Tree-based models

viettel

1. Predictive modeling

Predictive modeling is required for :

- Estimating the function behind the data
- Taking decisions based on science, not arbitrarily

2. Choosing a predictive model

Predictive models approximate a mapping function F that maps input feature $\boldsymbol{X}$ to the target feature Y: $F(\boldsymbol{X}) + \varepsilon$. The form of the true function F being unknown, models make assumptions regarding this functional form and estimate the function.

There are **2 types of models**:

- **Parametric models**: Models that simplify the function F to a known functional form[1]
- **Non-parametric models**: Models that are free to learn any functional form

There are **2 types of predictive tasks**:

- **Regression task**: Task of approximating a mapping function F when the target feature is quantitative
- **Classification task**: Task of approximating a mapping function F when the target feature is qualitative[2]

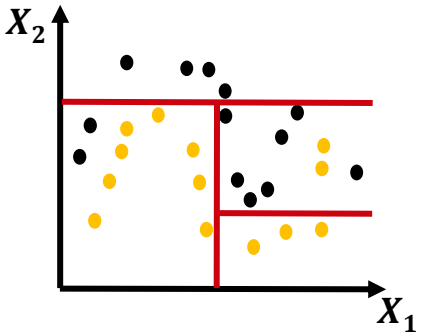
[1] Usually, the assumed functional form is a linear combination of the input features and, as such, those predictive models refer to linear models

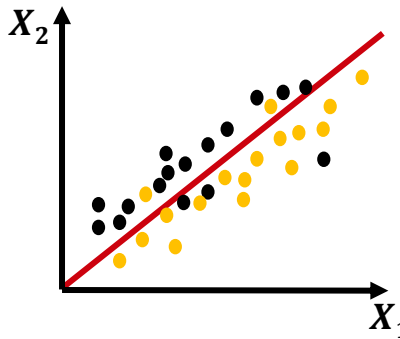
[2] Either binary or multi-class

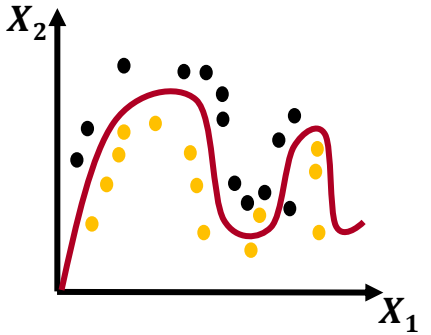
2. Choosing a predictive model

There are 2 types of models:

	Parametric models	Non-parametric models
Benefits	Less data required to estimate the function Fast to learn from data as a fixed number of parameters (small) needs to be estimated Simple to understand Robust to overfitting	Flexible to fit a large number of functional forms Robust as no/few assumptions are made about the function Good fit as many forms are involved
Limitations	Constrained to fit a specific functional form Not robust as strong assumptions are made about the function Poor fit if the function is not linear	More data required to estimate the function Slow to learn from data as the number of parameters grow with the size of data Difficult to understand Prone to overfitting[1]







[1] Non-parametric models estimate the function while getting the closest possible to the data points and are then more prone to overfitting

2. Choosing a predictive model

The **choice of the predictive model** will then depend on:

- The **type of the model**
- The **type of the predictive task**
- The **strengths and weaknesses** of the model

Model	Type	Task
Linear regression	Parametric	Regression
Logistic regression	Parametric	Classification
Poisson regression	Parametric	Regression
Linear discriminant analysis	Parametric	Classification
Penalized regression (Ridge, Lasso, Elastic Net)	Parametric	Both
Decision tree	Non-parametric	Both
Support vector machine	Both	Both
K-nearest neighbors	Non-parametric	Both
Tree-based models	Non-parametric	Both

	Linear/Logistic/ Poisson regression	Penalized regression	Linear SVC	Non-linear SVC	CART tree	Bagging	Random Forests	Boosting
Robustness to overfitting	No penalization	Penalization	- Support vectors only - Penalization	- Support vectors only - Penalization	Instable method[2]	Aggregation	Aggregation	Aggregation
Performance	- Dataset-dependent - Linear relation[1]	- Dataset-dependent - Linear relation[1]	- Margin maximization - Linear relation[1]	Margin maximization	Local optimum	Variance optimization	Variance optimization	Bias optimization
Interpretability	Coefficient analysis	Coefficient analysis	- Support vectors - Black box	- Support vectors - Black box	- Binary decisions - Tree diagram	- Multiple trees - Black box	- Multiple trees - Black box	- Multiple trees - Black box
Implementation	No hyper-parameter tuning	Few hyper-parameters	Few hyper-parameters	Few hyper-parameters	Some hyper-parameters	Many hyper-parameters	Many hyper-parameters	Many hyper-parameters
Computational cost	Simple model	Simple model	Scalar product	Scalar product	Greedy search	Many iterations	Many iterations	Many iterations

[1] GLM models and linear SVC might be under-performing if the relationship with the target feature is not linear

[2] Even though CART tree might be penalized (trimmed), the strategy still relies on a greedy search which makes the method unstable and prone to overfitting

2. Choosing a predictive model

1. CART Decision tree[1]

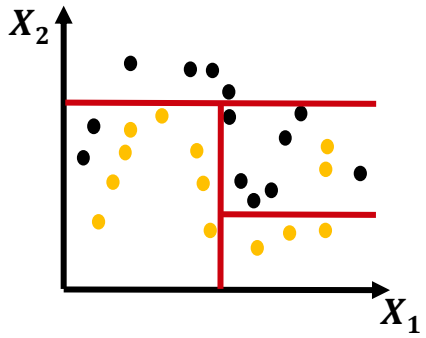
Type	Non-Parametric
Target feature	Qualitative[2]
Distribution	No assumption
Task	Classification[2]
Loss	Misclassification error or Gini index or Cross entropy
Method	Node purity maximization through greedy search

Mapping: No functional form

Loss function at leaf l[3]:

- Misclassification or 0-1 loss function: $Loss(l) = 1 - \max_k(\hat{p}_{lk})$
- Gini index: $Loss(l) = 1 - \sum_{k=1}^{K}(\hat{p}_{lk})^2$
- Cross entropy: $Loss(l) = -\sum_{k=1}^{K}\hat{p}_{lk}\log(\hat{p}_{lk})$

Assumption: None



The idea is to find recursively the input feature and split point (i.e. best binary decision) that minimizes an impurity measure of a sample of records forming a region

[1] Only the CART decision tree is considered here, which is a decision tree that splits a region into 2 sub-regions thanks to binary decisions

[2] Only the classification task, in a binary set-up, is considered here but the target feature could be quantitative with a resulting task being regression

[3] The loss at node level refers as the node impurity

2. Choosing a predictive model

1. CART Decision tree

Algorithm:

1. From a dataset D composed of N records
2. For each feature X_j and splitting point S_j from X_j, calculate the purity gain[1] from splitting the region into 2 sub-regions: $\mathbf{gain} = loss_{R_{root}} - \frac{\sum_{i=1}^{N} \mathbb{I}(X_i \in R_{left})}{\sum_{i=1}^{N} \mathbb{I}(X_i \in R_{root})} loss_{R_{left}} - \frac{\sum_{i=1}^{N} \mathbb{I}(X_i \in R_{right})}{\sum_{i=1}^{N} \mathbb{I}(X_i \in R_{root})} loss_{R_{right}}$
3. Choose (X_j, S_j) in order to maximize the purity gain
4. Split the region accordingly to (X_j, S_j)
5. Repeat steps 2-4 until a stopping criterion[2] is met
6. Prune tree[3]: $\mathbf{arg\,min}_{\alpha}\left[\sum_{l=1}^{L}\left[\sum_{i=1}^{N} \mathbb{I}(X_i \in R_l)\right] loss_{R_l}\right] + \alpha L$

[1] Which corresponds to the impurity decrease from splitting the region

[2] Usually, a stopping criteria is intended to prevent overfitting (number of records in a leaf, depth of a tree,...) but is different from pruning

[3] The higher the m, the smaller the tree

2. Choosing a predictive model

Type	Non-Parametric
Target feature	Qualitative[1]
Distribution	No assumption
Task	Classification[1]
Loss	Misclassification error or Gini index or Cross entropy
Method	Variance reduction through simultaneous creation of different experts (bootstrap sampling) and aggregation

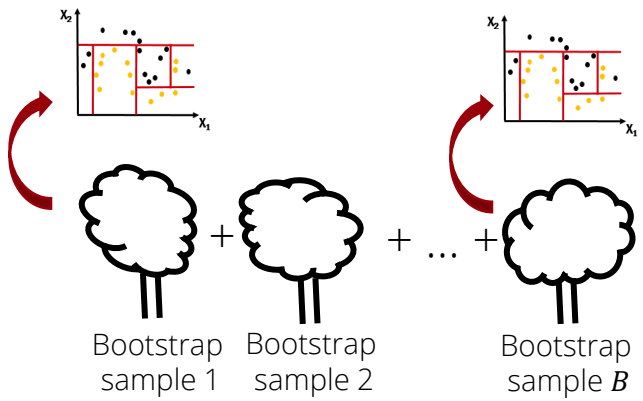
2. Bootstrap Aggregating or Bagging

Mapping: No functional form

Loss function at leaf $\boldsymbol{l}$:

- Misclassification or 0-1 loss function: $Loss(l) = 1 - \max_k(\hat{p}_{lk})$
- Gini index: $Loss(l) = 1 - \sum_{k=1}^{K}(\hat{p}_{lk})^2$
- Cross entropy: $Loss(l) = -\sum_{k=1}^{K}\hat{p}_{lk}\log(\hat{p}_{lk})$

Assumption: None



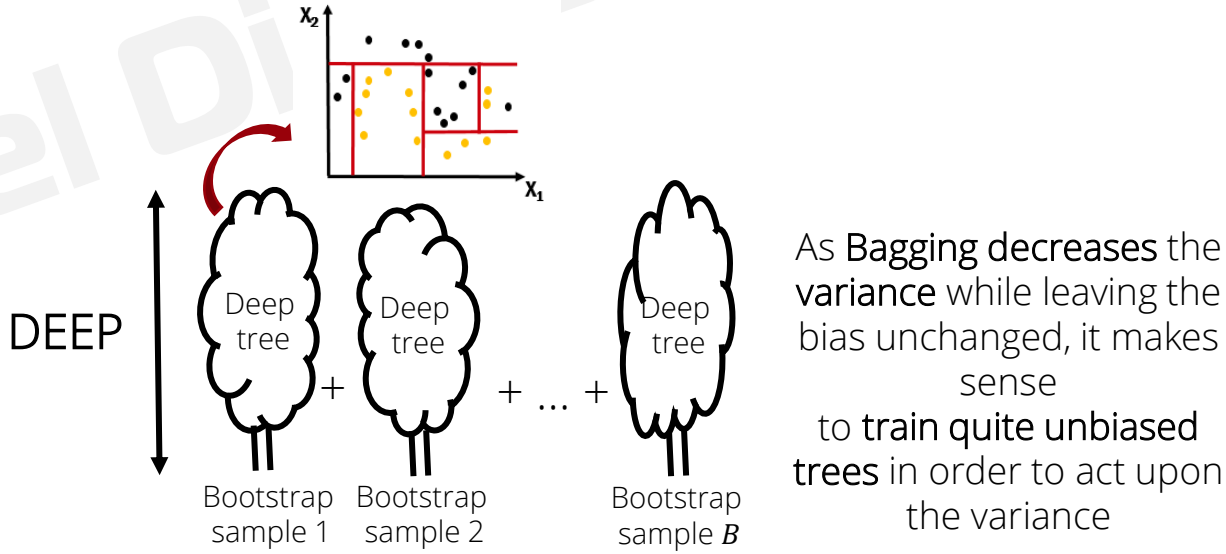
The idea is to generate simultaneously B bootstrap samples[2] from the dataset, train their own B CART trees and aggregating their votes through a majority voting system

[1] Again, only the classification task, in a binary set-up, is considered

[2] See Annex 3 about Bootstrapping for further explanation

2. Choosing a predictive model

2. Bootstrap Aggregating or Bagging
   - **In case of the squared loss (regression task):** it decreases the test error of the aggregated model whatever the tree's structure
   - **In case of the 0-1 loss (classification task):** it decreases the test error of the aggregated model only if the trees are rather unbiased[1]



As **Bagging decreases** the **variance** while leaving the bias unchanged, it makes sense to **train quite unbiased trees** in order to act upon the variance

[1] Trees are then usually grown without trimming, at least deep enough

2. Choosing a predictive model

2. Bootstrap Aggregating or Bagging

Algorithm :

1. From a dataset D composed of N records
2. For each iteration b ($b = 1, \dots, B$):
   I. Create a bootstrap sample D_b by randomly drawing N records with replacement from D
   II. Train a model $\hat{F}_b$ from D_b
3. Let each of the B models classify the records and assign the class k based on a majority voting:

$$\hat{F}(X) = \arg\min_k \sum_{b=1}^{B} \mathbb{I}\left(\hat{F}_b(X) \neq k\right)$$

2. Choosing a predictive model

Type	Non-Parametric
Target feature	Qualitative[1]
Distribution	No assumption
Task	Classification[1]
Loss	Misclassification error or Gini index or Cross entropy
Method	Variance reduction[2] through bagging and random subset of features

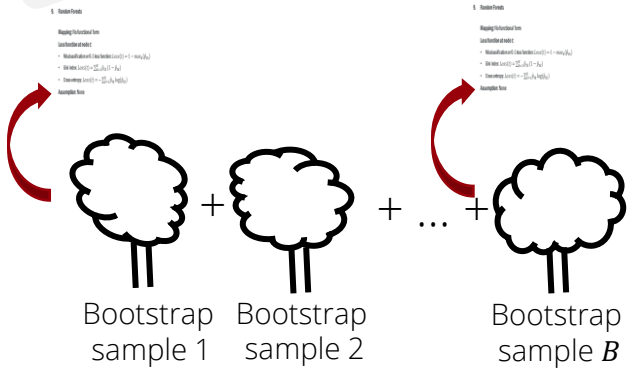
3. Random Forests

Mapping: No functional form

Loss function at leaf $\boldsymbol{l}$:

- Misclassification or 0-1 loss function: $Loss(l) = 1 - \max_k(\hat{p}_{lk})$
- Gini index: $Loss(l) = 1 - \sum_{k=1}^{K}(\hat{p}_{lk})^2$
- Cross entropy: $Loss(l) = -\sum_{k=1}^{K}\hat{p}_{lk}\log(\hat{p}_{lk})$

Assumption: None



The idea is similar to Bagging[3] except that an input feature is chosen for binary decision from a random sample of v features without replacement

[1] Again, only the classification task, in a binary set-up, is considered

[2] Variance decreases even more thanks to extra tree decorrelation (random sampling of records + random subset of features)

[3] See supra for explanation about Bagging

2. Choosing a predictive model

3. Random Forests

Algorithm :

1. From a dataset D composed of N records
2. For each iteration b ($b = 1, \dots, B$):
   I. Create a bootstrap sample D_b by randomly drawing N records with replacement from D
   II. Train a model $\hat{F}_b$ from D_b while:
      A. Choosing randomly v features without replacement from the p features
      B. Choosing X_j from v and S_j in order to maximize the gain and splitting the region accordingly
3. Let each of the B models classify the records and assign the class k based on a majority voting: $\hat{F}(X) = \arg\min_k \sum_{b=1}^{B} \mathbb{I}\left(\hat{F}_b(X) \neq k\right)$

2. Choosing a predictive model

4. Adaptive Boosting or AdaBoost

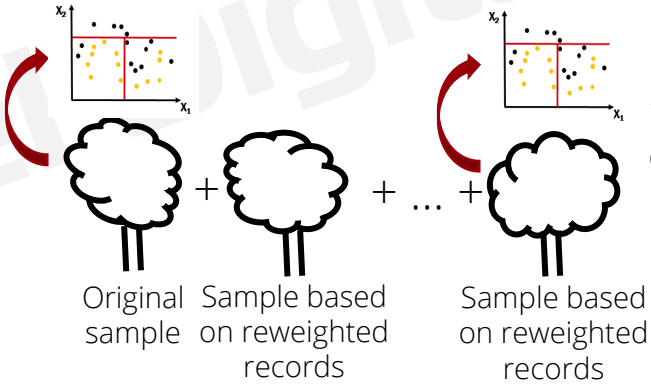
Type	Non-Parametric
Target feature	Qualitative[1]
Distribution	No assumption
Task	Classification[1-2]
Loss	Exponential loss
Method[3]	Bias reduction (local margin maximization) through sequential creation of corrective models and weighted aggregation

Mapping: No functional form

Loss function:

- Exponential loss: $Loss = e^{-Y\hat{Y}}$

Assumption: None



The idea is to improve a weak tree[4] over a sequence of B iterations of resampled data[5] and aggregating their votes through a weighted majority voting system

[1] Again, only the classification task, in a binary set-up, is considered

[2] Adaboost assumes the recoding of the binary target to $\{-1, +1\}$, which is important for the concept of "margin"

[3] While SVC maximizes the minimum margin for all records at once, Adaboost maximizes the margin locally for each record

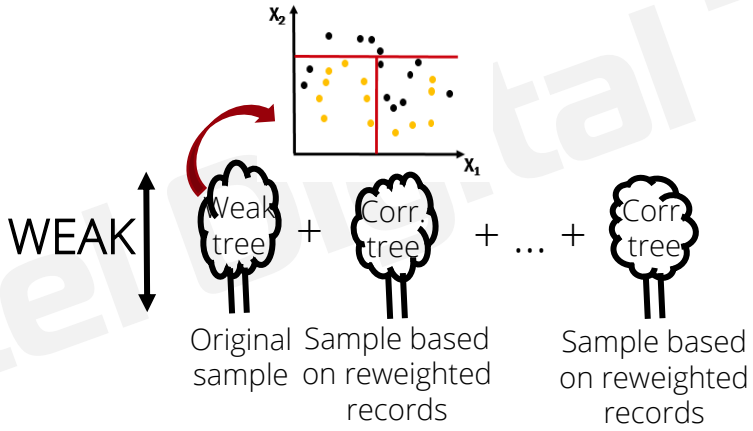
[4] A weak tree is a tree that performs slightly better than a random choice in a binary set-up ($\varepsilon \leq 50\%$)

[5] Last iteration's erroneous predictions get their weights increase and get more chance to get resampled at the next iteration in order to be further modeled for correction

2. Choosing a predictive model

4. Adaptive Boosting or AdaBoost

It decreases the test error of the aggregated model only if the trees are rather biased[1]



As **Adaboost decreases** the **bias** mostly even tough variance might decrease through aggregation, it makes sense to **train quite biased trees** in order to act upon the bias

[1] Trees are then usually grown simple (depth of 2 to 6 levels)

2. Choosing a predictive model

4. Adaptive Boosting or AdaBoost

Weights:

- First iteration: $w_{1,i} = 1/N$ → Records are given equal weight (uniform weighting)
- Next iterations: $w_{b+1,i} = w_{b,i} \exp\left(c_b \mathbb{I}\left(\hat{F}_b(X_i) \neq Y_i\right)\right)$ → Record's weights are updated with a constant[1]

Constants[2-3], which are functions of the test error:

- Breiman's constant: $c_b = \ln\left(\frac{1-\varepsilon_b}{\varepsilon_b}\right)$
- Freund & Schapire's constant: $c_b = \frac{1}{2}\ln\left(\frac{1-\varepsilon_b}{\varepsilon_b}\right)$

Margin[4]: $m(X_i) = \frac{\sum_{b=1}^{B} c_b \mathbb{I}(\hat{F}_b(X_i)=Y_i)}{\sum_{b=1}^{B} c_b} - \frac{\sum_{b=1}^{B} c_b \mathbb{I}(\hat{F}_b(X_i)\neq Y_i)}{\sum_{b=1}^{B} c_b}$

[1] Records with erroneous predictions will get their weights updated upwards, while others will get their weights unchanged

[2] The lower ε_b, the higher c_b. Given weak trees ($\varepsilon \leq 50\%$), the constants will be positive

[3] Constants, which are functions of the error, are used to weight the vote of each tree in the ensemble

[4] Correctly predicted records have positive margin while incorrectly predicted records have negative margin (link with SVC and Adaboost)

2. Choosing a predictive model

4. Adaptive Boosting or AdaBoost

Algorithm :

1. From a dataset D composed of N records
2. Set initial weights $w_{1,i} = 1/N \quad \forall i$
3. For each iteration b $(b = 1, \dots, B)$:
   I. Create a sample D_b by randomly drawing N records with replacement from D based on their weights
   II. Train a model $\hat{F}_b$ from D_b
   III. Calculate the error $\varepsilon_b = \sum_{i=1}^{N} w_{b,i}\, \mathbb{I}\left(\hat{F}_b(X_i) \neq Y_i\right)$ and constant $c_b = \ln\left(\frac{1-\varepsilon_b}{\varepsilon_b}\right)$ or $\frac{1}{2}\ln\left(\frac{1-\varepsilon_b}{\varepsilon_b}\right)$
   IV. If $\varepsilon_b > 50\%$, the model $\hat{F}_b$ is not weak, reinitialize weights to $1/N$ and get to 3.1.
   V. Update the weights $w_{b+1,i} = w_{b,i} \exp\left(c_b \mathbb{I}\left(\hat{F}_b(X_i) \neq Y_i\right)\right)$ and normalize them
4. Let each of the B models classify the records and assign the class k based on a weighted majority voting: $\hat{F}(X) = \arg\min_k \sum_{b=1}^{B} c_b \mathbb{I}\left(\hat{F}_b(X) \neq k\right)$

2. Choosing a predictive model

5. Gradient Boosting

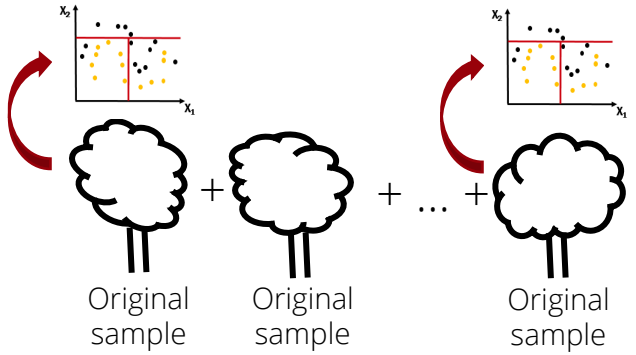
Type	Non-Parametric
Target feature	Qualitative[1]
Distribution	No assumption
Task	Classification[1-2]
Loss	Log-loss or logistic loss or exponential loss
Method	Bias reduction through sequential creation of corrective models and aggregation[5]

Mapping: No functional form

Loss function:

- Log-loss: $Loss = -Y \ln(\hat{Y}) - (1 - Y) \ln(1 - \hat{Y})$
- Logistic loss: $Loss = \ln(1 + e^{-Y\hat{Y}})$
- Exponential loss: $Loss = e^{-Y\hat{Y}}$

Assumption: None



The idea is to improve a weak tree[3] over a sequence of B iterations while focusing on the residuals[4] and aggregating[5] predictions

[1] Again, only the classification task, in a binary set-up, is considered

[2] Gradient Boosting assumes the binary target within the space $\{0, +1\}$

[3] Again, weak trees are used, due to the boosting process

[4] As we boost the residuals, which are obtained from the first derivative of the loss function, we boost the gradients

[5] Aggregation with Gradient Boosting is not weighted (i.e. each tree has the same weight) since no resampling of data is involved, unlike in Adaboost

2. Choosing a predictive model

5. Gradient Boosting

Algorithm :

1. From a dataset D composed of N records
2. Initialize a first model with a constant value[1-2]: $\hat{F}_0(X) = \arg\min_\gamma \sum_{i=1}^{N} L(Y_i, \gamma)$
3. For each iteration b $(b = 1, \dots, B)$:
   I. Calculate the pseudo residuals[3-4] $r_{ib} = -\left[\frac{\partial L(Y_i, \hat{F}_{b-1}(X_i))}{\partial \hat{F}_{b-1}(X_i)}\right] \; i = 1, \dots, N$
   II. Train a regression tree to the pseudo residuals r_{ib} to minimize $\sum_{i=1}^{N} L(r_{ib}, \hat{r}_{ib})$ and create the leaves $R_{jb}, j = 1, \dots, J_b$

Given the loss is the log-loss:

[1] $L(Y, \hat{Y}) = -Y \ln(\hat{p}) - (1 - Y) \ln(1 - \hat{p})$ in terms of probability and $L(Y, \hat{Y}) = -Y \ln(\hat{p}/1 - \hat{p}) + \ln(1 + e^{\ln(\hat{p}/1-\hat{p})})$ in terms of odds

[2] γ corresponds to the log-odds: $\gamma = \ln(\hat{p}/1 - \hat{p})$

[3] First derivative regards to the log-odds $\frac{\partial L(Y_i, \hat{F}_{b-1}(X_i))}{\partial \hat{F}_{b-1}(X_i)} = g_{ib} = -(Y_i - \hat{p}_{ib})$ and refers to gradient, second derivative $\frac{\partial^2 L(Y_i, \hat{F}_{b-1}(X_i))}{\partial^2 \hat{F}_{b-1}(X_i)} = h_{ib} = \hat{p}_{ib}(1 - \hat{p}_{ib})$ and refers to the hessian

[4] Pseudo residuals can be expressed as: $r_{ib} = -g_{ib} = Y_i - \hat{p}_{ib}$ in terms of probability or $r_{ib} = Y_i - \frac{e^{\ln(\hat{p}_{ib}/1-\hat{p}_{ib})}}{1+e^{\ln(\hat{p}_{ib}/1-\hat{p}_{ib})}}$ in terms of odds, with $\hat{p}_{ib}$ relating to $b - 1$

2. Choosing a predictive model

5. Gradient Boosting

Algorithm :

III. Calculate the leaf output[1-2] $\gamma_{jb} = \arg\min_{\gamma} \sum_{X_i \in R_{jb}} L\left(Y_i, \hat{F}_{b-1}(X_i) + \gamma\right)$, for each leaf $j = 1, \dots, J_b$

IV. Update the predictions[3-4] $\hat{F}_b(X) = \hat{F}_{b-1}(X) + \varphi \sum_{j=1}^{J_b} \gamma_{jb} \mathbb{I}\left(X \in R_{jb}\right)$

4. Let the last model predict the records[5]: $\hat{F}(X) = \hat{F}_B(X)$

[1] One leaf may indeed have several records and thus several residuals, so we need to aggregate the results leaf-wise

[2] Given the loss is the log-loss, leaf output corresponds to: $\gamma_{jb} = \frac{-\sum_{X_i \in R_{jb}} g_{ib}}{\sum_{X_i \in R_{jb}} h_{ib}} = \frac{-\sum_{X_i \in R_{jb}} g_{ib}}{cover} = \frac{\sum_{X_i \in R_{jb}} r_{ib}}{\sum_{X_i \in R_{jb}} [\hat{p}_{ib}(1-\hat{p}_{ib})]}$ with $\hat{p}_{ib}$ relating to $b-1$

[3] φ is the learning rate

[4] The binary function refers to whether a record ends up in multiple leaves

[5] The final output is nothing more than the sum over all previous models' updated predictions (simple aggregation): since the first prediction is a log-odds prediction, the final prediction will also be and can be turned to a probability prediction with $p = \frac{e^{\ln(p/1-p)}}{1+e^{\ln(p/1-p)}}$

2. Choosing a predictive model

6. Extreme Gradient Boosting or XGBoost

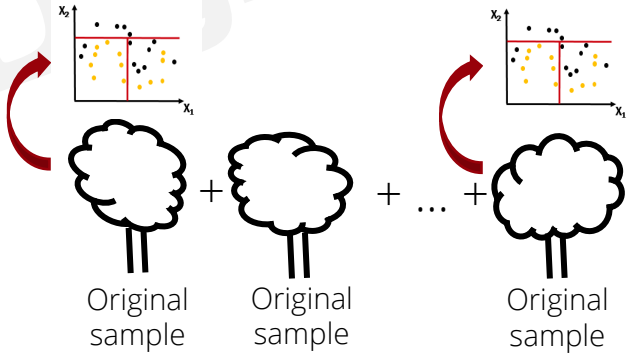
Mapping: No functional form

Loss function:

- Log-loss: $Loss = -Y \ln(\hat{Y}) - (1 - Y) \ln(1 - \hat{Y})$

Assumption: None

Type	Non-Parametric
Target feature	Qualitative[1]
Distribution	No assumption
Task	Classification[1-2]
Loss	Log-loss
Method	Bias reduction through sequential creation of corrective models and aggregation



The idea is similar to Gradient Boosting except that the weak trees[3] are regularized[4]

[1] Again, only the classification task, in a binary set-up, is considered

[2] Extreme Gradient Boosting assumes the binary target within the space $\{0, +1\}$

[3] Again, weak trees are used, due to the boosting process

[4] The regularization is similar to Ridge Regression

2. Choosing a predictive model

6. Extreme Gradient Boosting or XGBoost

Algorithm :

1. From a dataset D composed of N records
2. Initialize a first model with a constant value[1]: $\hat{F}_0(X) = 0{,}5$
3. For each iteration b $(b = 1, \dots, B)$:
   I. Calculate the residuals[2] $r_{ib} = -g_{ib} = Y_i - \hat{p}_{ib} \quad i = 1, \dots, N$
   II. Train a regression tree to the residuals r_{ib} to minimize[3] $\sum_{i=1}^{N} L(r_{ib}, \hat{r}_{ib}) + \frac{1}{2}\lambda\gamma_b^2 + \psi T_b$ and create the leaves $R_{jb}, j = 1, \dots, J_b$:
      A. Calculate the leaf output[2-4] $\gamma_{jb} = \frac{-\sum_{X_i \in R_{jb}} g_{ib}}{\left[\sum_{X_i \in R_{jb}} h_{ib}\right] + \lambda} = \frac{\sum_{X_i \in R_{jb}} r_{ib}}{\left[\sum_{X_i \in R_{jb}} \hat{p}_{ib}(1-\hat{p}_{ib})\right] + \lambda}$, for each leaf $j = 1, \dots, J_b$

[1] XGBoost chooses 50% as initial model

[2] Expressed in probability, given the loss is the log-loss, with $\hat{p}_{ib}$ relating to $b - 1$

[3] λ is the shrinkage parameter, γ is the leaf output, ψ is the tree pruning penalty, T_b is the number of leaves in the tree

[4] $\sum_{X_i \in R_{jb}} h_{ib}$ is the cover and should be set with 0 in the classification task (~min_child_weight)

2. Choosing a predictive model

6. Extreme Gradient Boosting or XGBoost

Algorithm :

B. Calculating the similarity score[1-2] $score_{jb} = \frac{\left(\sum_{X_i \in R_{jb}} r_{ib}\right)^2}{\left[\sum_{X_i \in R_{jb}} \hat{p}_{ib}(1-\hat{p}_{ib})\right]+\lambda}$, for each leaf $j = 1, \dots, J_b$

C. Calculating the split gain $\mathbf{gain} = score_{R_{left}} + score_{R_{right}} - score_{R_{root}}$

D. Choose (X_k, S_k) in order to maximize the gain

E. Prune branches if[3] $gain - \psi < 0$ and get the final tree

III. Update the predictions[4-5] $\hat{F}_b(X) = \hat{F}_{b-1}(X) + \varphi \sum_j^{J_b} \gamma_{jb} \mathbb{I}(X \in R_{jb})$

4. Let the last model predict the records[6]: $\hat{F}(X) = \hat{F}_B(X)$

[1] Expressed in probability, given the loss is the log-loss, with $\hat{p}_{ib}$ relating to $b-1$

[2] The higher the λ (which appears with one residual in the leaf or if residuals are similar), the lower the similarity score and the lower the gain, the more likely the tree to be pruned

[3] The higher the pruning penalty ψ, the more likely the tree to be pruned (warning: $\psi = 0$ doesn't turn off pruning if λ is involved)

[4] φ is again the learning rate

[5] Since the first prediction is a probability ($\hat{F}_0(X) = 0{,}5$), we need to convert this probability to log-odds before updating: $\ln(p/1-p)$

[6] The final prediction, which is again the sum over all previous models' updated predictions (simple aggregation), will be a log-odds prediction and can be then turned to a probability prediction with $p = \frac{e^{\ln(p/1-p)}}{1+e^{\ln(p/1-p)}}$

viettel

2. Choosing a predictive model

Recap: Key differences about tree-based ensemble

Dependence

- Bagging: Independent trees
- Boosting: Sequential trees (~dependence)

Bagging

- Bagging: Randomization through record sampling
- Random Forests: Randomization through record sampling and feature sampling

Boosting

- Adaboost: Boosting based on sampling of erroneous records, weighted aggregation
- Gradient Boosting: Boosting based on shrinking pseudo residuals, aggregation
- Extreme Gradient Boosting: Gradient Boosting with regularization

3. Model's hyper-parameters

1. CART Decision tree

Hyper-parameter	Space	Explanation
Loss	[misclassification, gini, entropy]	Loss function used for calculating the loss over all records
Depth	Positive integer	Maximum depth of the tree: the deeper the tree, the more overfitting
Split samples	Positive integer or float	Minimum number of records for splitting a node: the larger the number, the smaller the tree and the less overfitting
Leave samples	Positive integer or float	Minimum number of records to be part a leaf: the larger the number, the smaller the tree and the less overfitting
Leave number	Positive integer	Maximum number of leaves: the smaller the number, the smaller the tree and the less overfitting
Impurity decrease	Positive float	Minimum impurity decrease for splitting a node: the larger the decrease requested, the smaller the tree and the less overfitting
α[1]	Positive float	Amount of penalization: the larger the amount, the smaller the tree and the less overfitting

[1] CART decision tree also exists with a penalized form which accounts for the size of tree

3. Model's hyper-parameters

2. Bootstrap Aggregating or Bagging

Hyper-parameter	Space	Explanation
All CART tree's hyper-parameters are valid for Bagging		
Tree number	Positive integer	Number of trees: the more trees, the larger the reduction in variance and the less overfitting[1]

[1] In practice, however, it is not always the case so it is advised to perform hyper-parameter tuning

3. Model's hyper-parameters

3. Random Forests

Hyper-parameter	Space	Explanation
All CART tree's hyper-parameters are valid for Random Forests		
Tree number	Positive integer	Number of trees: the more trees, the larger the reduction in variance and the less overfitting[1]
Feature number	Positive integer or float	Maximum number of features for splitting a node: the smaller the number, the more tree decorrelation

[1] In practice, however, it is not always the case so it is advised to perform hyper-parameter tuning

3. Model's hyper-parameters

4. Adaptive Boosting or AdaBoost

Hyper-parameter	Space	Explanation
All CART tree's hyper-parameters are valid for Adaboost		
Loss[1]	[exponential]	Loss function used for calculating the loss over all records
Tree number	Positive integer	Number of trees: the more trees, the larger the reduction in bias but the more overfitting[2]

[1] All CART decision tree's hyper-parameters are valid for Adaboost except for the loss

[2] As the number of trees increases, the algorithm keeps trying to maximize the margin locally and continuously correct the errors which eventually leads to overfitting

3. Model's hyper-parameters

5. Gradient Boosting

Hyper-parameter	Space	Explanation
All CART tree's hyper-parameters are valid for Gradient Boosting		
Loss[1]	[exponential, logistic, log]	Loss function used for calculating the loss over all records
Tree number	Positive integer	Number of trees: the more trees, the larger the reduction in bias but the more overfitting[2]
Learning rate	Positive float	Step size of the coefficients at each iteration while moving to the minimum of the loss function: the lower the rate, the slower the convergence but the more chance to converge to the global optimum

[1] All CART decision tree's hyper-parameters are valid for Gradient Boosting except for the loss

[2] As the number of trees increases, the algorithm keeps trying to boost the errors which eventually leads to overfitting

3. Model's hyper-parameters

6. Extreme Gradient Boosting or XGBoost

Hyper-parameter	Space	Explanation
All CART tree's hyper-parameters are valid for Extreme Gradient Boosting		
Loss[1]	[log]	Loss function used for calculating the loss over all records
Tree number	Positive integer	Number of trees: the more trees, the larger the reduction in bias but the more overfitting[2]
Learning rate	Positive float	Step size of the coefficients at each iteration while moving to the minimum of the loss function: the lower the rate, the slower the convergence but the more chance to converge to the global optimum
l1/l2 penalty	[0, +∞[	Amount of penalization: the larger the amount, the more the model is regularized, the less overfitting

[1] All CART decision tree's hyper-parameters are valid for XGBoost except for the loss

[2] As the number of trees increases, the algorithm keeps trying to boost the errors which eventually leads to overfitting

Annexes

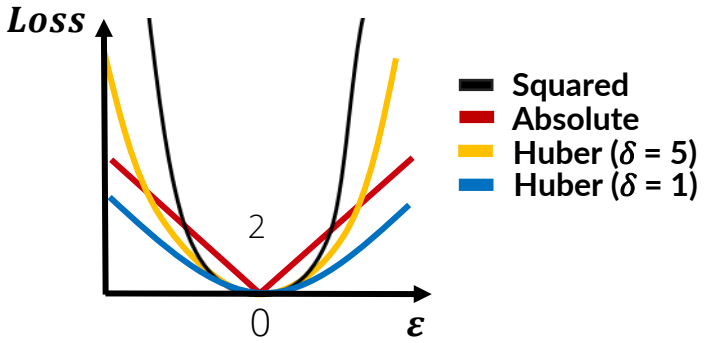
Annex 1 – Loss functions

Loss functions differ depending on the nature of the predictive task.

Regression task

Loss functions include:

- Squared loss function: $Loss = (Y - \hat{Y})^2$
- Absolute loss function: $Loss = |Y - \hat{Y}|$
- Huber loss function[1]: $Loss = \begin{cases} \frac{1}{2}(Y - \hat{Y})^2 & if |Y - \hat{Y}| \leq \delta \\ \delta|Y - \hat{Y}| - \frac{1}{2}\delta^2 & otherwise \end{cases}$



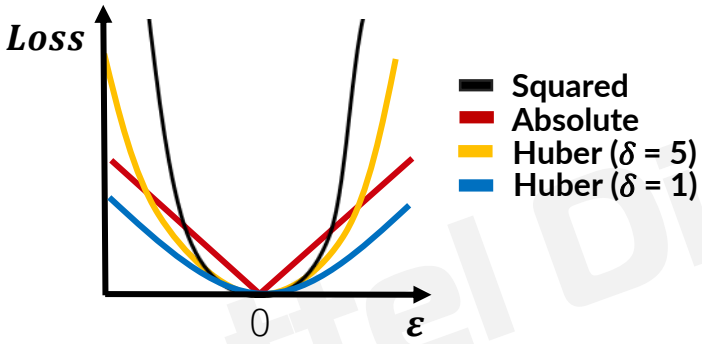
[1] δ is the tuning parameter used to determine which data point is considered as an outlier

[2] Derivability and convexity are nice properties for a function to have because respectively that's easier at optimizing and the minimum found is the global one

Annex 1 – Loss functions

Loss functions differ depending on the nature of the predictive task.

Regression task



Loss function	Optimization[1-2-3]	Comments
Squared loss	Continuous, differentiable and convex	Penalizing more heavily large errors and thus more sensitive to outliers
Absolute loss	Continuous but not differentiable at x=0, convex	Penalizing less heavily large errors and thus more robust to outliers
Huber loss	Continuous, differentiable and convex	Combination of squared loss (when δ = +∞) and absolute loss (when δ = 0) to determine the level of sensitiveness to outliers

[1] A function is continuous if we can ensure arbitrarily small changes by triggering small changes in the input

[2] A real-valued function is differentiable if it is differentiable at every point in its domain – A function that is differentiable at x is by default continuous at x but the contrary doesn't hold true

[3] A real-value function is convex if the line segment between any 2 points lies above the graph between these 2 points

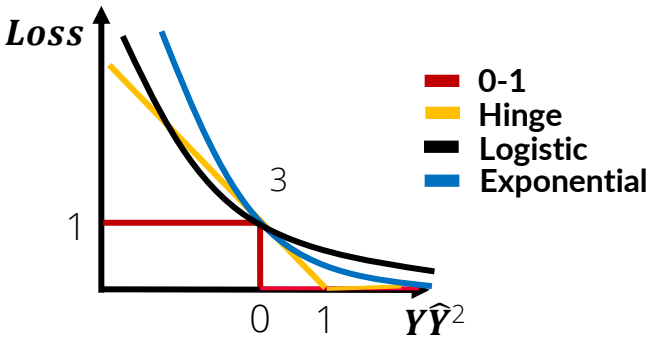
Annex 1 – Loss functions

Loss functions differ depending on the nature of the predictive task.

Classification task[1]

Loss functions include:

- Misclassification or 0-1 loss function1: $Loss = \mathbb{I}(Y \neq \hat{Y})$ $\hat{Y} \rightarrow class$
- Hinge loss function[2]: $Loss = \max(0; 1 - Y\hat{Y})$ $\hat{Y} \rightarrow \boldsymbol{X}\hat{\beta}(< or > 0)$
- Squared Hinge loss function[2]: $Loss = (\max(0; 1 - Y\hat{Y}))^2$ $\hat{Y} \rightarrow \boldsymbol{X}\hat{\beta}(< or > 0)$
- Log-loss function: $Loss = -Y \ln(\hat{Y}) - (1 - Y)\ln(1 - \hat{Y})$ $\hat{Y} \rightarrow \hat{p}$
- Logistic or Binomial Deviance loss function: $Loss = \ln(1 + e^{-Y\hat{Y}})$ $\hat{Y} \rightarrow \hat{p}$
- Exponential loss function (Adaboost)[2]: $Loss = e^{-Y\hat{Y}}$ $\hat{Y} \rightarrow weighted\ sum\ of\ -1/1(< or > 0)$



[1] Only the binary case is considered here with the target feature coded to $\{-1, +1\}$, which is important for the concept of "margin"

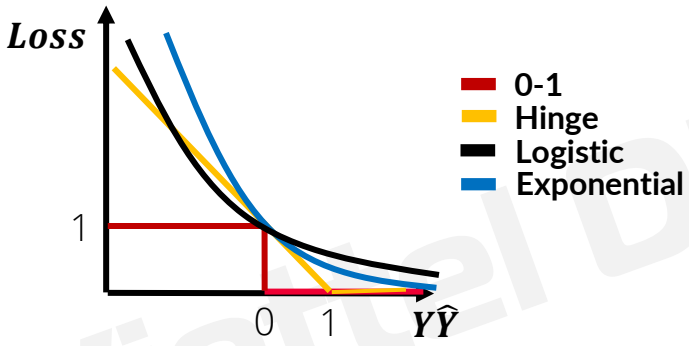
[2] $Y\hat{Y}$ refers to the term "margin": Correctly predicted records have positive margin while incorrectly predicted records have negative margin (link with SVC and Adaboost)

[3] Derivability and convexity are nice properties for a function to have because respectively that's easier at optimizing and the minimum found is the global one

Annex 1 – Loss functions

Loss functions differ depending on the nature of the predictive task.

Classification task



Loss function	Optimization[1-2-3]	Comments
0-1 loss	Not continuous and not differentiable at x=0, not convex	- Giving unit penalty for negative margins and no penalty for positive margins - Robust to outliers
Hinge loss	Continuous but not differentiable at x=1, convex	- Penalizing continuously small margin and increasingly negative margin wile giving no penalty if enough margin
Logistic loss	Continuous, differentiable and convex	- Penalizing continuously decreasingly negative margin more heavily than rewarding increasingly positive margin - Penalizing less heavily negative margin and thus more robust to outliers
Exponential loss	Continuous, differentiable and convex	- Penalizing continuously decreasingly negative margin more heavily than rewarding increasingly positive margin - Penalizing more heavily negative margin and thus more sensitive to outliers

[1] A function is continuous if we can ensure arbitrarily small changes by triggering small changes in the input

[2] A real-valued function is differentiable if it is differentiable at every point in its domain – A function that is differentiable at x is by default continuous at x but the contrary doesn't hold true

[3] A real-value function is convex if the line segment between any 2 points lies above the graph between these 2 points

Annex 2 – Bias and variance trade-off

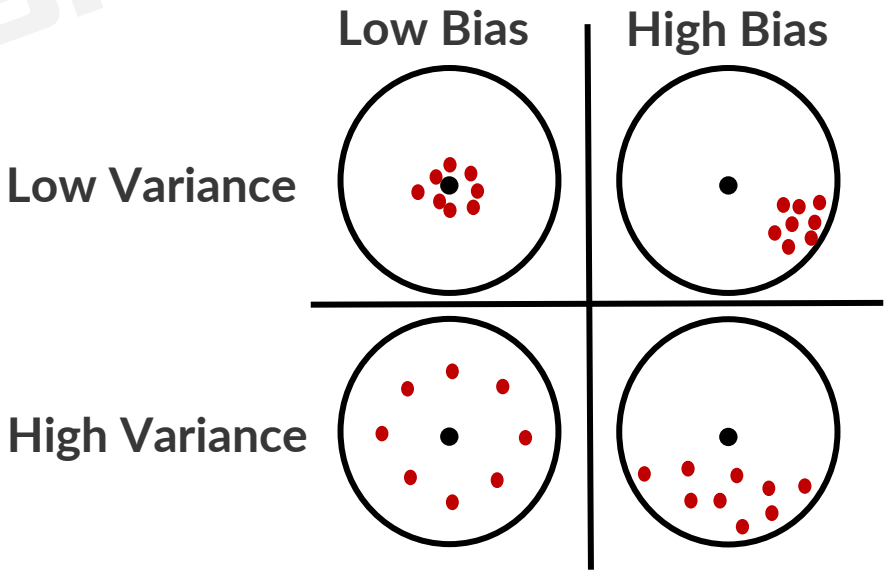
While approximating the true mapping function F with $\hat{F}$, predictive models make errors.

The generalization error in the case of the quadratic loss is[1-2-3]:

$$ERR(\hat{F}) = \mathrm{E}\left[\left[y - \hat{F}(X)\right]^2\right] = Bias(\hat{F})^2 + Var(\hat{F}) + irreducible\ error$$

With:

- $Bias(\hat{F}) = \mathrm{E}\left[\hat{F}(X) - F(X)\right] = \mathrm{E}\left[\hat{F}(X)\right] - F(X)$
- $Var(\hat{F}) = \mathrm{E}\left[\left[\hat{F}(X) - \mathrm{E}\left[\hat{F}(X)\right]\right]^2\right]$



[1] Bias is the error by how much the average prediction differs from the reality: it refers to erroneous assumptions or decisions about the data which affect the representation of results

[2] Variance is the error by how much the prediction fluctuates around the average prediction

[3] Error which cannot be reduced whatever the quality of the model, it is a measure of the amount of noise in the data

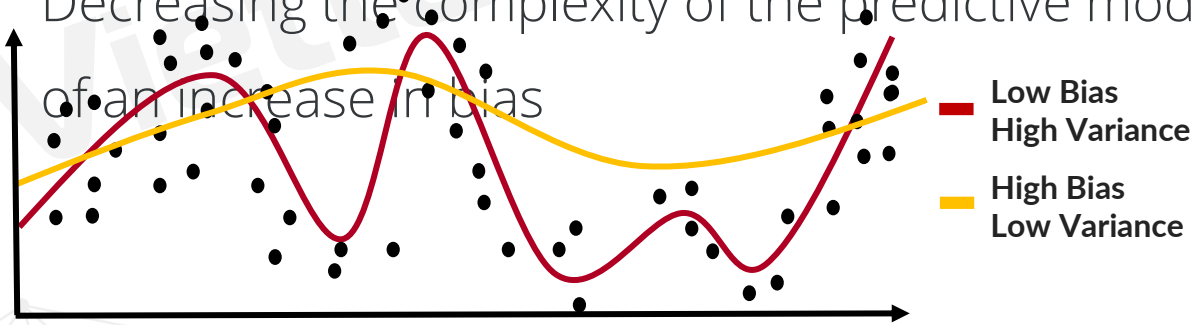
Annex 2 – Bias and variance trade-off

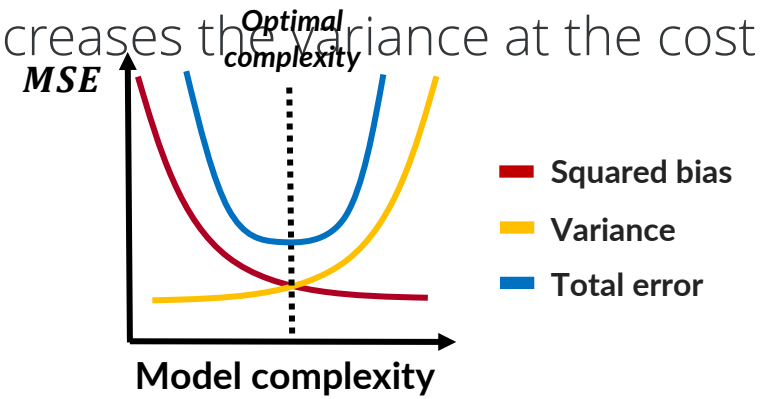
If bias is high, $\hat{F}$ is unable to capture the underlying pattern from the training data **UNDERFITTING**

If variance is high, $\hat{F}$ captures too much the noise from the training data **OVERFITTING**

Bias and variance are 2 quantities that compete against each other and are part of a **trade-off**[1]:

- Increasing the complexity of the predictive model decreases the bias at the cost of an increase in variance
- Decreasing the complexity of the predictive model decreases the variance at the cost of an increase in bias





[1] Model will be chosen and its complexity will be tuned with hyper-parameters in order to minimize the generalization loss

Annex 3 – Bootstrapping

Bootstrapping is a **resampling method used to estimate the sampling distribution**[1] of almost any statistic.

Given θ the unknown parameter of interest, T the related sample's statistic, D a sample of N independent and identically distributed records[2], $T(D)$ is an estimator of θ and depends on D. The procedure consists of :

- **Bootstrap sampling**: Drawing randomly B samples D_b of size N with replacement (~phantom samples) from D
- **Bootstrap's statistic**: Calculating T for each bootstrap sample D_b, $T(D_b)$
- **Bootstrap distribution**: Collecting all the statistic's values across the bootstrap samples and obtaining the bootstrap distribution of T which approximates the sampling distribution of T

[1] The population parameter being unknown, we want to estimate it from a sample's statistic; as any statistic of interest fluctuate across samples, we want to know the magnitude of those fluctuations and, as a result, want to know the sampling distribution of that statistic and its standard deviation (= standard error)

[2] Independent records: records are unrelated to each other – Identically distributed records: records come from the same probability distribution with same parameters