$ where $\{c_1, \ldots, c_r\}$ is a subset of the possible values of X.
- If the condition is true, then the observation goes in the left node and if it is false, it goes in the right node.



CART: algorithm

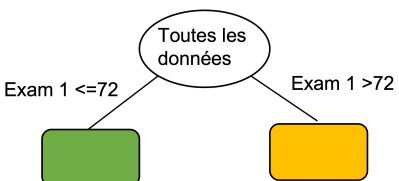
- CART proceeds by evaluating all possible splits on all covariates and selecting the split that has the best value of a criterion.
- Let's illustrate this through a simple example...

CART algorithm: example - 1st split

Quizz	Exam1	Y=Note finale
Reussi	85	90
Echec	70	75
Echec	50	48
Reussi	65	70
Echec	72	77
Reussi	90	98
Reussi	95	90
Echec	78	78
Echec	58	63

Partitions possibles
Quizz = Reussi / Quizz = Echec
Exam 1 <=50 / Exam 1 >50
Exam 1 <=58 / Exam 1 >58
Exam 1 <=65 / Exam 1 >65
Exam 1 <=70 / Exam 1 >70
Exam 1 <=72 / Exam 1 >72
Exam 1 <=85 / Exam 1 >85
Exam 1 <=90 / Exam 1 >90
Exam 1 <=95 / Exam 1 >95

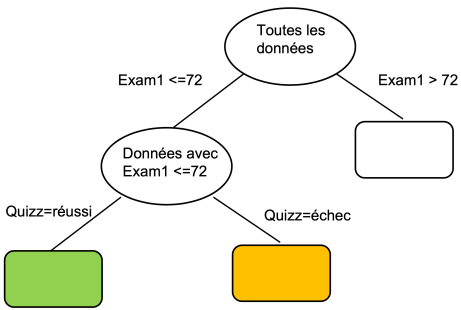
Quizz	Exam1	Y=Note finale
Reussi	85	90
Echec	70	75
Echec	50	48
Reussi	65	70
Echec	72	77
Reussi	90	98
Reussi	95	90
Echec	78	78
Echec	58	63



CART algorithm: example - 2nd split

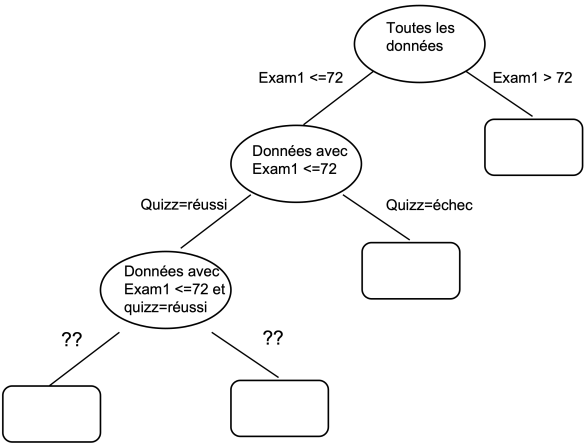
Partitions possibles
Quizz = Reussi / Quizz = Echec
Exam 1 <=50 / Exam 1 >50
Exam 1 <=58 / Exam 1 >58
Exam 1 <=65 / Exam 1 >65
Exam 1 <=70 / Exam 1 >70

Quizz	Exam1	Y=Note finale
Reussi	85	90
Echec	70	75
Echec	50	48
Reussi	65	70
Echec	72	77
Reussi	90	98
Reussi	95	90
Echec	78	78
Echec	58	63



CART algorithm: example - other splits

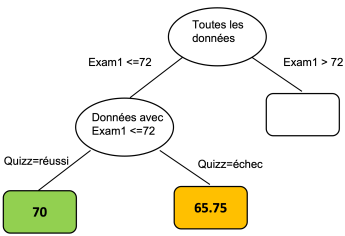
We continue until the tree is complete...



CART: prediction

- The prediction from a tree is the average of the Y in the terminal node.
- That is, to predict an observation, we find in which terminal node it ends up and the prediction is the average of the Y in that terminal node.

Quizz	Exam1	Y=Note finale
Reussi	85	90
Echec	70	75
Echec	50	48
Reussi	65	70
Echec	72	77
Reussi	90	98
Reussi	95	90
Echec	78	78
Echec	58	63



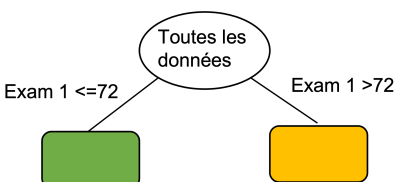
CART: split criterion

- Usually, for a regression tree, the criterion is the **squared-error**.
- Thus, the best split is the one that minimizes

$$\sum_{i \in t_L} (y_i - \bar{y}_L)^2 + \sum_{i \in t_R} (y_i - \bar{y}_R)^2,$$

where t_L (t_R) is the set of indices of the observation that are in the left (right) node, and $\bar{y}_L$ ($\bar{y}_R$) is the average of the observations in the left (right) node.

Quizz	Exam1	Y=Note finale
Reussi	85	90
Echec	70	75
Echec	50	48
Reussi	65	70
Echec	72	77
Reussi	90	98
Reussi	95	90
Echec	78	78
Echec	58	63

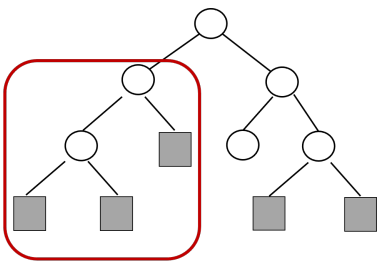


CART: algorithm (cont')

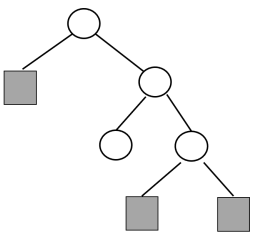
- Note that certain constraint can be imposed on a split to be allowed.
- For example, we might impose that a node has a minimum number of observations.
- Hence the search for the best split is really a search for the best split among the allowable splits.
- **Splitting stops when a stopping criterion is reached**.
- For example, when the parent node has fewer than a certain number of observations, or when the parent node has reached the maximum depth allowed.

CART: tree pruning

- Large trees will usually overfit the data.
- The CART method builds a large tree, and then prunes it to get a "honest" tree that will capture the signal present in the data but not the noise.



Large tree



Pruned tree

CART: tree pruning (cont')

- CART uses cost-complexity pruning.
- For a regression tree, the cost complexity criterion for a tree T is defined by

$$C_\alpha(T) = SSE(T) + \alpha N_T$$

where $SSE(T)$ is the in-sample sum of squares of the error when we predict each training data point by the tree prediction and N_T **is the number of terminal nodes**.

- It is thus similar to a penalized criterion like the AIC, where N_T measures the complexity of the model.
- The parameter α governs the tradeoff between the tree complexity and its goodness of fit.

CART: tree pruning (cont')

- For each α, we can find the subtree (by cutting some branches of the large tree) that minimizes $C_\alpha(T)$, since there is only a finite number of possible subtrees.
- This provides a sequence of subtrees.
- The best value of α is then selected by cross-validation.
- That is, the best α is the one that minimizes the cross-validated sum of squares of the error.
- The 1SE rule can also be used to select the final α.

CART: classification trees

- If the target Y is categorical, with K possible values $1, 2, \ldots, K$, the ideas to build a **classification tree** are the same except that a different splitting and evaluation criterion are used, instead of the least-squares.
- For example, the misclassification error can be used but another measure called the **Gini index** is more popular.
- In a node t, let $\hat{p}_{tk}$ be the proportion of observations (training) with a response value k, $k = 1, 2, \ldots, K$. The Gini index for that node is

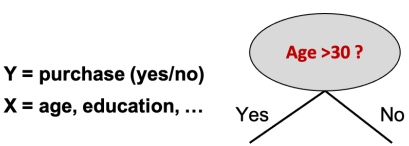
$$G(t) = \sum_{k=1}^{K} \hat{p}_{tk}(1 - \hat{p}_{tk}).$$

CART: classification trees (cont')

- A small value of G indicates that the $\hat{p}_{tk}$'s are more dispersed, that is the node contains predominantly observations from a single class.
- Hence, smaller value of G indicates a purer node which is what we want.
- The best split according to the Gini splitting rule is the one that minimizes the weighted sum of the Gini index

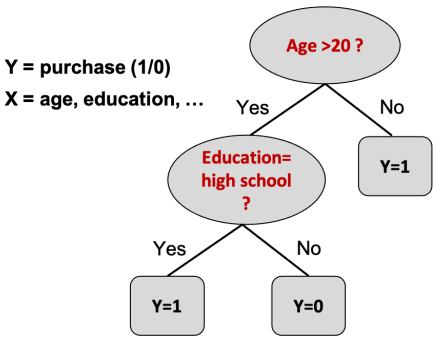
$$n_L G(t_L) + n_R G(t_R),$$

where n_L and $G(t_L)$ are the number of observation and the value of G in the left node, and the two others are the same quantities for the right node.



CART: classification trees prediction

- The prediction at a terminal node of a classification tree is the majority class, that is the value of Y which is the most common is that node.



CART: final notes

- One important property of CART and similar methods is that they are invariant to monotone transformations of the predictor variables.
- If we apply any transformation that preserves the order of a continuous (or ordinal) predictor, then using the transformed variable or the original one will produce the same tree.
- Hence, contrarily to parametric models like linear or logistic regression, we do not need to find a transformation (e.g. the log) of the predictor that fits the data well.
- Moreover, a tree can automatically detect certain types of interactions between the predictors.
- Indeed, all the conditions leading to a terminal node are interactions of binary variables.
- If the root node is the depth 0 node, then at depth 1 we have main effects only, at depth 2 we have order 2 interactions, at depth 3 we have order 3 interactions and so on.

Agenda

Introduction

1. **Introduction to Tree-Based Methods**
   (a) CART
   (b) Conditional Inference Trees
2. Example: Breast Cancer Data
3. Basic Random Forest
4. Forest with Other Tree Building Algorithms
5. Ames Data (continued)
6. German Credit (continued)
7. Variable Importance with Random Forests

Conditional inference trees

- Other types of methods can be used to build trees.
- CART trees can suffer from "biased split selection".
- This means that CART tends to favor splitting on variables that have a lot of possible splits.
  - This is because the variable and split selection are performed simultaneously.
- Example of methods that separate the two operations are the CHAID (Kass et al., 1980) and GUIDE (Loh et al., 2002) methods.
- But in this section we will describe a very general approach, conditional inference trees (Hothorn et al., 2006).

Conditional inference trees: algorithm

- To split a node, the basic algorithm goes as follows.
  1. Test the global null hypothesis of independence between any of the p covariates and the response. Stop if this hypothesis cannot be rejected. The node becomes terminal. Otherwise select the covariate X with strongest association to Y. Suppose it is X_j.
  2. Split the node with X_j according to a criterion.
- The procedure stops when all (non-internal) nodes are terminal.
- No pruning are required because the process proceeds in a forward fashion. Deciding or not to split a node is based on a statistical test.

Conditional inference trees: algorithm step 1

- In step 1) we need to test the global null hypothesis of independence between any of the p covariates and the response
- If the hypothesis is rejected, we need to assess the association between each covariate and the response to select the one with the strongest association.
- This is done in a very general way with a, possibly multivariate, linear statistic.
- For simplicity, we describe a simplified version here.

Conditional inference trees: algorithm step 1 (cont')

- Let t be the set of indices in a node. Measuring the association between one covariate X_j and Y is done with

$$T_j = \sum_{i \in t} g_j(X_{ji}) h(Y_i)$$

  where g_j is a transformation of X_j and h is a transformation of Y.

- If X_j are Y are continuous, then using $g_j(x) = x$ and $h(x) = x$ entails $T_j = \sum_{i \in t} X_{ji} Y_i$ which is equivalent to using Pearson correlation to measure the association.

- But the formulation is much more general and allows using ranks, for example.

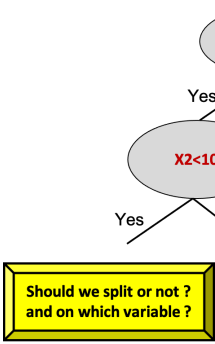
- That is, $g_j(X_{ji})$ is the rank of X_{ji} among the X_j's in the node. If we take the ranks too for Y, then T_j is equivalent to using Spearman correlation to measure the association.

Conditional inference trees: algorithm step 1 (cont')

- Then we can compute the p-value corresponding to the hypothesis that there is no association between X_j and Y, in this node.
- The best covariate is then the one with the smallest p-value.
- Using p-values are important because it puts all covariates on the same scale for comparison.
- But this is not the end...
- We must decide if the node is to be split or not, using the seleted covariate.
- This is done by performing a **global test** based on the individual tests and, preferably, one that controls for multiple testing like the Bonferroni method.
- Thus, to test the global hypothesis, we use adjusted p-values. If the minimum of the adjusted p-values is less than a pre-specified level, then we split the node, otherwise, it becomes a terminal node.

Conditional inference trees: summary of the algorithm

Example of an outcome Y and 5 X variables



- **Compute**:
  - ✓ $T_1 = Cor(Y, X_1) \rightarrow$ p-value1
  - ✓ $T_2 = Cor(Y, X_2) \rightarrow$ p-value2
  - ✓ …
  - ✓ $T_5 = Cor(Y, X_5) \rightarrow$ p-value5
- **Correct p-values for multiple testing**
  - ✓ p-value 1* = p-value1 * 5
  - ✓ p-value 2* = p-value2 * 5
  - ✓ …
  - ✓ p-value 5* = p-value5 * 5
- **Should we split or not ?**
  - ✓ Yes if min(p-value*) < 0.05
- **If YES, on which variable ?**
  - ✓ On the variable corresponding to min(p-value)

Conditional inference trees: final remarks

- Note that, contrarily to CART, this tree building method is in general not invariant to monotone transformations of the predictor variables.
- This is because the tests of hypothesis depend on the transformations used.
- The packages `party` (Hothorn et al., 2006) and `partykit` implement this general method and have numerous options for transformations and to perform the tests.
- These packages can also do model-based recursive partitioning (mob), where a model (e.g. linear or logistic regression) is fit in a node, instead of simply computing an average (for a regression tree) (Zeileis et al., 2008)

Agenda

Introduction

1. Introduction to Tree-Based Methods
2. **Example: Breast Cancer Data**
3. Basic Random Forest
4. Forest with Other Tree Building Algorithms
5. Ames Data (continued)
6. German Credit (continued)
7. Variable Importance with Random Forests

Breast cancer data

- To illustrate trees, we use the Breast Cancer data from the `mlbench` package.
- See `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Original)` for details.

Code R

```
> library(mlbench)
> data(BreastCancer)
> summary(BreastCancer)
      Id               Cl.thickness   Cell.size     Cell.shape  Marg.adhesion
 Length:699         1       :145   1       :384   1       :353   1       :407
 Class :character   5       :130   10      : 67   2       : 59   2       : 58
 Mode  :character   3       :108   3       : 52   10      : 58   3       : 58
                    4       : 80   2       : 45   3       : 56   10      : 55
                    10      : 69   4       : 40   4       : 44   4       : 33
                    2       : 50   5       : 30   5       : 34   8       : 25
                    (Other):117   (Other): 81   (Other): 95   (Other): 63
  Epith.c.size  Bare.nuclei   Bl.cromatin  Normal.nucleoli    Mitoses
 2       :386   1       :402   2       :166   1       :443   1       :579
 3       : 72   10      :132   3       :165   10      : 61   2       : 35
 4       : 48   2       : 30   1       :152   3       : 44   3       : 33
 1       : 47   5       : 30   7       : 73   2       : 36   10      : 14
 6       : 41   3       : 28   4       : 40   8       : 24   4       : 12
 5       : 39   (Other): 61   5       : 34   6       : 22   7       :  9
 (Other): 66   NA's    : 16   (Other): 69   (Other): 69   (Other): 17
      Class
 benign    :458
 malignant :241
```

Breast cancer data (cont')

- The target variable is Class (binary).
- Nine predictors are available since the ID variable is of no use here.
- They could be treated as numeric but right now they are seen as categorical in the data frame.
- They all have at most 10 different values and one of them (Bare.nuclei) has 16 missing values.
- We will keep them as categorical for this example.

Breast cancer data (cont')

- We begin by splitting the data set ($n = 699$) into a training data set of size 349 and a test data set, treated as new data not available at training time, of size 350.

Code R

```
> # split data into training (n=349) and test data (n=350)
> set.seed(68576897)
> indtrain=sample(1:NROW(BreastCancer),349,replace=FALSE)
> trainbc=BreastCancer[indtrain,]
> testbc=BreastCancer[-indtrain,]
```

Breast cancer data: fitting a CART

- We start with `rpart`.
- We first grow a large tree that we will prune afterwards.
- The arguments `rpart.control` allows control over the tree building.

Code R

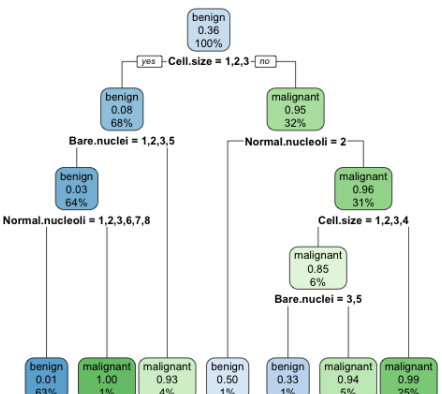
```
> library(rpart)
> library(rpart.plot)
> rptree=rpart(Class~Cl.thickness + Cell.size + Cell.shape + Marg.adhesion +
+                  Epith.c.size + Bare.nuclei + Bl.cromatin + Normal.nucleoli + Mitoses,
+              data=trainbc,method="class",
+              control = rpart.control(xval = 10, minsplit=10, minbucket = 3, cp = 0))
```

Breast cancer data: fitting a CART (cont')

- We plot the tree with the package `rpart.plot` which produces nicer plots. Here is the fully grown tree.

Code R

```
> rpart.plot(rptree)
```

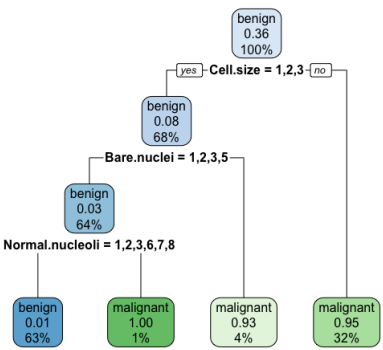


Breast cancer data: fitting a CART (cont')

- Then we select the tuning parameter that minimizes the cross-validated error, and prune the tree.

Code R

```
> rptreepruned=prune(rptree, cp=rptree$cp[which.min(rptree$cp[,"xerror"]),"CP"])
> rpart.plot(rptreepruned)
```



Breast cancer data: fitting a CART (cont')

- We then compute the predictions for the new data and compute the good classification rate, which is 94.9%.

Code R

```
> predrpart=predict(rptreepruned,newdata=testbc,type="class")
> mean(predrpart==testbc$Class)
[1] 0.9485714
```

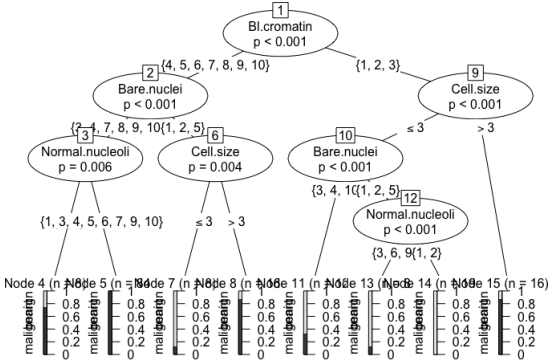
- Note that we simply use a threshold of 0.5 for assigning the classes. This is what the `predict` function does by default.
- We could try to find a better threshold but since a single tree is rarely used for predictions, we use 0.5 here as a simple illustration.

Breast cancer data: fitting a conditional inference tree

- We can do the same thing with a conditional inference tree with `party`.
- We first build and plot the tree with the default parameters.

Code R

```
> library(party)
> cttree=ctree(Class~Cl.thickness + Cell.size + Cell.shape + Marg.adhesion +
+        Epith.c.size + Bare.nuclei + Bl.cromatin + Normal.nucleoli + Mitoses, data=trainbc)
> plot(cttree)
```



Breast cancer data: fitting a conditional inference tree (cont')

- We then compute the predictions for the new data and compute the good classification rate, which is 94.8%.

Code R

```
> plot(cttree)
> predct=predict(cttree,newdata=testbc,type="response")
> mean(predct==testbc$Class)
[1] 0.9457143
```

Agenda

Introduction

1. Introduction to Tree-Based Methods
2. Example: Breast Cancer Data
3. **Basic Random Forest**
4. Forest with Other Tree Building Algorithms
5. Ames Data (continued)
6. German Credit (continued)
7. Variable Importance with Random Forests

About trees...

- Trees have many advantages. Here are a few:
  - They can handle any types of covariates
  - They can model any types of target Y
  - They can detect certain types of interactions automatically
  - They scale well to large sample sizes
  - Small trees are easy to interpret.
- However, a single tree can often be beaten by other methods in terms of prediction performance.
- Moreover, the interpretability of trees is quickly lost when the tree is large.

About forets...

- Fortunately, it is well established that combining many trees can often improve drastically the prediction performance of a single tree and in fact these "ensemble" methods are often among the best performer in terms of prediction accuracy.
- On the downside, ensemble of trees are difficult to interpret but, as just mentioned, a large single tree cannot easily be interpreted anyway.
- There are many ways to combine trees.
- The two most popular are random forests (Breiman, 2001) and boosting, which will be explored in the next chapter.

Instability of trees and random forets

- The original motivation for random forests (RF) was that a single tree is an unstable learner, in the sense that slight modifications in the data set can produce a very different tree.
- This is easy to understand since at a given node, there are often many potential splits that are as good, with respect to the splitting rule.
- Hence, a slightly different data set (than the original) might have selected a different split.
- But then this can have a cascading effect for the rest of the tree.

Instability of trees and random forets (cont')

- Hence, a large tree can be seen as a learner with potentially a small bias and a large variance.
- The bias will likely be small because a large tree fits the data locally.
- However, the instability of trees is likely to induce a large variance in the prediction.
- Breiman's idea was to add some randomness to the tree growing algorithm and to repeat the process many times to stabilize the results.

Original Random Forest Algorithm

- Assume we have p covariates.
- Select the number of trees B, and the number of covariates $p_0 \leq p$, to select at random at a node to find the best split.
- For $b = 1$ to B:
  1. Create a bootstrap sample from the original data.
  2. Build a tree with the bootstrap sample (large trees are usually built and no pruning is performed).
     - At each node, select at random p_0 out of the p covariates and find the best split with these covariates only.
     - Note that the subset of covariates can vary from node to node.
     - Let $\hat{T}_b(x)$ be this tree.

Original Random Forest Algorithm: final prediction model

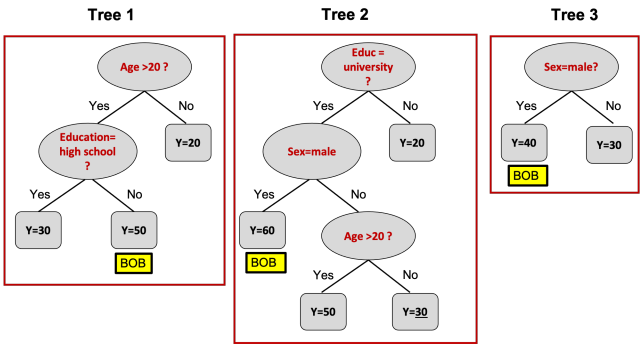
- The final prediction model is the average of all trees in the forest. of the target.
- For a continuous target, each tree returns a predicted value (which is the average of the observations in the terminal node). **The final prediction model is the average of these predictions**, that is

$$\hat{T}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{T}_b(x).$$

- For a categorical target, there are different ways to combine the trees. The first one is to get the prediction from each tree and **define the forest prediction as the majority vote** (the class with the most votes among all trees). **Alternatively, we can obtain the proportion of each class in each tree and average them**. The forest prediction is then the class with the highest averaged proportion.
- More complex situations, like survival data with censoring will be discussed in another chapter.

Recap: final prediction for a continuous variable

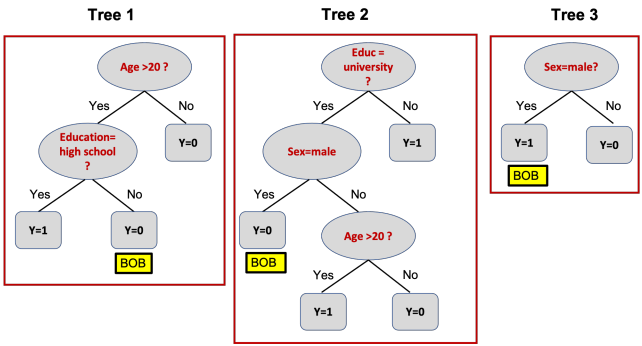
- **Example**: $B = 3$ trees, outcome Y is salary
- **Predict Bob's salary**: Bob is a male, 35 years old, with a university degree



- The prediction for Bob is the average of (50,60,40) = 50$

Recap: final prediction for a categorical variable

- **Example**: $B = 3$ trees, outcome Y is purchase (yes/no ; 0/1)
- **Predict Bob's purchase**: Bob is a male, 35 years old, with a university degree



- The prediction for BOB is the majority of (0,0,1) = 0

More on random forests

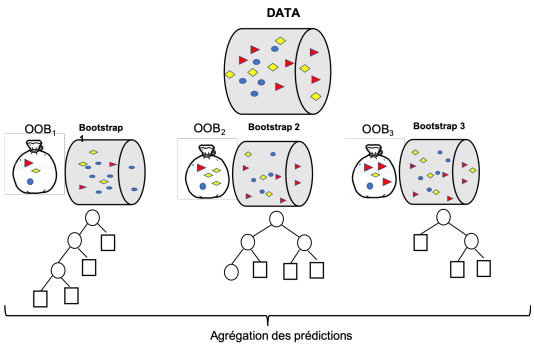
- There are two sources of randomness in the RF algorithm.
- The first one is to use a new boostrap sample for each tree.
- The second one is to select a subset of covariates at each node.
- This second source of randomness has another benefit : it speeds up the computations since less splits need to be evaluated.
- The default values of p_0 in many packages, like `randomForest`, are $p/3$ for a regression forest, and $\sqrt{p}$ for a classification forest. But this parameter can be selected with the data themselves via data splitting or cross-validation.

Random forests versus bagging

- Note that bagging (Breiman 1996) is the "ancestor" of RF.
- Bagging is the particular case of RF, where we use all covariates p to find the best split at each node, and not a sample of them.
- Very often, using $p_0 < p$ produces better results. This is due to the fact that the resulting trees are less correlated because of this added randomness. They are thus able to find more structure in the data.
- RF is thus in a win-win situation over bagging because it 1) speeds up computations and 2) perform better most of the time.

Random forests and OOB

- One very useful by-product of a RF are the OOB (Out-Of-Bag) observations.
- For a given tree, the OOB data are the observations that are not part of the bootstrap sample (they are not used to built the tree).



- On average, about 37% of the observations are not part of the boostrap sample.

Random forests and OOB (cont')

- For a given tree, the OOB data can be used as test data.
- We can thus get an out-of sample prediction for a training data point by computing the forest prediction for this point, but only using the trees where it is OOB.
- These are called the OOB predictions.
- For example, the OOB prediction for observation i with a regression forest is

$$\hat{T}_{\text{oob}}(x_i) = \frac{1}{B_i} \sum_{b=1}^{B} \hat{T}_b(x_i) I(x_i \in \text{OOB}_b),$$

  where B_i is the number of times that observation i is in the OOB sample among the B trees, and $I(x_i \in \text{OOB}_b)$ is an indicator taking a value of 1 if observation i is in the OOB sample for tree b, OOB_b.
- Hence the OOB prediction for observation i is simply the average of the tree predictions, **but only for the trees for which it is OOB** (i.e. the trees for which it was not used to build the tree).

Random forests and OOB (cont')

- Hence, **instead of using data splitting or cross-validation**, we can use the OOB information to, for instance, optimize some parameters (like p_0).
- But we must remember that the OOB predictions are, on average, obtained from a forest with $0.37B$ trees.
- Hence, we might want to increase the number of trees to get more accurate OOB predictions.

Implementations of the Original Random Forest Algorithm

- Many packages can build random forests using the original method based on the CART paradigm.
- The original one is the package `randomForest`
- One of the most comprehensive, in terms of options and features, is `randomForestSRC` .
- A fast implementation of RF is provided in the package `ranger`.

Agenda

Introduction

1. Introduction to Tree-Based Methods
2. Example: Breast Cancer Data
3. Basic Random Forest
4. **Forest with Other Tree Building Algorithms**
5. Ames Data (continued)
6. German Credit (continued)
7. Variable Importance with Random Forests

Forest with Other Tree Building Algorithms

- The original random forest was based on the CART paradigm.
- But the idea of building many trees on bootstrap samples and then combining the results is very general. In principle, any tree building algorithm can be used.
- We introduced the conditional inference tree method previously, as another tree building paradigm.
- We illustrated the use of `ctree` in the `party` package. The function `cforest` in the same package can build a forest of conditional inference trees.
- However, the whole package is coded in R and building forests (and even a single tree) is not as fast as the other implementations of RF, which are usually coded in C (C++).

Forest with Other Tree Building Algorithms

- Remember that conditionnal inference trees were introduced to overcome the potential bias in split selection that CART can have.
- However, it seems that the potential split selection bias present in CART has no impact on the predictive performance when building a forest.
- See a HEC MSc thesis "Comparaison des performances prédictives de deux algorithmes construisant des forêts aléatoires"
- In this thesis, the student used real data and simulated data to compare the classical random forest and a forest of conditional inference trees.
- Results show that `randomForest` is generally better or at least as good as `cforest` in a wide variety of situations.

Agenda

Introduction

1. Introduction to Tree-Based Methods
2. Example: Breast Cancer Data
3. Basic Random Forest
4. Forest with Other Tree Building Algorithms
5. **Ames Data (continued)**
6. German Credit (continued)
7. Variable Importance with Random Forests

Ames data: CART

- We already analysed these data with the lasso and related methods.
- We will now use a random forest.
- But before, we quickly show the result of using a single tree with `rpart`.

Code R

```
> library(rpart)
> set.seed(37569)
> rptree=rpart(Sale_Price~.,data=amestrain,method="anova",
+              control = rpart.control(xval = 10, minsplit=10, minbucket = 3, cp = 0))
> rptreepruned=prune(rptree,cp=rptree$cp[which.min(rptree$cp[,"xerror"]),"CP"])
> predrpart=predict(rptreepruned,newdata=amestest)
> errrpart=data.frame(mean(abs(predrpart-amesdumtest$Sale_Price)),
+                     mean((predrpart-amesdumtest$Sale_Price)^2))
> names(errrpart)=c("MAE","MSE")
> row.names(errrpart)=c("single tree (rpart)")
> errrpart
                          MAE  MSE
single tree (rpart) 25.35748 1452
```

Ames data: CART (cont')

- We see that the performance is not good compared to the regularized linear regression of the previous chapter.

Code R

```
> allres=rbind(allres,errrpart)
> allres
                          MAE       MSE
lasso                18.04351  901.9403
ridge                17.83458  889.6937
OLS                  19.64333 1278.4017
lasso-OLS            18.43752  956.1045
lasso-lasso          18.27340  941.5744
relaxed lasso        18.12304  904.1166
single tree (rpart) 25.35748 1452.0004
```

Ames data: random forest

- Now let's fit a random forest with `randomForest`.

Code R

```
> set.seed(33967)
> library(randomForest)
> rf=randomForest(Sale_Price~.,data=amestrain,ntree=500)
> predrf=predict(rf,newdata=amestest)
> errrf=data.frame(mean(abs(predrf-amesdumtest$Sale_Price)),
+                  mean((predrf-amesdumtest$Sale_Price)^2))
> names(errrf)=c("MAE","MSE")
> row.names(errrf)=c("random forest")
> errrf
                   MAE      MSE
random forest 16.64009 701.4298
```

Ames data: random forest (cont')

- However, a random forest performs a lot better.
- In fact, its MAE and MSE on the test set is better than the best one we got with regularized linear regression.
- The difference between a single tree and a RF is striking in this example.

Code R

```
> allres=rbind(allres,errrf)

# Best methods so far
> allres[order(allres[,1]),]
                         MAE       MSE
random forest       16.64009  701.4298
ridge               17.83458  889.6937
lasso               18.04351  901.9403
relaxed lasso       18.12304  904.1166
lasso-lasso         18.27340  941.5744
lasso-OLS           18.43752  956.1045
OLS                 19.64333 1278.4017
single tree (rpart) 25.35748 1452.0004
```

Final comments

- By default, for a continuous Y, `randomForest` uses a value of `mtry` (the number of covariates chosen at random in a node) of $p/3$.
- This is just a convention and it is not clear that this is always the best choice.
- Sometimes, using other values of `mtry` can lead to substantial improvements (see Hamza, 2005)
- Here, with the Ames data, we tried a few other values and were able to get notably better results.
- In practice, this parameter could be selected by using the OOB error estimation, without the need to perform cross-validation.

Final comments (con't)

- The packages `caret` and `caretEnsemble` integrate many learning algorithms in a unified framework and have many features to perform tuning parameter optimization.
- The list of available methods is given here: `https://topepo.github.io/caret/available-models.html`

Agenda

Introduction

1. Introduction to Tree-Based Methods
2. Example: Breast Cancer Data
3. Basic Random Forest
4. Forest with Other Tree Building Algorithms
5. Ames Data (continued)
6. **German Credit (continued)**
7. Variable Importance with Random Forests

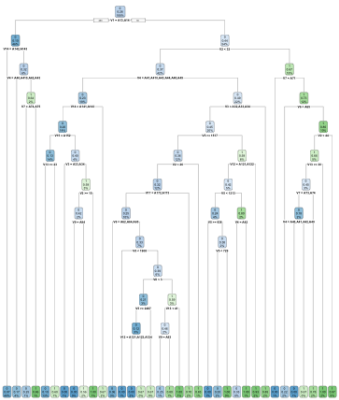
German credit data: CART

- This time the target is binary, we analysed these data earlier with a lasso logistic regression.
- We first use a single `rpart` tree.

Code R

```
> library(rpart)
> library(rpart.plot)
> set.seed(46576)
> # building the tree with rpart
> rptreegc=rpart(V21~.,data=gercredtrain,method="class",
+                control = rpart.control(xval = 10, minsplit=10, minbucket = 3, cp = 0))
> rpart.plot(rptreegc)
```

German credit data: CART (cont')



- We see that the tree is fairly large with many terminal nodes and is not easy to interpret.
- It will probably overfit the data, we can prune it.

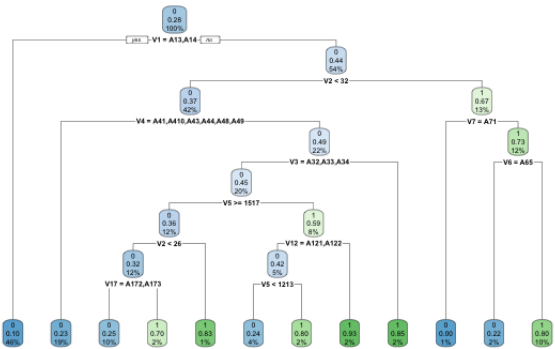
German credit data: CART (cont')

Code R

```
> rptreegc$cp
             CP nsplit rel error    xerror       xstd
1  0.076023392      0 1.0000000 1.0000000 0.06466287
2  0.035087719      2 0.8479532 0.8888889 0.06230015
3  0.029239766      3 0.8128655 0.8654971 0.06174877
4  0.026315789      4 0.7836257 0.8538012 0.06146567
5  0.023391813      9 0.6198830 0.8011696 0.06012832
6  0.017543860     11 0.5730994 0.7836257 0.05965868
7  0.013645224     12 0.5555556 0.8187135 0.06058585
8  0.008771930     15 0.5146199 0.8479532 0.06132223
9  0.005847953     23 0.4385965 0.8888889 0.06230015
10 0.000000000     34 0.3625731 0.9415205 0.06347073

> # pruning the tree
> rptreegcpruned=prune(rptreegc,cp=rptreegc$cp[which.min(rptreegc$cp[,"xerror"]),"CP"])
> rpart.plot(rptreegcpruned)
> # getting the predictions (with a threshold of 0.5)
> predrpartgc=predict(rptreegcpruned,newdata=gercredtest,type="class")
> mean(as.factor(gercredtest$V21)==predrpartgc)
[1] 0.7
```

German credit data: CART (cont')



- The pruned tree is simpler with 12 terminal nodes.

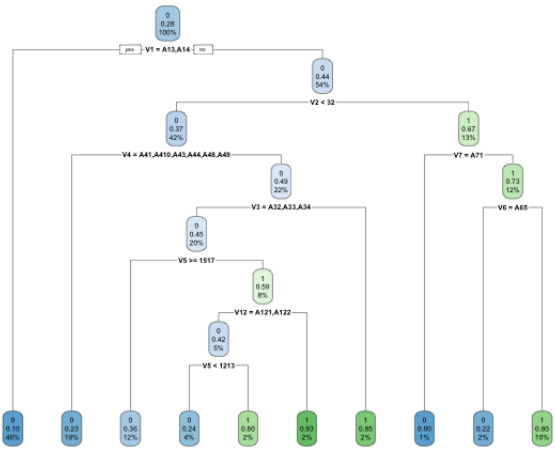
German credit data: CART (cont')

- We can go a step further and use the 1 SE rule.

Code R

```
> # pruning the tree with the 1SE rule
> rptreegc$cp
           CP nsplit rel error    xerror       xstd
1  0.076023392      0 1.0000000 1.0000000 0.06466287 0
2  0.035087719      2 0.8479532 0.8888889 0.06230015 0
3  0.029239766      3 0.8128655 0.8654971 0.06174877 0
4  0.026315789      4 0.7836257 0.8538012 0.06146567 0
5  0.023391813      9 0.6198830 0.8011696 0.06012832 1
6  0.017543860     11 0.5730994 0.7836257 0.05965868 1
7  0.013645224     12 0.5555556 0.8187135 0.06058585 1
8  0.008771930     15 0.5146199 0.8479532 0.06132223 0
9  0.005847953     23 0.4385965 0.8888889 0.06230015 0
10 0.000000000     34 0.3625731 0.9415205 0.06347073 0
> bstd=rptreegc$cp[which.min(rptreegc$cp[,"xerror"]),"xstd"]
> bstd
[1] 0.05965868
> berr=rptreegc$cp[which.min(rptreegc$cp[,"xerror"]),"xerror"]
> berr
[1] 0.7836257
> rptreegc$cp=cbind(rptreegc$cp,rptreegc$cp[,"xerror"]<=(berr+bstd))
> cp1se=
+     rptreegc$cp[rptreegc$cp[,6]==1,][which.min(rptreegc$cp[rptreegc$cp[,6]==1,
+                                                    "nsplit"]),"CP"]
> rptreegcpruned1se=prune(rptreegc,cp=cp1se)
> rpart.plot(rptreegcpruned1se)
> predrpartgc1se=predict(rptreegcpruned1se,newdata=gercredtest,type="class")
> mean(as.factor(gercredtest$V21)==predrpartgc1se)
[1] 0.705
```

German credit data: CART (cont')



- The tree now has 10 terminal nodes.

German credit data: random forest

- We can use a random forest with 500 trees.

Code R

```
> set.seed(4856767)
> library(randomForest)
>
> rfgc=randomForest(as.factor(V21)~.,data=gercredtrain,ntree=500)
> rfgc

Call:
 randomForest(formula = as.factor(V21) ~ ., data = gercredtrain,      ntree = 500)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 22.83%
Confusion matrix:
    0  1 class.error
0 398 31  0.07226107
1 106 65  0.61988304
```

- The estimated OOB good classification rate is 0.7717 (1-0.2283).

German credit data: random forest (cont')

Code R

```
> # to get the estimated probabilities on the test set
> predrfgc=predict(rfgc,newdata=gercredtest,type="prob")
> predrfgc[1:10,]
        0     1
3   0.918 0.082
7   0.952 0.048
9   0.946 0.054
10  0.646 0.354
14  0.648 0.352
16  0.502 0.498
18  0.480 0.520
19  0.276 0.724
22  0.824 0.176
25  0.934 0.066
> # to get the 0-1 predictions. Uses a threshold of 0.5 by default
> predrfgc01=predict(rfgc,newdata=gercredtest)
> predrfgc01[1:10]
 3  7  9 10 14 16 18 19 22 25
 0  0  0  0  0  0  1  1  0  0
Levels: 0 1
> # good classification rate
> mean(as.factor(gercredtest$V21)==predrfgc01)
[1] 0.7325
```

- The good classification rate on the test set is 0.7325.

German credit data: random forest (cont')

- Although we saw earlier that the best threshold is close to 0.5, just as an example, we can try to find a better threshold with the function `bestcutp` described in the previous chapter.

Code R

```
> # try to find a better threshold
> # get the OOB predictions
> predoob=predict(rfgc,type="prob")
> # find the best threshold
> res=bestcutp(predoob[,2],gercredtrain$V21,gainmat=diag(2),
+              cutp=seq(0,1,.02),plotit=TRUE)
> res[[2]]
cutp gain
0.44 0.78
```

- The best threshold is estimated to be 0.44.
- Using the best threshold 0.44, the estimated good classification rate is 0.78, obtained by CV.

German credit data: random forest (cont')

Code R

```
> # get the new predictions with this threshold
> predrfgc01v2=as.numeric(predrfgc[,2]>res[[2]][1])
> predrfgc01v2[1:10]
 [1] 0 0 0 0 0 0 0 1 1 0 0
> # good classification rate
> mean(gercredtest$V21==predrfgc01v2)
[1] 0.7475
```

- Using the best threshold of 0.44 produces a good classification rate of 0.74 on the test set.

German credit data: random forest (cont')

- The good classification rate we obtained is a little bit better than the rate that we got from the lasso logistic regression in the previous chapter, and the single tree rates that we just saw.
- Remember that the naive rule of classifying everyone in the same class would get a good classification rate of 0.685.

Remarks

- We must be careful with the `predict` function if we use it to get predictions for the training data set.
- Using predict as above, `predoob=predict(rfgc,type="prob")`, produces the OOB predictions. These are the right ones because we do not want to use the same observations that served to build the tree.
- If we use predict like below, with `predinbag=predict(rfgc,newdata=gercredtrain)`, `randomForest` interprets it as if we want to predict new data and it uses the basic predictions, that use all the training data set. We see here that we get a good classification rate of 1, which is overly optimistic.

Code R

```
> # be careful and not use the in-bag predictions
> #  with the training data
> # we have a good classification rate of 1
> # which is overly optimistic!
> predinbag=predict(rfgc,newdata=gercredtrain)
> mean(as.factor(gercredtrain$V21)==predinbag)
[1] 1
```

Agenda

Introduction

1. Introduction to Tree-Based Methods
2. Example: Breast Cancer Data
3. Basic Random Forest
4. Forest with Other Tree Building Algorithms
5. Ames Data (continued)
6. German Credit (continued)
7. **Variable Importance with Random Forests**

Variable Importance with Random Forests

- At first, a random forest is a black-box type of method.
- It is not obvious to know which covariates are important and how they are linked to the target variable.
- In the original RF article (Breiman 2001) the author proposed methods to evaluate the importance of the covariates.
- Many other such variable importance (VIMP) measures were developed since then.
- These methods can be used to assess the relative performance of the variables to predict the target variable.
- But we can go one step further and use them to perform variable selection with a RF, as we will see in a subsequent chapter...

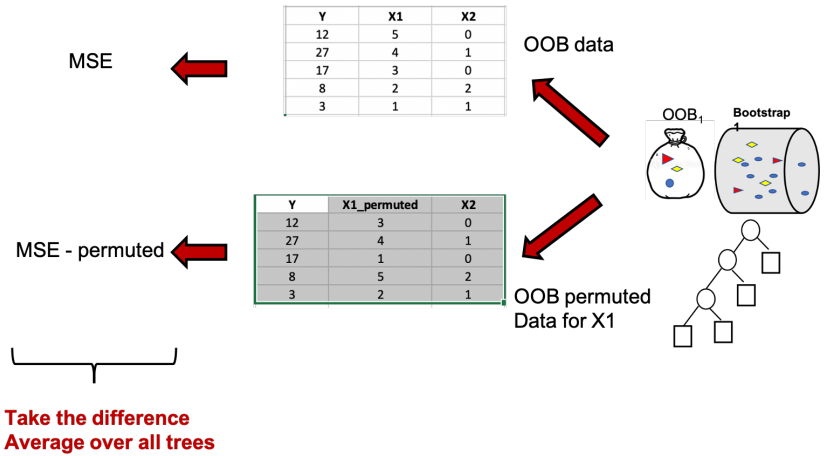
VIMP measures

- One of the original VIMP measure proposed by Breiman (2001) is based on the following idea...
- A variable X is important for predicting Y if the prediction error increases when the link between X and Y is broken.
- To fix ideas, we look at the regression forest case with a continuous Y. Suppose we build a forest with B trees.
- Recall that the out-of-bag (OOB) samples are the samples of observations not used to build the trees.
- There is one OOB sample per tree. Call this sample OOB_b for the b^{th} tree.
- One way to break the link between a variable X_j and Y is to permute the values of X_j is the OOB sample.

VIMP measures (cont')

- Define by OOB_b^j, the OOB sample for the b^{th} tree where the order of X_j has been randomly permuted.
- There are p (the number of covariates) permuted OOB samples per tree and hence, there are pB permuted OOB samples in all.
- For each variable and each tree, we can compute the error using the original OOB sample of the tree and the error using the OOB sample of the tree permuted for this variable.
- If the difference is large, this means that the link between this variable and Y was important for this tree.
- The final VIMP is the average of these over the trees.

VIMP measures: summary



- The difference in error (MSE) between the permuted OOB sample and the original OOB sample is computed for each tree and then we take the average over all trees. The larger the value of $VIMP(X_j)$ is, the more important X_j is.

VIMP measures (cont')

- Formally, let $\hat{f}$ be a prediction model and D a validation sample. Define

$$MSE(\hat{f}, D) = \frac{1}{D}\sum_{i \in D}(Y_i - \hat{f}(X_i))^2,$$

  to be the average squared error of the prediction model for this validation sample.

- The VIMP for variable X_j is

$$VIMP(X_j) = \frac{1}{B}\sum_{b=1}^{B}[MSE(\hat{f}_b, OOB_b^j) - MSE(\hat{f}_b, OOB_b)] \qquad (1)$$

  where $\hat{f}_b$ is the tree prediction model for the b^{th} bootstrap sample.

- This VIMP is available in the packages `randomForest` and `randomForestSRC`.

- Note that the actual VIMP reported by software and packages can vary.

- For instance, the package `randomForest` standardizes the VIMP.

VIMP measures (cont')

- Other VIMP available in `randomForestSRC` are the following.
- Instead of permuting the variable X_j, another way to break its link with Y is to assign 1) the node at random or 2) choose systematically the other node, each time a split is made with that variable.
- The idea is again to compare the OOB error estimate with and without the perturbation.
- The formula is the same as before, except that $MSE(\hat{f}_b, OOB_b^j)$ is replaced by the OOB error obtained from predicting the OOB data with the modified trees.
- `randomForestSRC` also proposes ensemble versions of these VIMP that are faster to compute because the perturbations are performed globally at the forest level and not at the individual tree level.

VIMP measures (cont')

- We continue the Ames data example by computing the VIMP with `randomForest`.
- We already built the RF earlier, it is stored in the object `rf`. Here is how to get and/or to plot of the VIMP (for the top 30 variables by default).

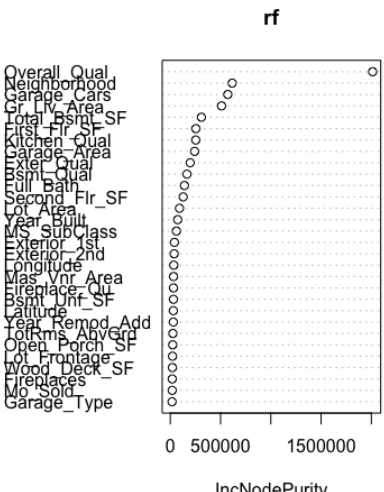
Code R

```
> virf=importance(rf)
> virf
                  IncNodePurity
MS_SubClass        6.095905e+04
MS_Zoning          5.032641e+03
Lot_Frontage       2.234134e+04
Lot_Area           9.163769e+04
Street             6.764867e+01
Alley              8.560377e+02
Lot_Shape          3.007480e+03
Land_Contour       3.006001e+03
Lot_Config         5.507863e+03
Land_Slope         2.987817e+03
Neighborhood       6.159242e+05
Condition_1        6.300707e+03
Condition_2        2.159994e+03
Bldg_Type          3.241823e+03
House_Style        6.785360e+03
(...)
```

VIMP measures (cont')

Code R

```
> varImpPlot(rf)
```



VIMP measures (cont')

- We see that the variable Overall_Qual is the most important according to the VIMP.
- Out of curiosity, we could explore if the VIMP are related to the correlations between the target variable and the covariates.
- To do it, we use the data set with the dummy variables for the categorical covariates, because it makes no sense to compute a correlation with a categorial covariate.
- We compute the absolute value of the correlation between each covariate and the target Sale_price, and order them in decreasing order.
- The top 30 variables with the highest correlations are shown next.

VIMP measures (cont')

Code R

```
# sort the absolute value correlations between
#   the target and the other variables
#   using the dummy variables for the categorical
#   variables
> library(corrr)
> co=correlate(amesdumtrain)
> coy=as.data.frame(focus(co,Sale_Price))
> coy[,2]=abs(coy[,2])
> coys=coy[order(-coy[,2]),,drop=FALSE]
> coys[1:30,]
                                term Sale_Price
6                       Overall_Qual  0.7889939
27                       Gr_Liv_Area  0.7109443
11                        Exter_Qual  0.6806597
34                      Kitchen_Qual  0.6633875
40                       Garage_Cars  0.6415488
41                       Garage_Area  0.6391310
21                     Total_Bsmt_SF  0.6350220
24                      First_Flr_SF  0.6166463
13                         Bsmt_Qual  0.6007528
39                     Garage_Finish  0.5598839
30                         Full_Bath  0.5560557
8                         Year_Built  0.5332261
38                      Fireplace_Qu  0.5298941
(...)
```

VIMP measures (cont')

- Interestingly, we see that Overall_Qual also has the largest correlation with Sale_Price.
- In fact, we see some coherence between these two measures in this example.
- But we must keep in mind that the correlation is a bivariate measure while the VIMP is a multivariate measure, that takes into account all the other variables.