1. Random forests
   - Bagging and split randomization
   - Random forest algorithm
   - Out-of-bag error
   - Variable importance

2. Tree Boosting
   - Motivation
   - General Boosting algorithm
   - Gradient Boosting Decision Trees

Tree ensemble methods

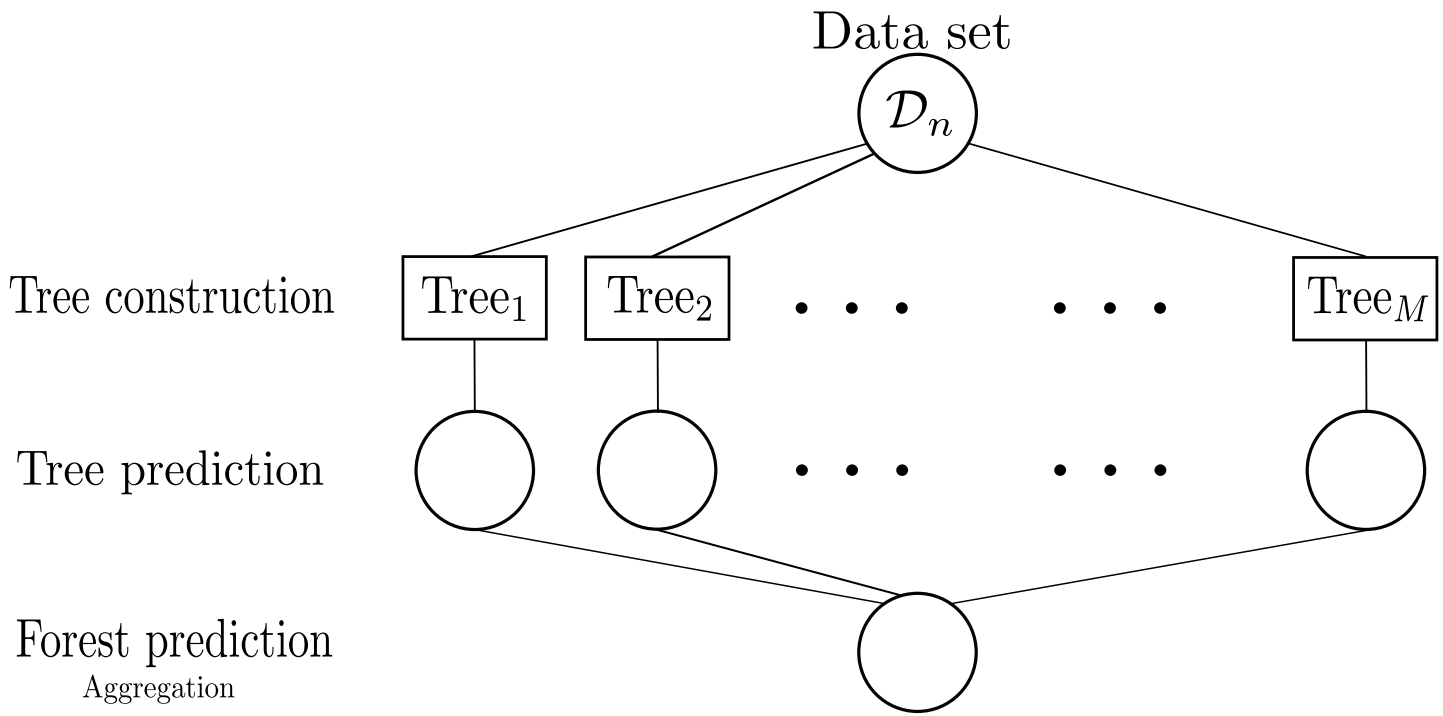
Consist in aggregating the predictions of several decision trees:

- More flexible methods / useful for modelling complex input-output relations
- More robust than individual trees to changes in data

Tree ensemble methods

How to do that?

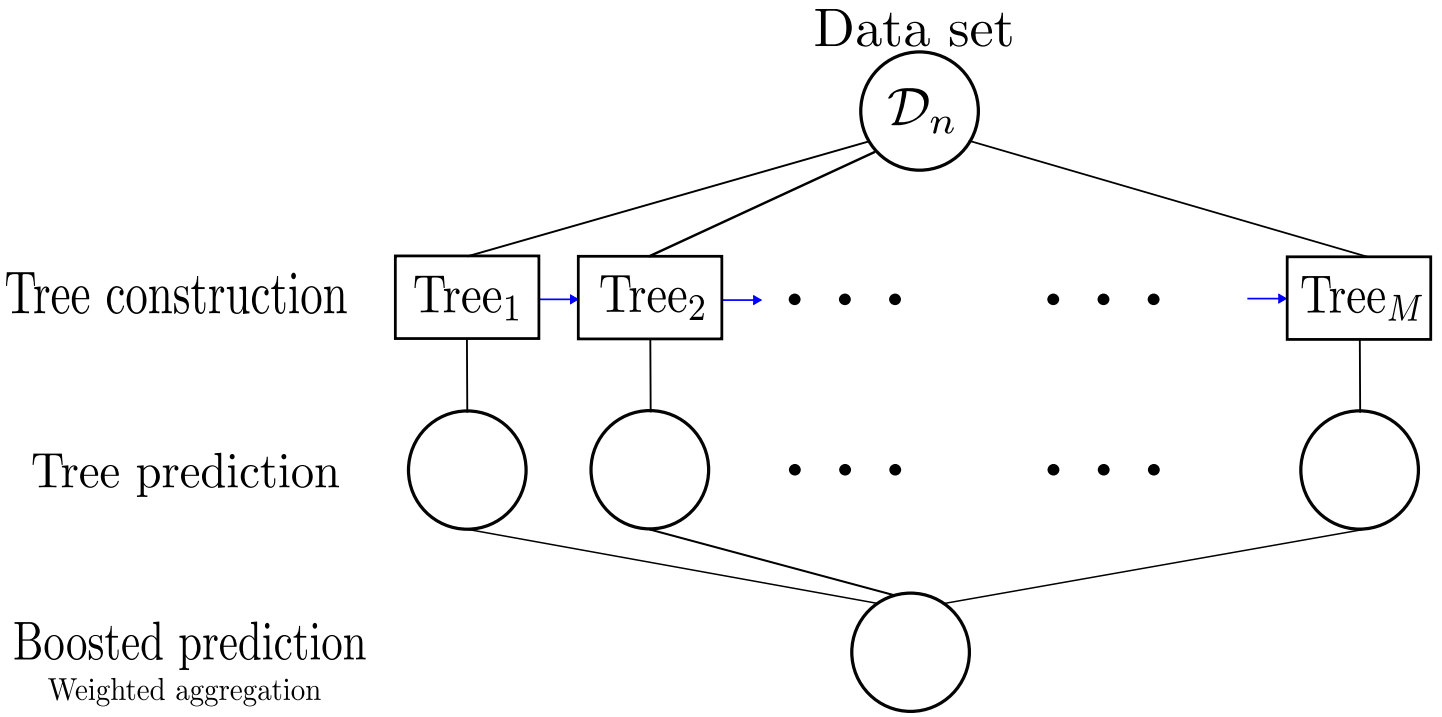
- Random forests (parallel methods)



Tree ensemble methods

How to do that?

- Random forests (parallel methods)
- Tree boosting (sequential methods)



Outline

1. Random forests
   - Bagging and split randomization
   - Random forest algorithm
   - Out-of-bag error
   - Variable importance
2. Tree Boosting
   - Motivation
   - General Boosting algorithm
   - Gradient Boosting Decision Trees

Outline

Tree construction

Tree construction

- Input: a dataset, an impurity measure.
- At each node A, select the best split via
$$(j^\star, s^\star) \in \underset{j \in \{1,\ldots,d\}, s \in \text{range}(X^{(j)})}{\text{argmax}} \Delta Imp(j, s; A).$$
- Repeat for each cell until each leaf contains one observation.
- Output: a fully-grown decision tree.

Tree pruning

- Input: A fully-grown decision tree, a data set, an impurity measure.
- Choose one of the two pruning strategies:
  - Reduction Error pruning (RE, C4.5)
  - Cost complexity pruning (CART)
- Output: a pruned decision tree.

Tree construction in random forests

Tree construction

- Input: a dataset, an impurity measure.
- At each node A, select the best split via
$$(j^\star, s^\star) \in \underset{j \in \{1,\ldots,d\}, s \in \text{range}(X^{(j)})}{\text{argmax}} \Delta Imp(j, s; A).$$
- Repeat for each cell until each leaf contains one observation.
- Output: a fully-grown decision tree.

Tree pruning

For trees in random forests, no pruning strategy

Tree construction

- Input: a dataset, an impurity measure.
- At each node A, select the best split via
$$(j^\star, s^\star) \in \underset{j \in \{1,\dots,d\}, s \in \text{range}(X^{(j)})}{\text{argmax}} \Delta Imp(j, s; A).$$
- Repeat for each cell until each leaf contains one observation.
- Output: a fully-grown decision tree.

Tree pruning

For trees in random forests, no pruning strategy

Such trees have a small bias (fully-grown) but a large variance (one point per leaf).

They cannot be used as single estimators!

Bagging - Averaging predictors via data set resampling

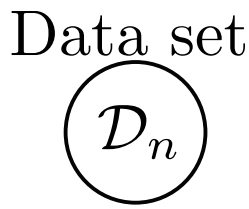
Bagging (**B**ootstrap **agg**regat**ing**) consists in running a learning algorithm on mulitple modified data sets to stabilize its performance.

Bagging - Averaging predictors via data set resampling

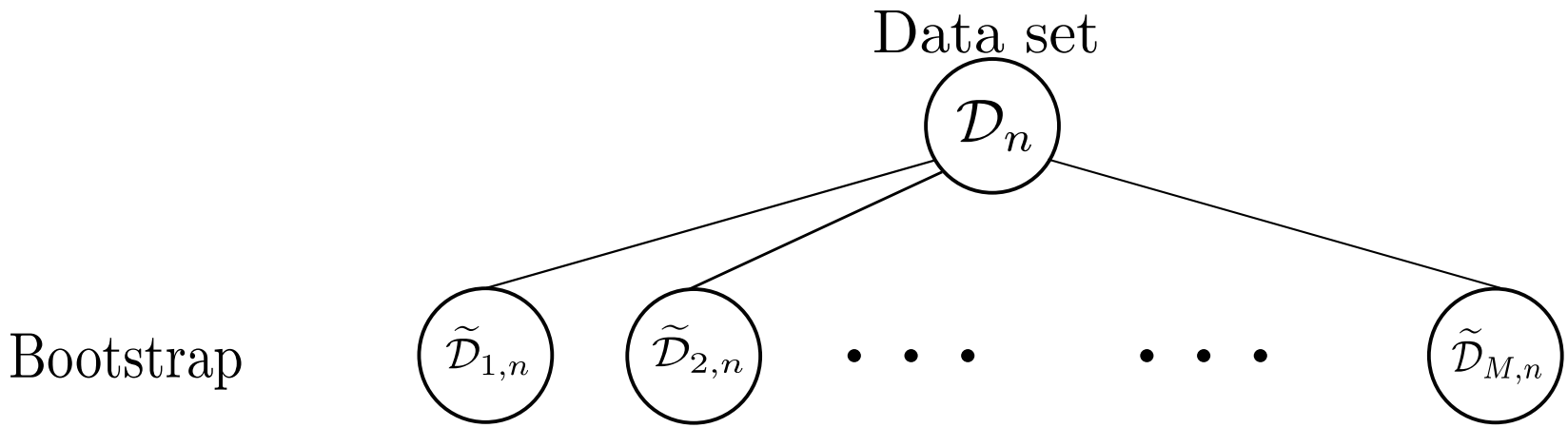
Bagging (**B**ootstrap **agg**regat**ing**) consists in running a learning algorithm on mulitple modified data sets to stabilize its performance.

Bootstrap. A sampling scheme that consists in drawing n observations with replacement among the n original ones. Applied once, bootstrap creates one new data set, called a bootstrapped data set.

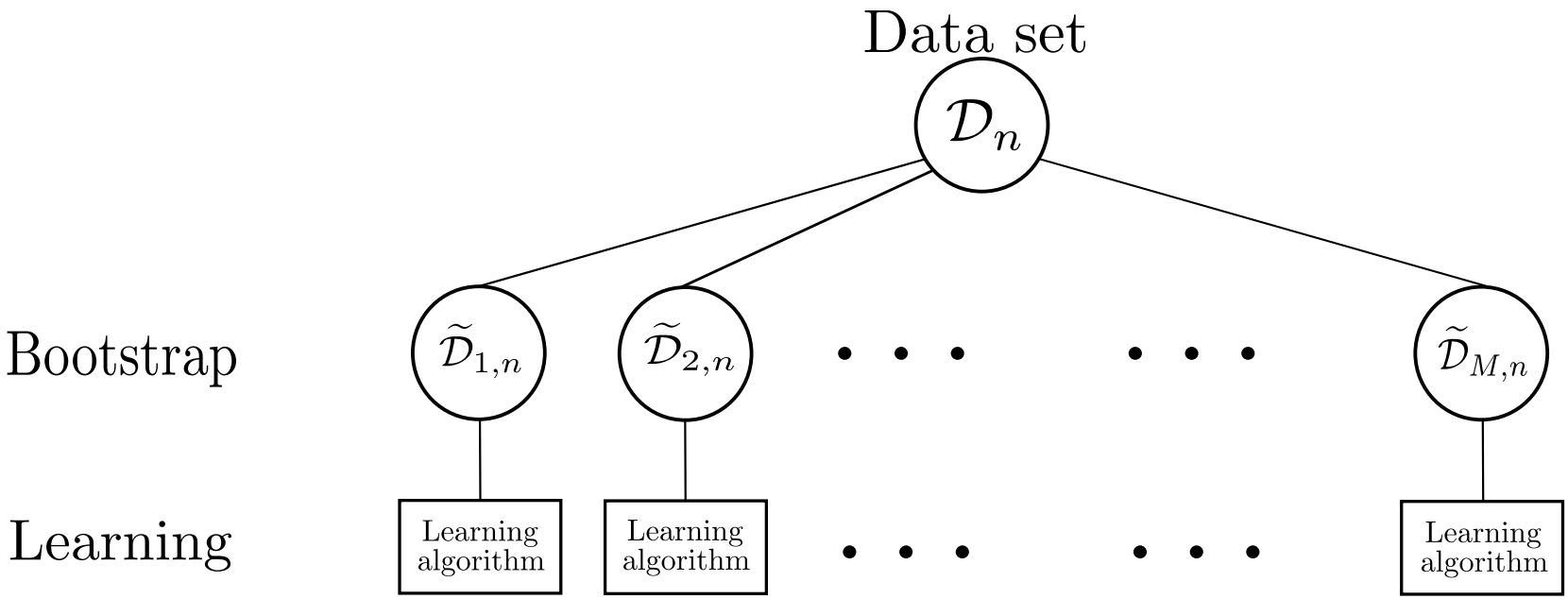
Bagging - Averaging predictors via data set resampling



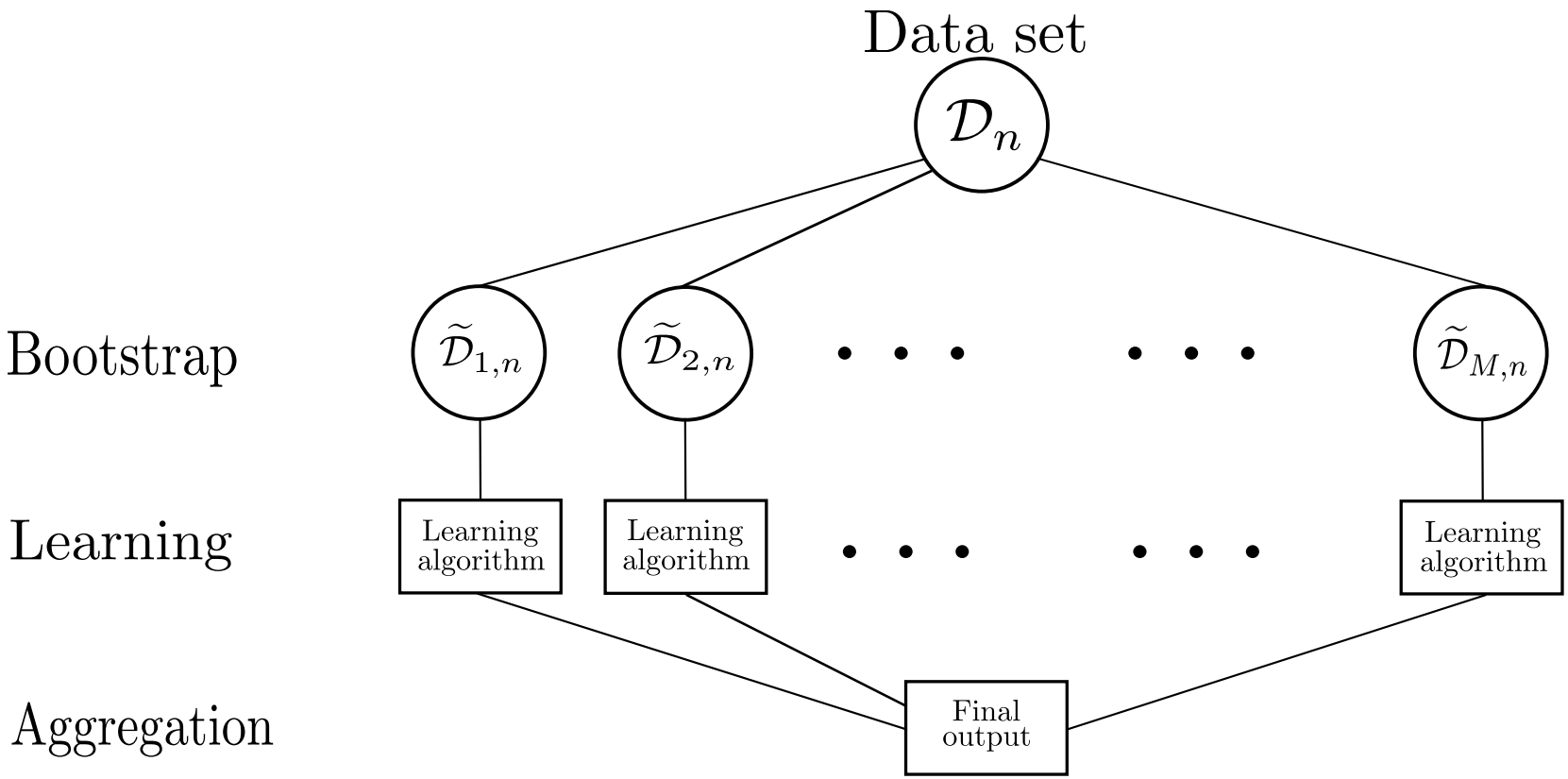
Bagging - Averaging predictors via data set resampling



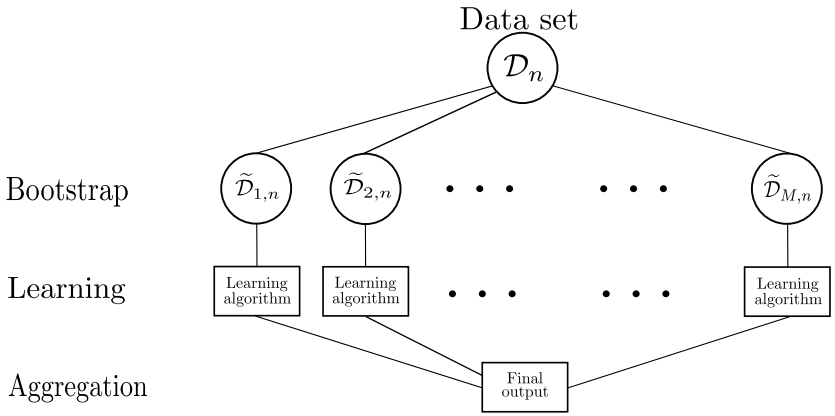
Bagging - Averaging predictors via data set resampling



Bagging - Averaging predictors via data set resampling



Bagging - Averaging predictors via data set resampling



Interests:

- Increase stability - data modification has less impact on the final predictor
- Parallel method - computationnally efficient
- Can be applied to a wide range of learning algorithm (for example, decision trees!)

Inconvenient: individual predictors may be too correlated (built on similar observations).

Split randomization in tree construction

Random forests

Two randomization ingredients:

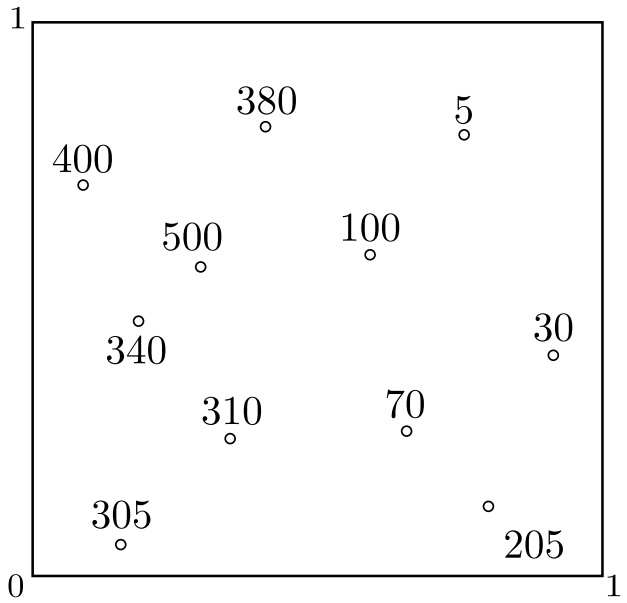
- Bagging
- Split randomization

Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.

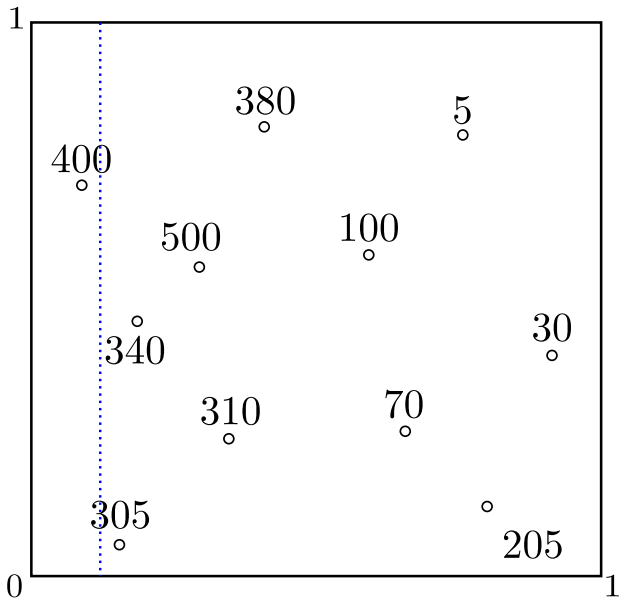
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



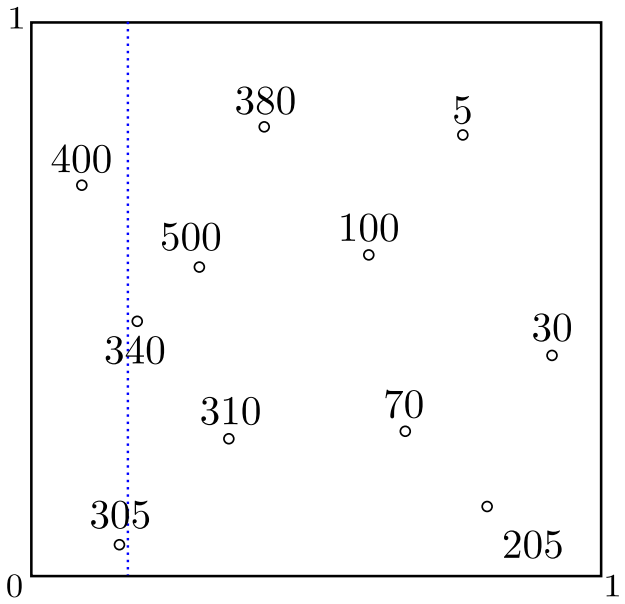
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



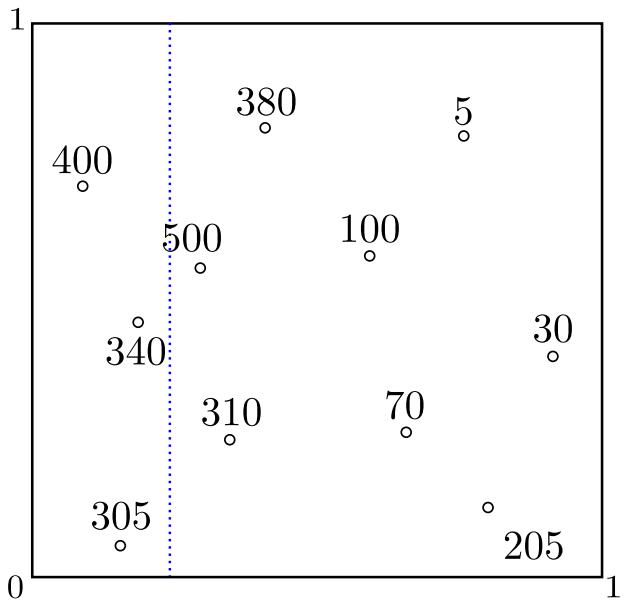
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



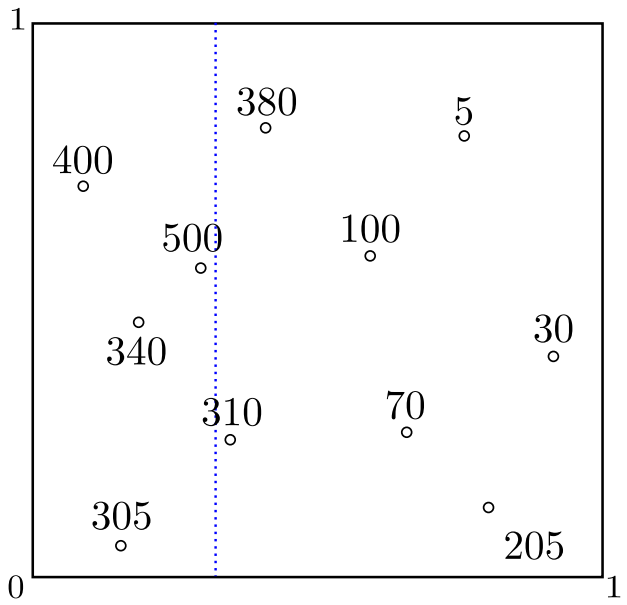
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



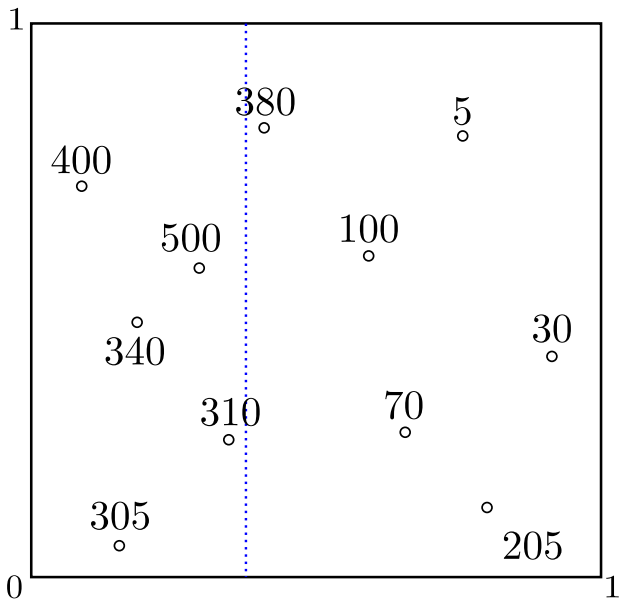
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



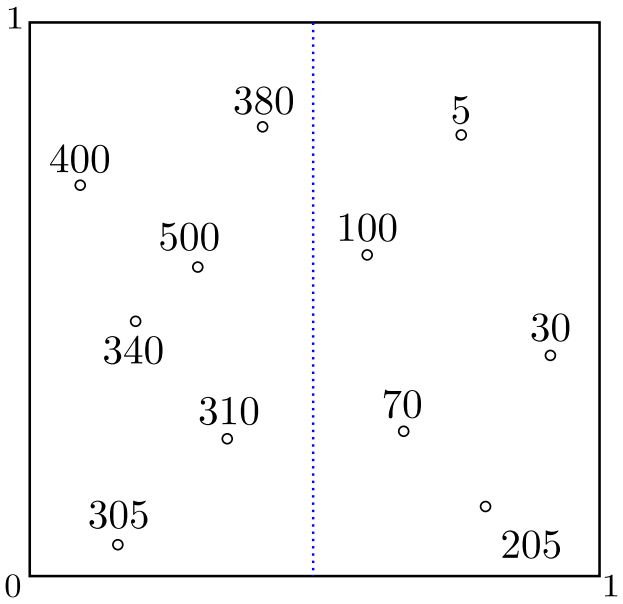
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



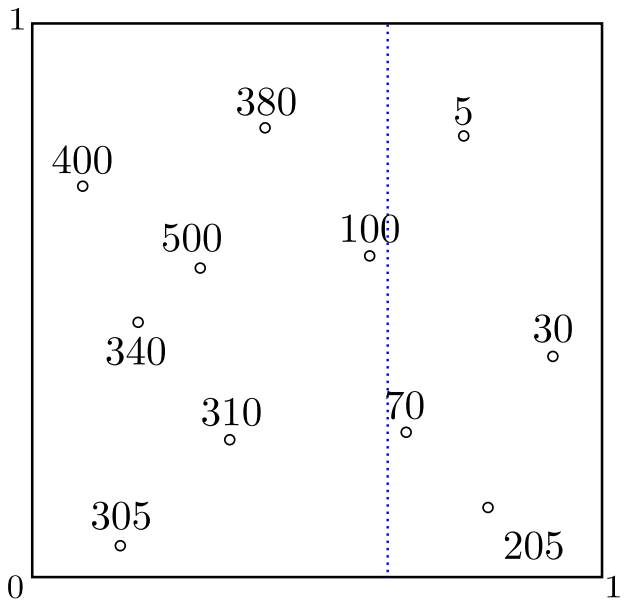
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



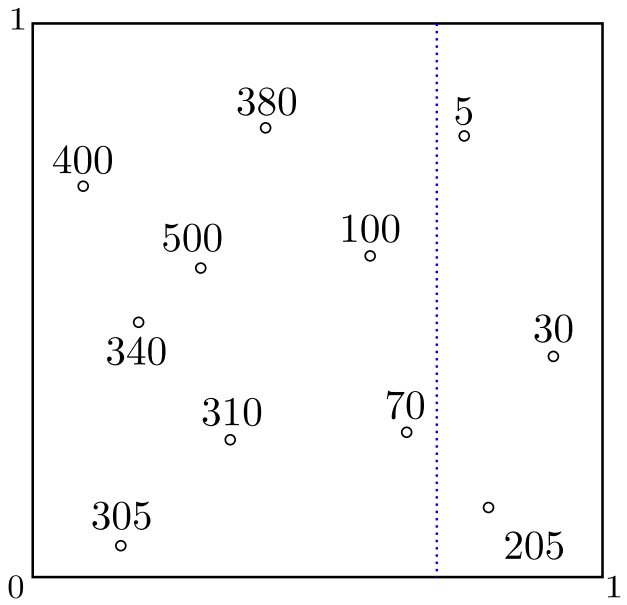
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



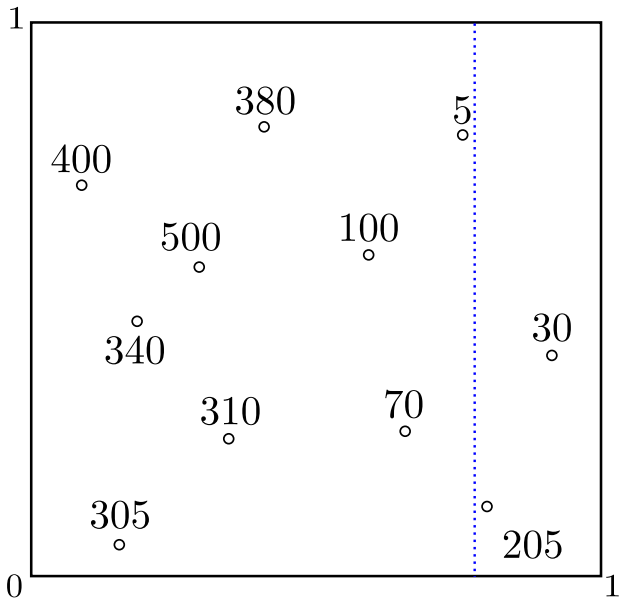
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



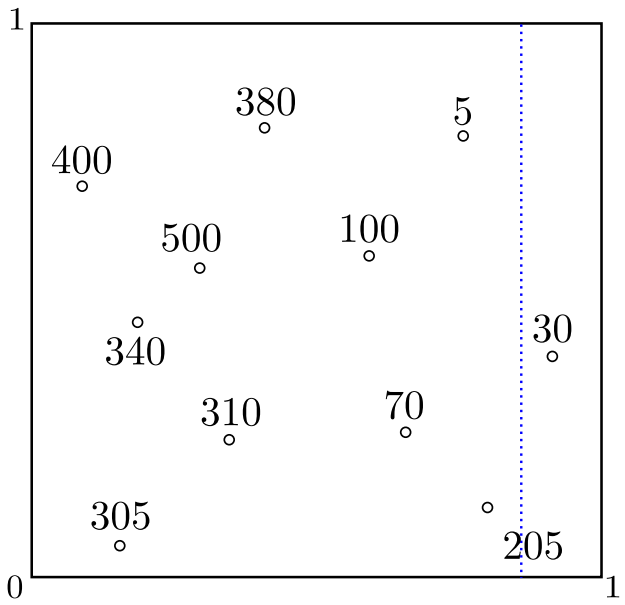
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



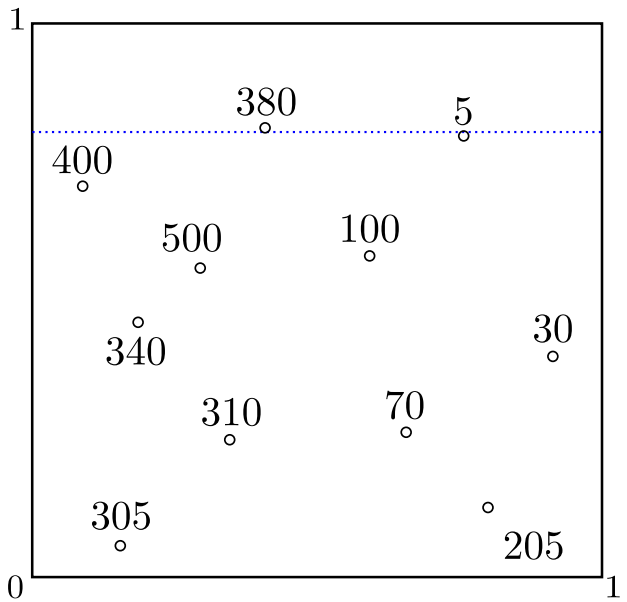
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



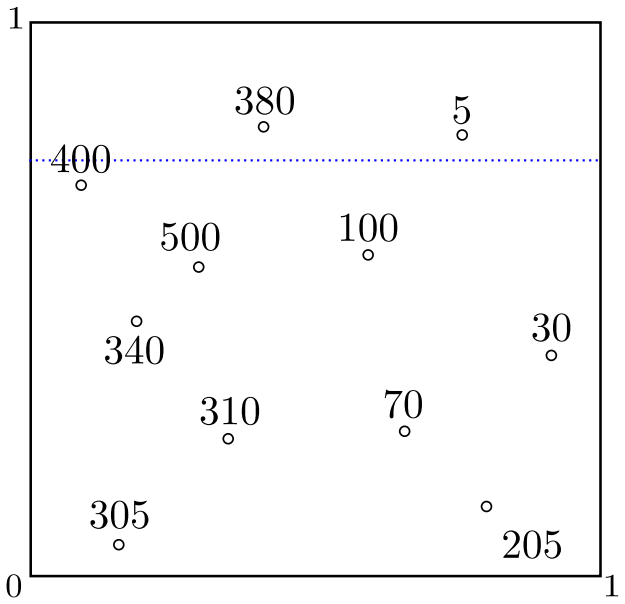
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



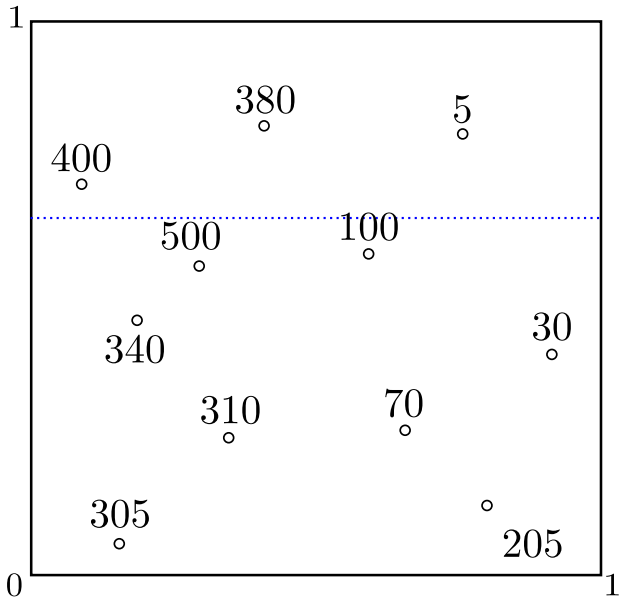
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



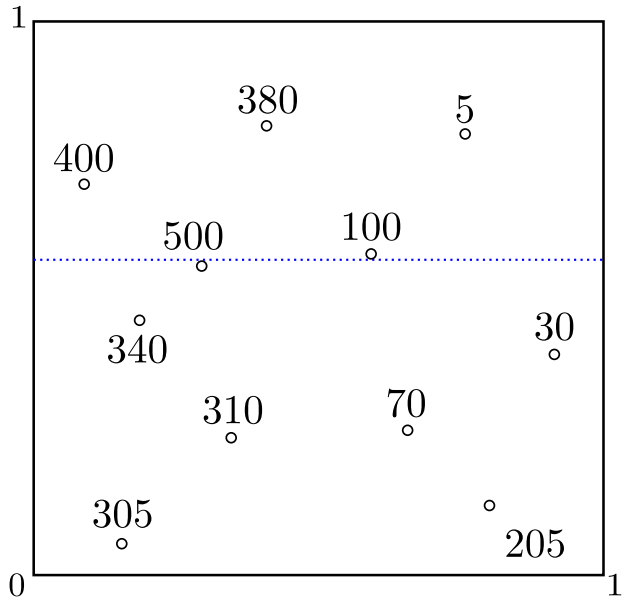
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



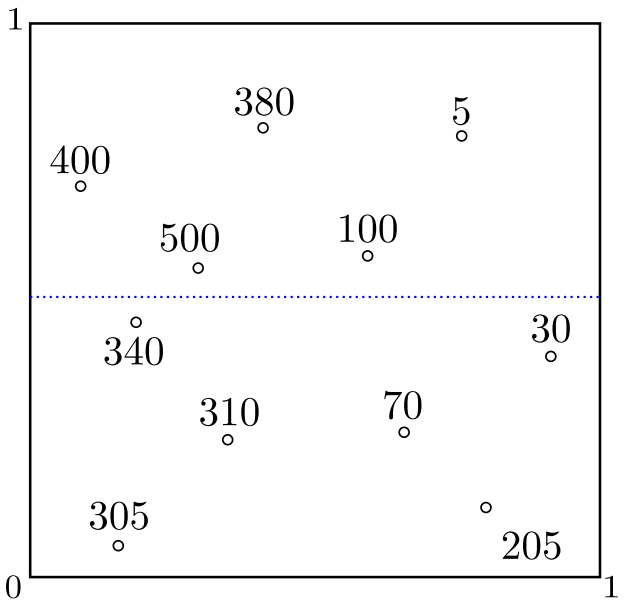
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



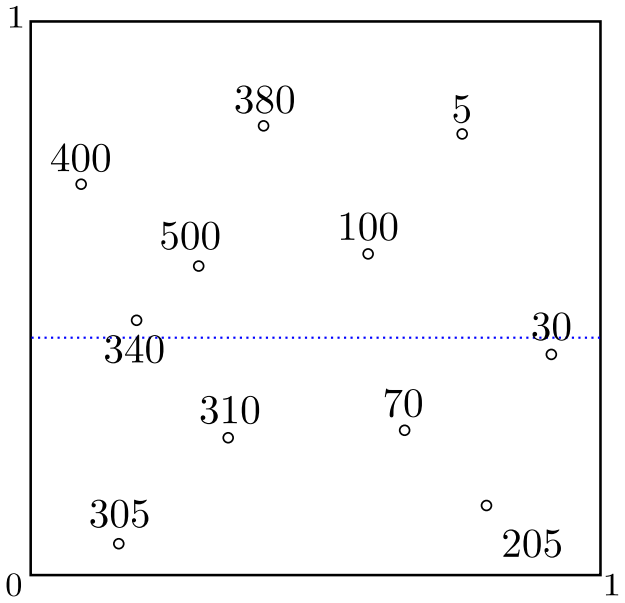
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



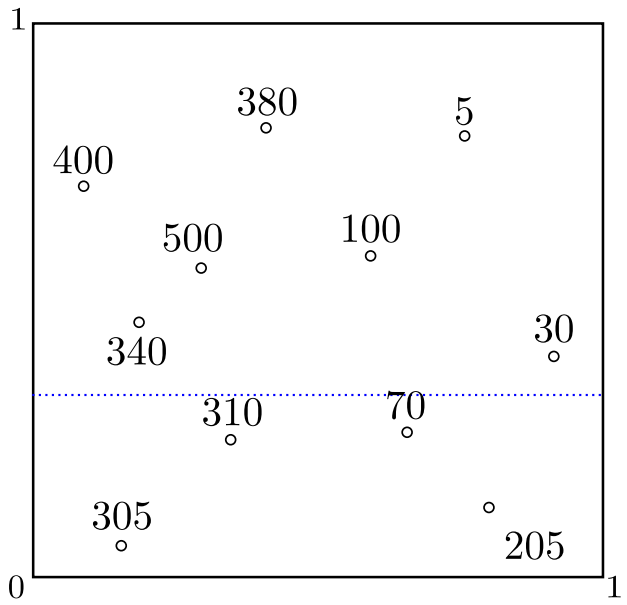
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



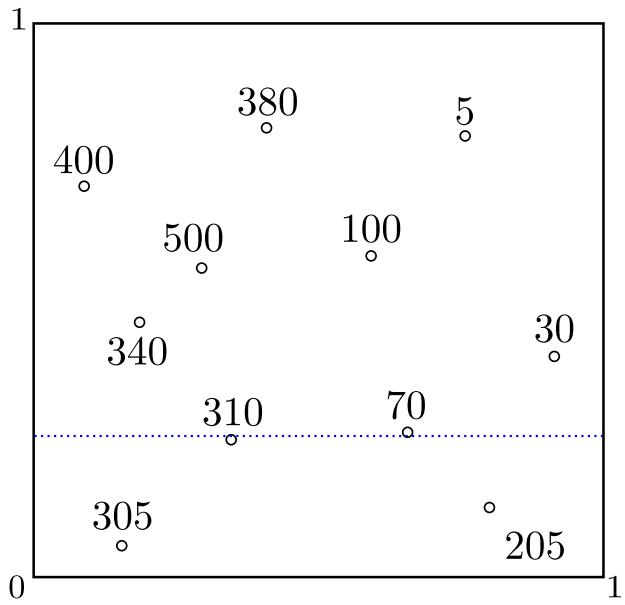
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



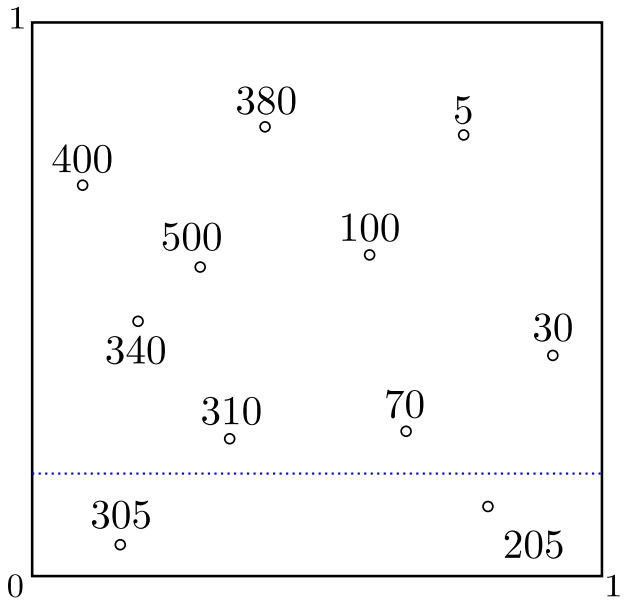
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



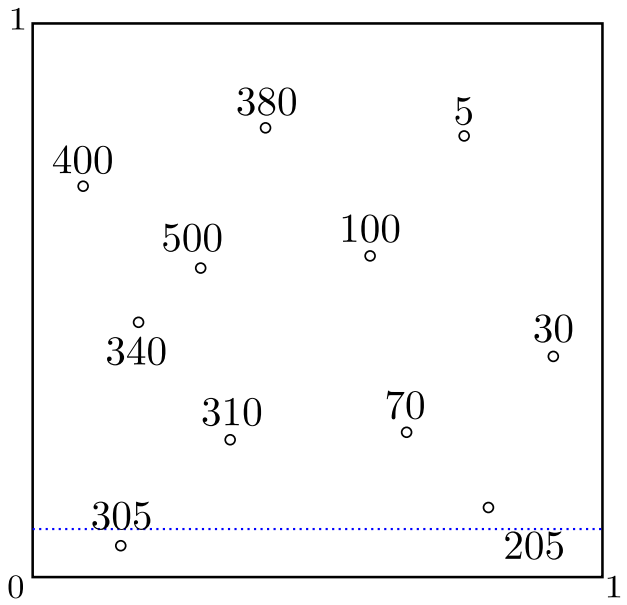
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



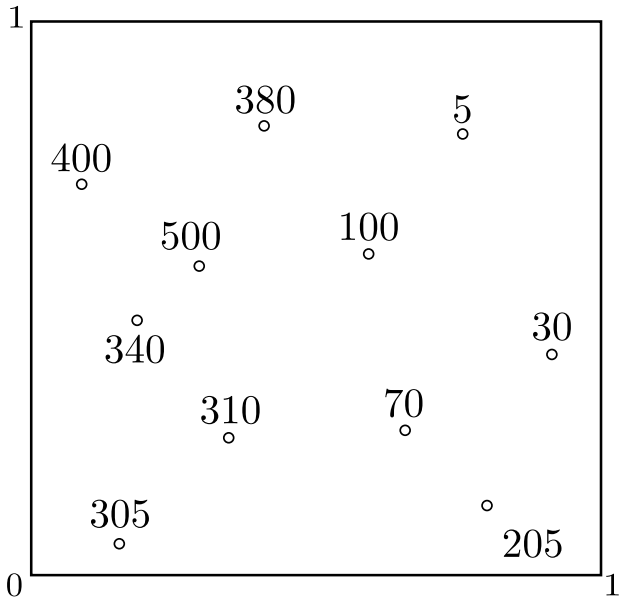
Split randomization in tree construction

Regular split selection. In each cell of a tree, select the best split, by optimizing the splitting criterion along all directions.



Split randomization in tree construction

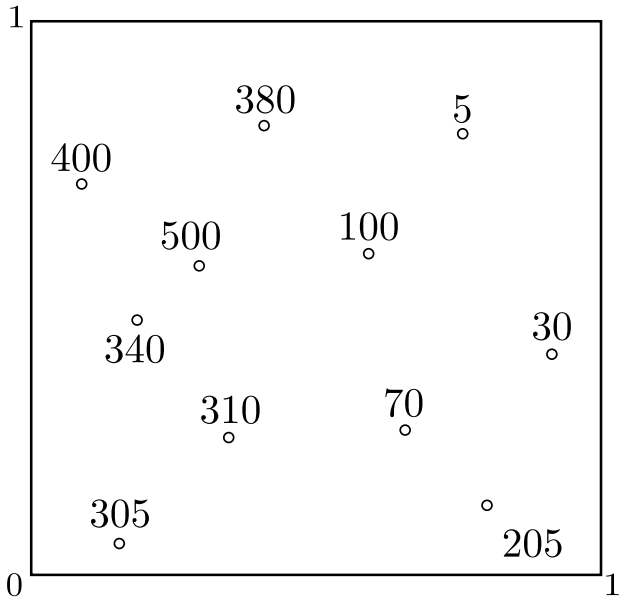
Split randomization.
In each cell of a tree, select **uniformly at random a prespecified number of directions**. Select the best split (by optimizing the splitting criterion) **along these directions only**.



Split randomization in tree construction

Split randomization.
In each cell of a tree, select **uniformly at random a prespecified number of directions**. Select the best split (by optimizing the splitting criterion) **along these directions only**.

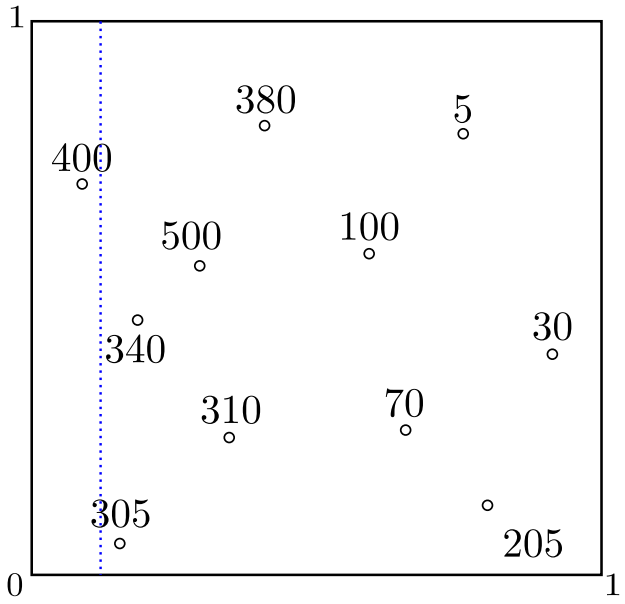
Here, for example, randomly selecting the first direction (variable $X^{(1)}$) leads to considering the following splits only.



Split randomization in tree construction

Split randomization.
In each cell of a tree, select **uniformly at random a prespecified number of directions**. Select the best split (by optimizing the splitting criterion) **along these directions only**.

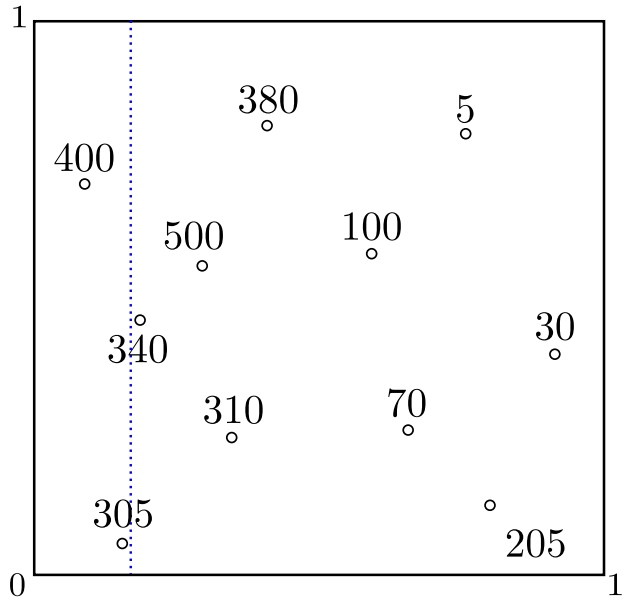
Here, for example, randomly selecting the first direction (variable $X^{(1)}$) leads to considering the following splits only.



Split randomization in tree construction

Split randomization.
In each cell of a tree, select **uniformly at random a prespecified number of directions**. Select the best split (by optimizing the splitting criterion) **along these directions only**.

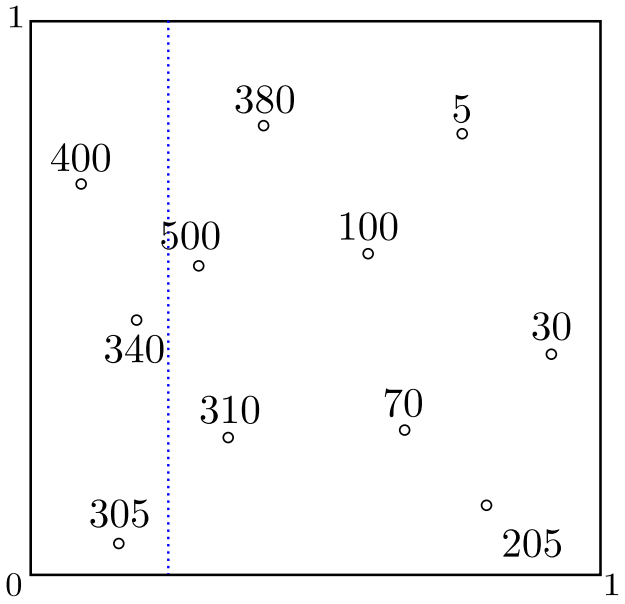
Here, for example, randomly selecting the first direction (variable $X^{(1)}$) leads to considering the following splits only.



Split randomization in tree construction

Split randomization.
In each cell of a tree, select **uniformly at random a prespecified number of directions**. Select the best split (by optimizing the splitting criterion) **along these directions only**.

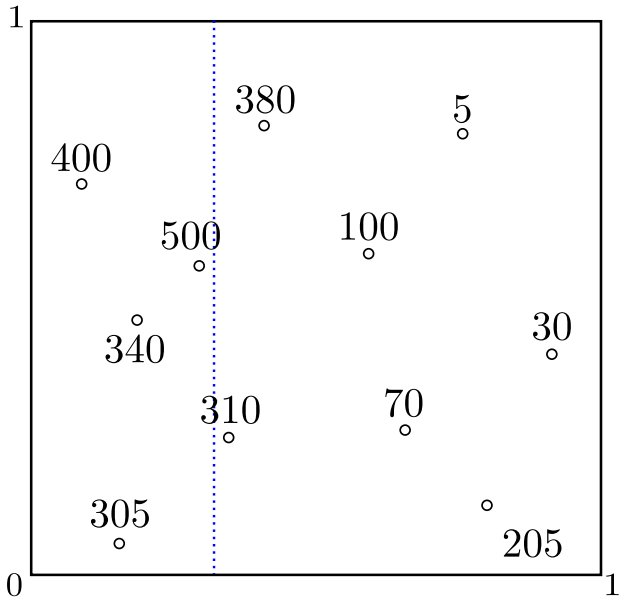
Here, for example, randomly selecting the first direction (variable $X^{(1)}$) leads to considering the following splits only.



Split randomization in tree construction

Split randomization.
In each cell of a tree, select **uniformly at random a prespecified number of directions**. Select the best split (by optimizing the splitting criterion) **along these directions only**.

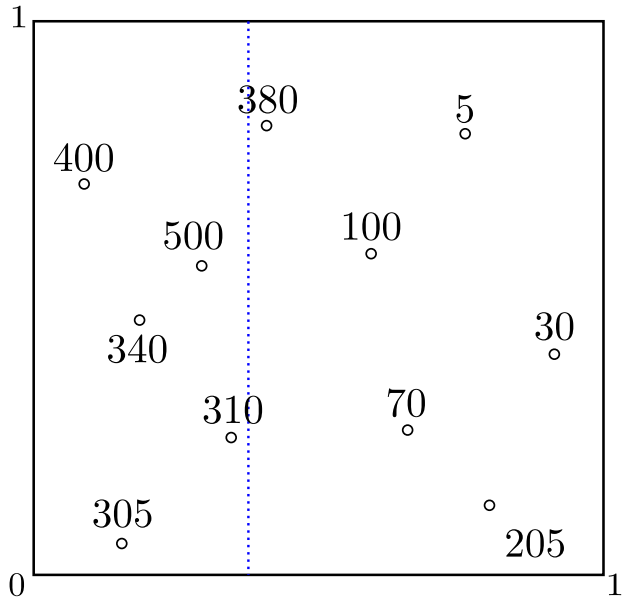
Here, for example, randomly selecting the first direction (variable $X^{(1)}$) leads to considering the following splits only.



Split randomization in tree construction

Split randomization.
In each cell of a tree, select **uniformly at random a prespecified number of directions**. Select the best split (by optimizing the splitting criterion) **along these directions only**.

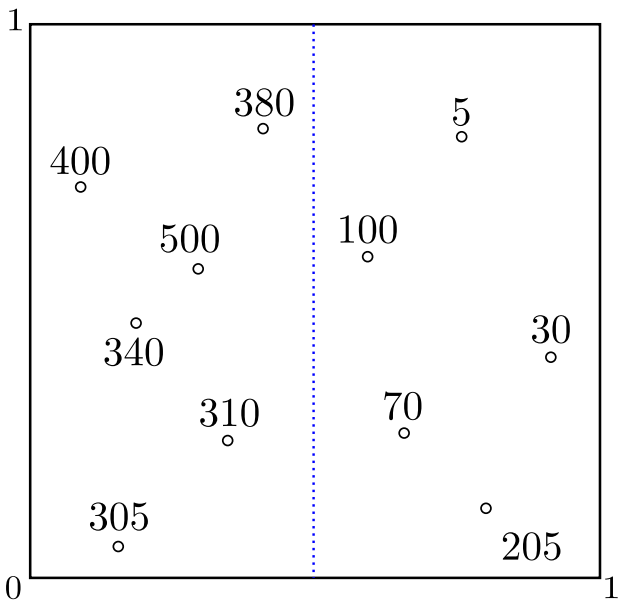
Here, for example, randomly selecting the first direction (variable $X^{(1)}$) leads to considering the following splits only.



Split randomization in tree construction

Split randomization.
In each cell of a tree, select **uniformly at random a prespecified number of directions**. Select the best split (by optimizing the splitting criterion) **along these directions only**.

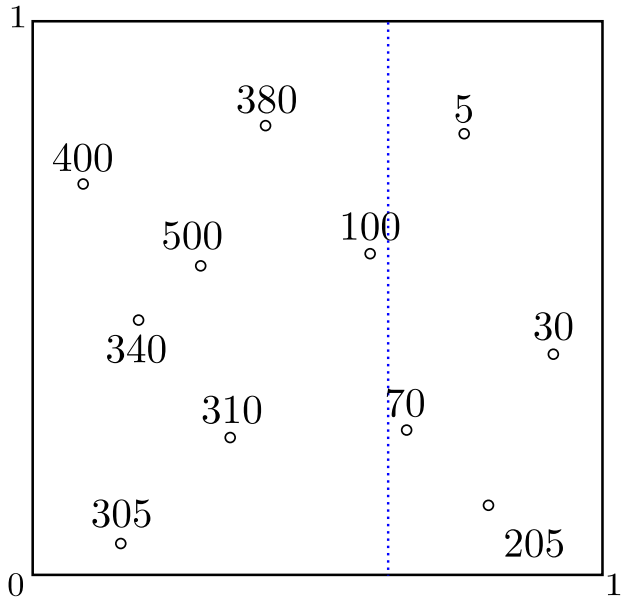
Here, for example, randomly selecting the first direction (variable $X^{(1)}$) leads to considering the following splits only.



Split randomization in tree construction

Split randomization.
In each cell of a tree, select **uniformly at random a prespecified number of directions**. Select the best split (by optimizing the splitting criterion) **along these directions only**.

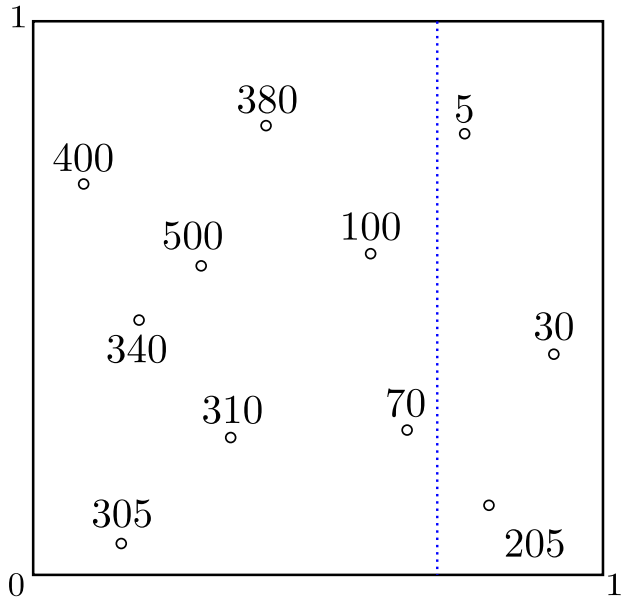
Here, for example, randomly selecting the first direction (variable $X^{(1)}$) leads to considering the following splits only.



Split randomization in tree construction

Split randomization.
In each cell of a tree, select **uniformly at random a prespecified number of directions**. Select the best split (by optimizing the splitting criterion) **along these directions only**.

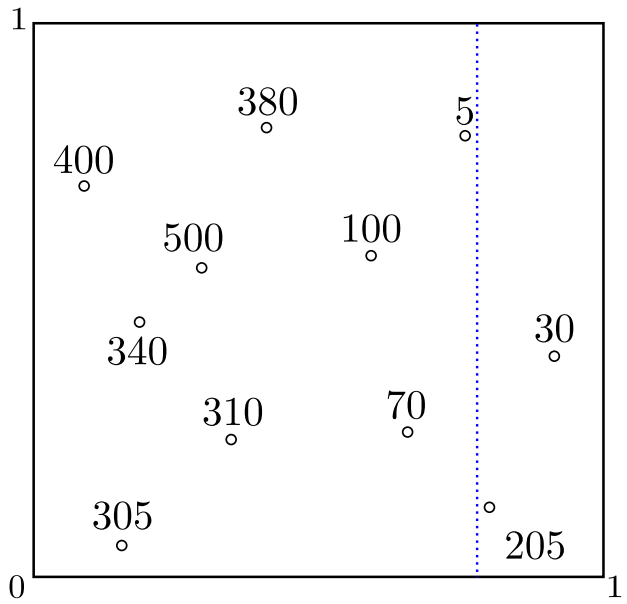
Here, for example, randomly selecting the first direction (variable $X^{(1)}$) leads to considering the following splits only.



Split randomization in tree construction

Split randomization.
In each cell of a tree, select **uniformly at random a prespecified number of directions**. Select the best split (by optimizing the splitting criterion) **along these directions only**.

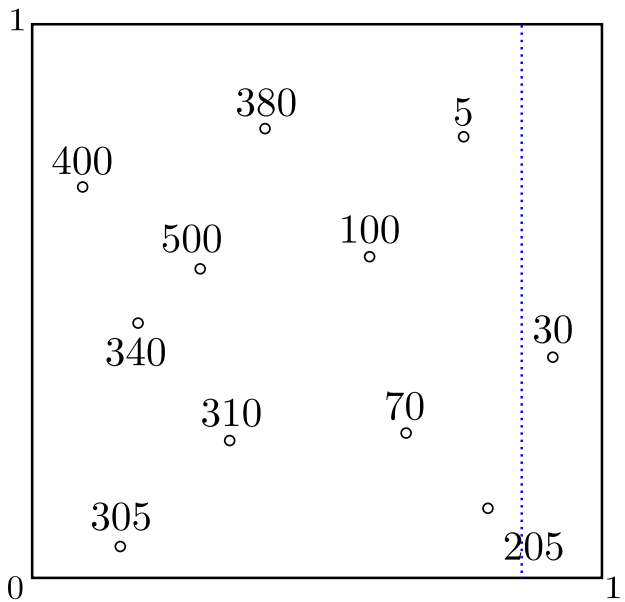
Here, for example, randomly selecting the first direction (variable $X^{(1)}$) leads to considering the following splits only.



Split randomization in tree construction

Split randomization.
In each cell of a tree, select **uniformly at random a prespecified number of directions**. Select the best split (by optimizing the splitting criterion) **along these directions only**.

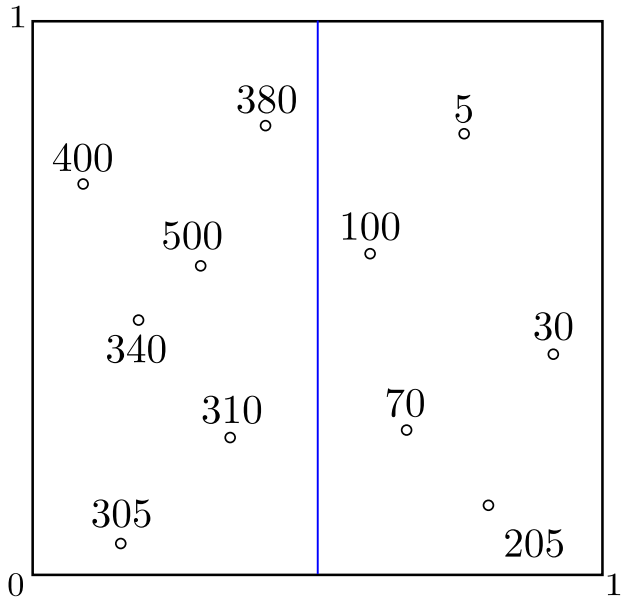
Here, for example, randomly selecting the first direction (variable $X^{(1)}$) leads to considering the following splits only.



Split randomization in tree construction

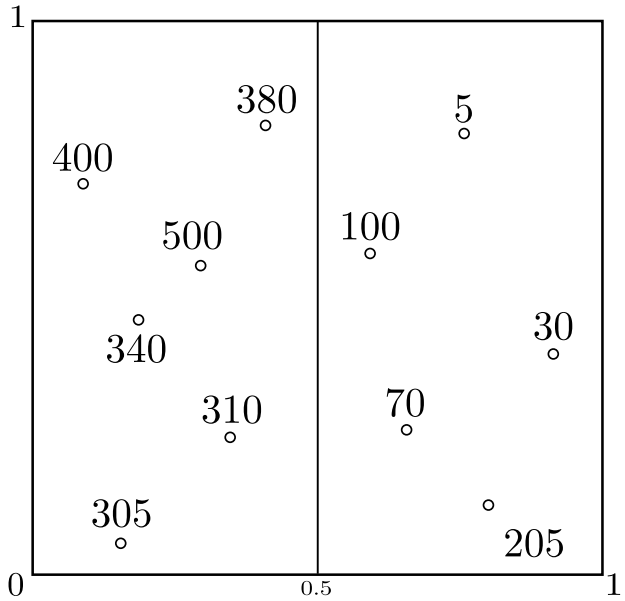
Split randomization.
In each cell of a tree, select **uniformly at random a prespecified number of directions**. Select the best split (by optimizing the splitting criterion) **along these directions only**.

Here, for example, randomly selecting the first direction (variable $X^{(1)}$) leads to selecting this split.



Split randomization in tree construction

Split randomization.
In each cell of a tree, select **uniformly at random a prespecified number of directions**. Select the best split (by optimizing the splitting criterion) **along these directions only**.

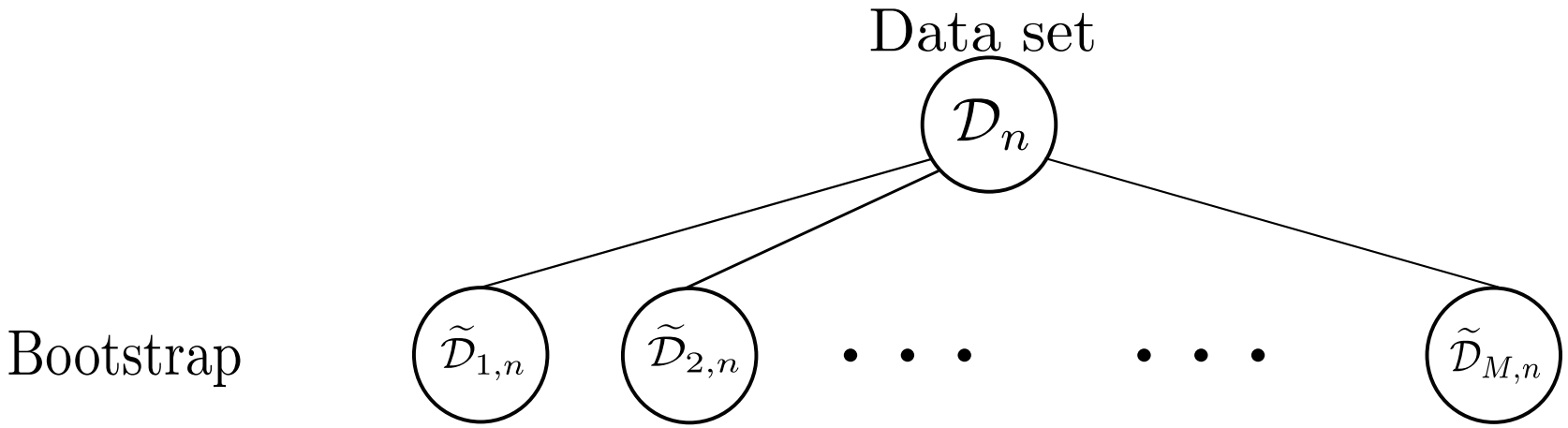


The same procedure is repeated on the resulting cells, with a new random choice of the splitting directions.

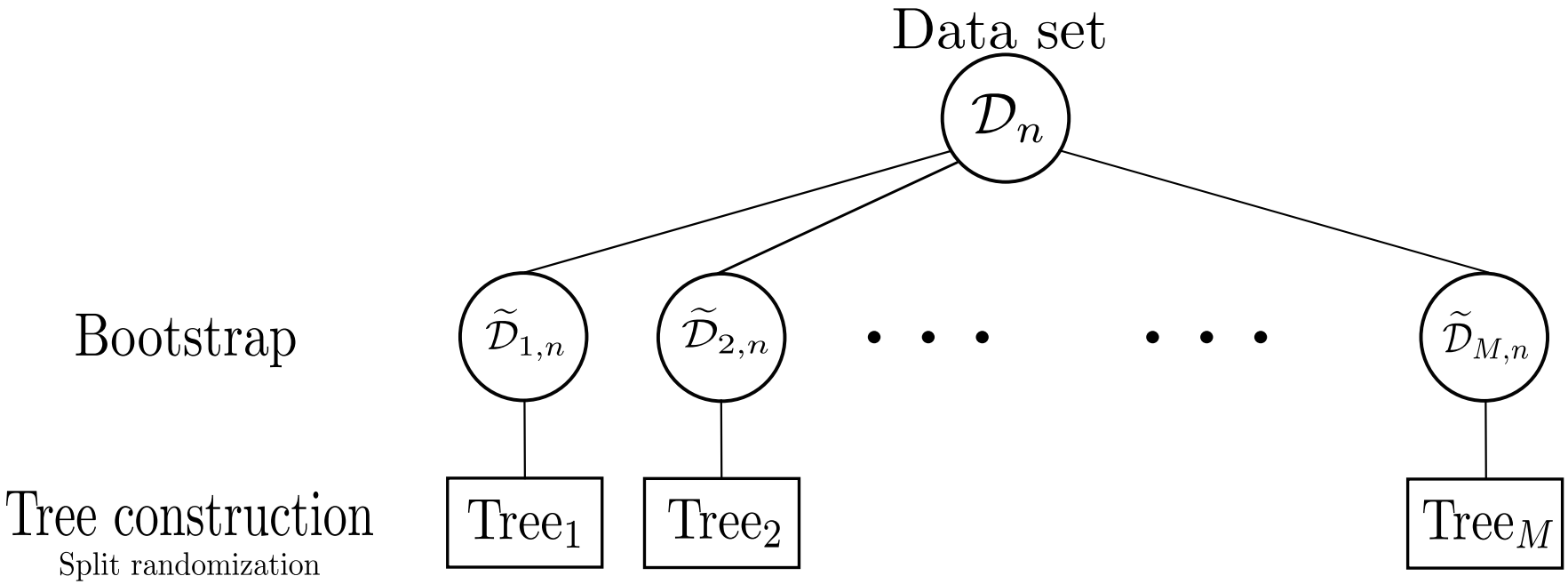
Outline

Construction of random forests

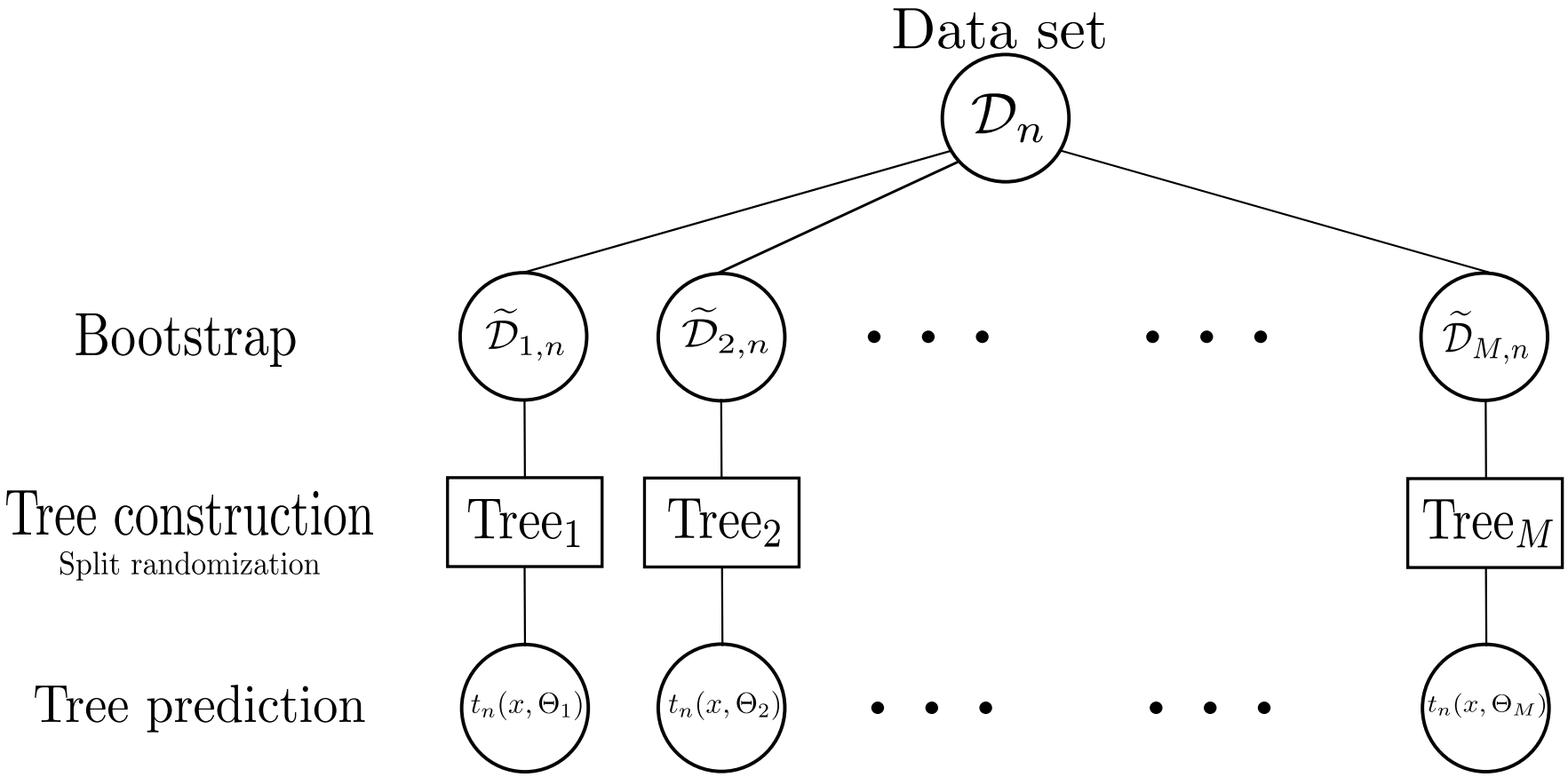
Construction of random forests



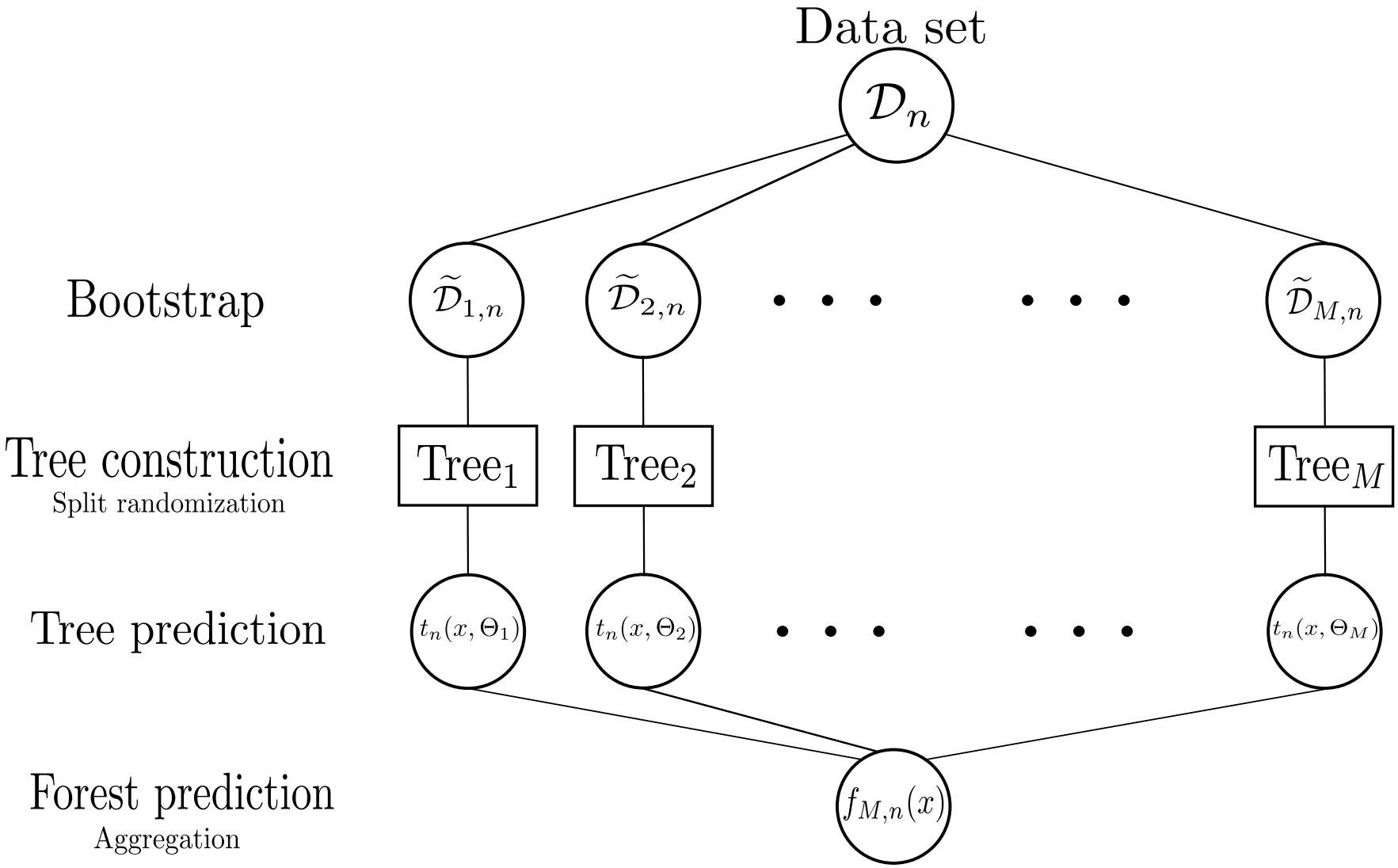
Construction of random forests



Construction of random forests



Construction of random forests



Construction of Breiman forests

- Build in parallel **`n-estimators`** CART as follows.

CART

Bootstrap - Select **`max-samples`** observations with replacement among the original sample $\mathcal{D}_n$. Use only these observations to build the tree.

For each cell,

Select randomly **`max-features`** coordinates among $\{1, \ldots, d\}$;
Choose the best split along previous directions, based on the choosen `criterion` (impurity measure).

Stop splitting each cell when all observations inside it have the same label or when a stopping criterion is met:

there are less than **`min-sample-leaf`** observation in the leaf
resulting cells would contain less than **`min-sample-split`** observation
cell is already split **`max-depth`** times
there are already **`max-leaf-nodes`** leaves

- Compute the forest prediction by averaging the predictions of all trees.

List of all random forest hyperparameters

See `Scikit-learn` documentation for more details: `RandomForestRegressor` / `RandomForestClassifier`.

List of all random forest hyperparameters

See `Scikit-learn` documentation for more details: `RandomForestRegressor` / `RandomForestClassifier`.

List of all hyperparameters in the forest:

- `n-estimators` = 100
- `criterion`='gini'
- `max-depth`=None,
  `min-samples-split` = 2,
  `min-samples-leaf` = 1,
  `min-weight-fraction-leaf` = 0.0,
  `min-impurity-decrease` = 0.0,
  `max-leaf-nodes`=None

  → By default, trees are fully grown with no pruning strategy
- `max-features`='sqrt' (classif.) 'None' (regression).
- `bootstrap`=True, `max-samples`=None

List of all random forest hyperparameters

See `Scikit-learn` documentation for more details: `RandomForestRegressor` / `RandomForestClassifier`.

Remarks.

- Due to bootstrap and split randomization, running twice RF may lead to different results. Increasing the number of trees limits this difference.
- Fixing a `random-state` makes two runs of RF identical.
- Beware, by default, split randomization is used in classification but not in regression!

Role of each hyperparameter

Number of trees.

- Larger values are better
- No statistical tradeoff between low and high values
- Limited by computational power - growing many trees is expensive
- Default values (several hundreds / thousands) usually do a good job

Role of each hyperparameter

Bootstrap size / tree shape.

- Control the bias/variance tradeoff: small bootstrap size / shallow trees lead to predictors with a large bias but a small variance
- Use small bootstrap size or shallow tree if data are very noisy
- Use default setting for modelling very complex phenomenon

$\rightarrow$ Precise tuning can help but default values are good in general

Role of each hyperparameter

Split randomization

- Most complex parameter to tune
- Small values of `max-features` lead to very different trees

  $\rightarrow$ `max-features`=1 corresponds to drawing randomly the splitting direction
- Large values of `max-features` lead to similar trees

  $\rightarrow$ `max-features`=d corresponds to building the same tree (if no bootstrap is used)
- No precise heuristic, can be tuned by cross-validation.

Outline

Out-of-bag error

Idea. Evaluate the error of a random forest using the fact that each observation has not been used in all tree constructions and can thus be used as test points for such aggregated trees.

Out-of-bag error

Idea. Evaluate the error of a random forest using the fact that each observation has not been used in all tree constructions and can thus be used as test points for such aggregated trees.

General procedure (short):

- Consider that a forest has been trained on the data set $\mathcal{D}_n$.
- For each observation $i \in \{1, \dots, n\}$,
  - Consider the bootstrap samples that do not contain this observation.
  - For the trees that are not built using observation i, compute the predictions at X_i and aggregate them. Compute the loss of such an aggregated prediction.
- Compute the Out-of-bag error by averaging the losses over all observations.

Out-of-bag error

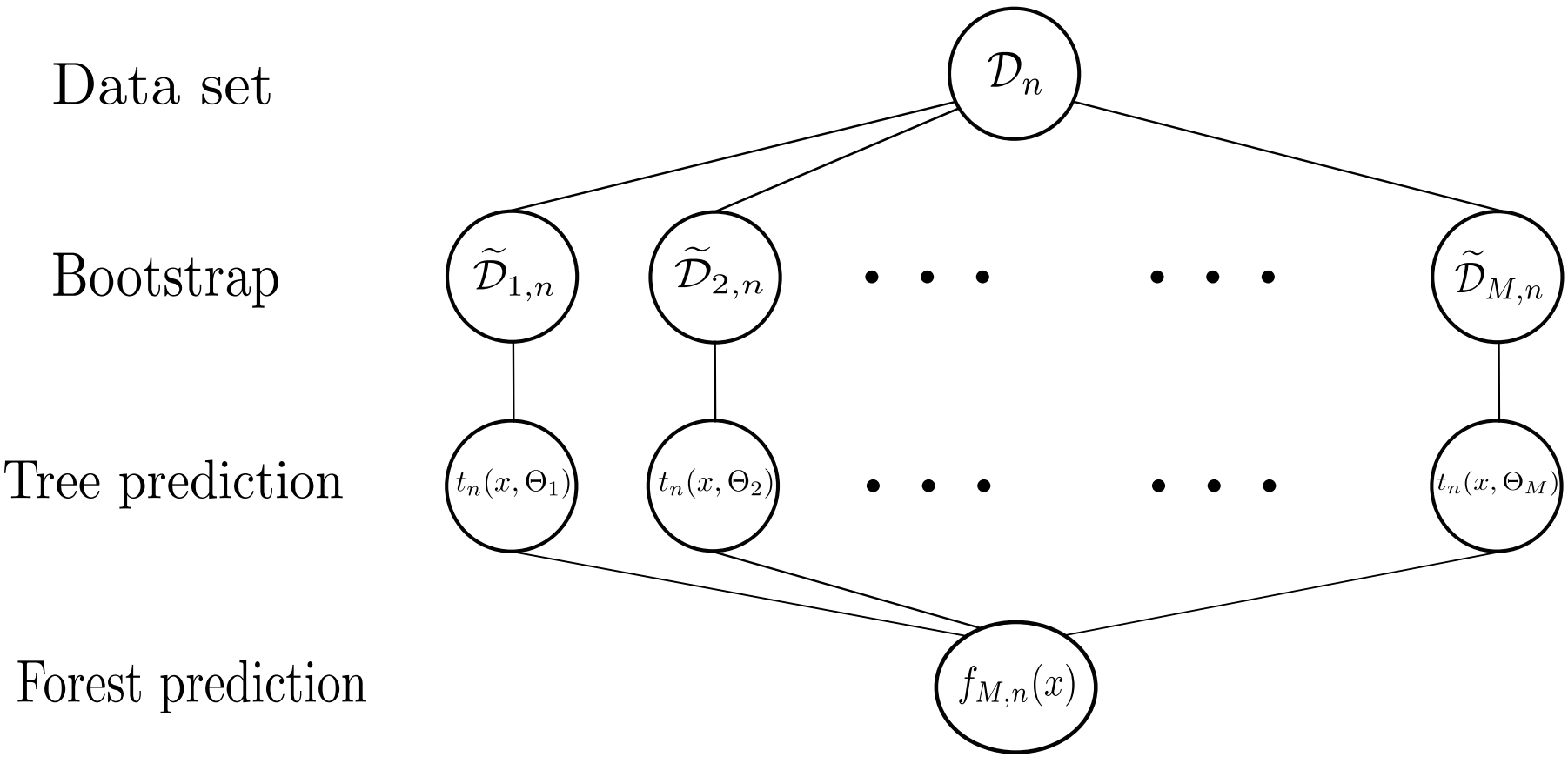
Benefits:

- No need for dividing the data set into a train and a test set
- Easily parallelizable
- Asymptotically equivalent to the risk of the forest for large M.

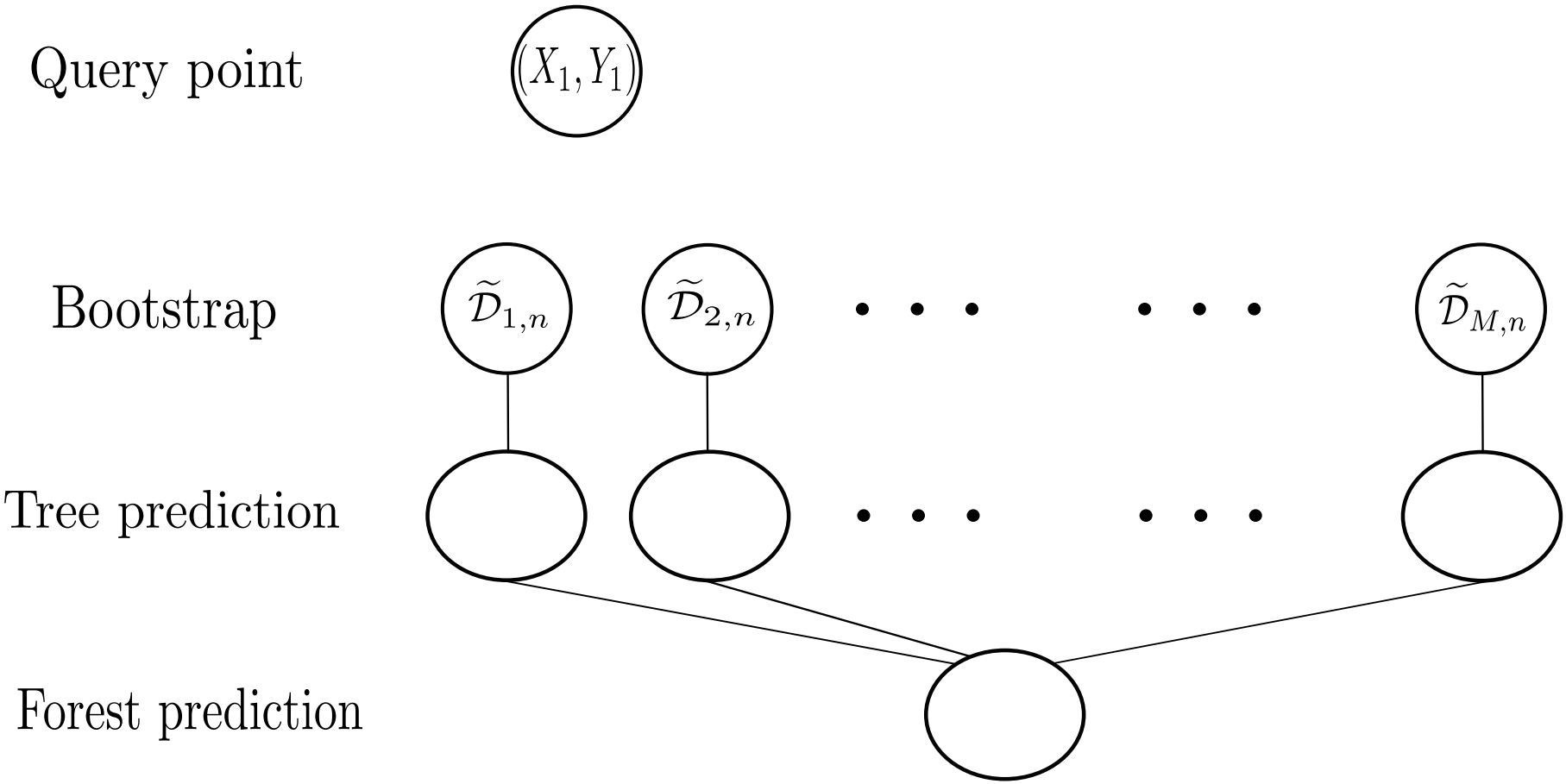
Drawback:

- Do not compute exactly the error of the whole forest but rather the aggregated error of some trees in the forest.

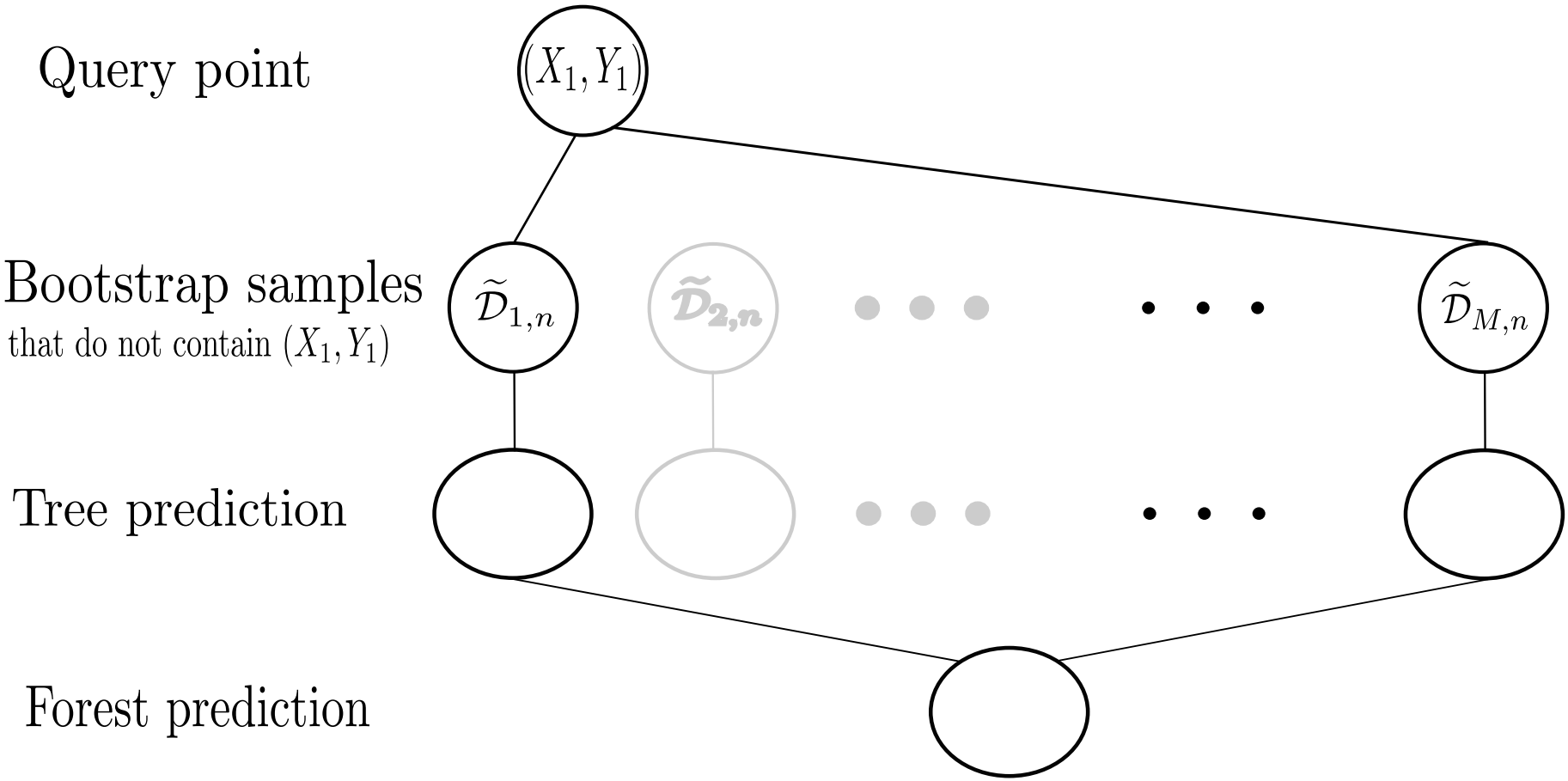
Out-of-bag procedure



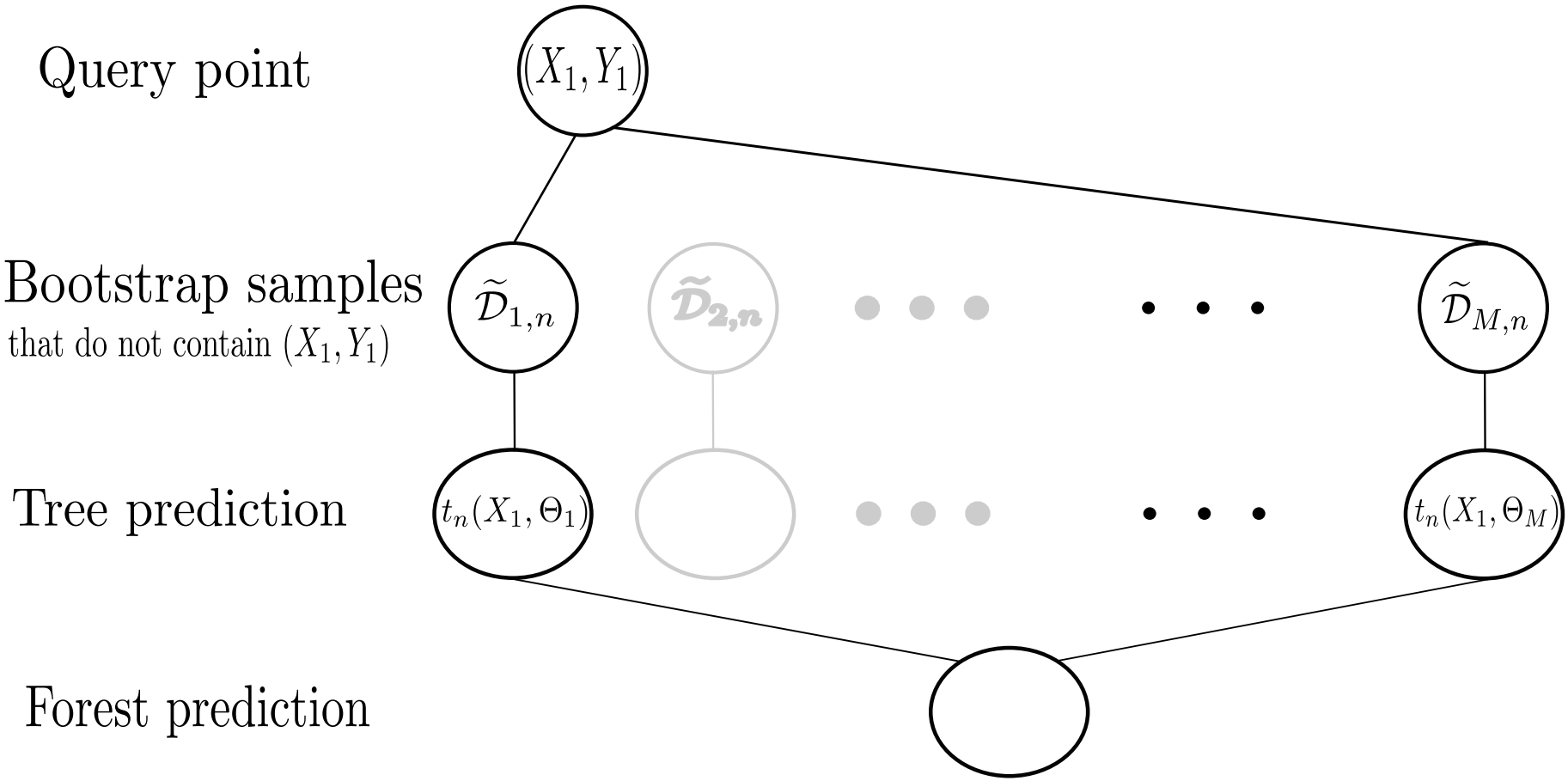
Out-of-bag procedure



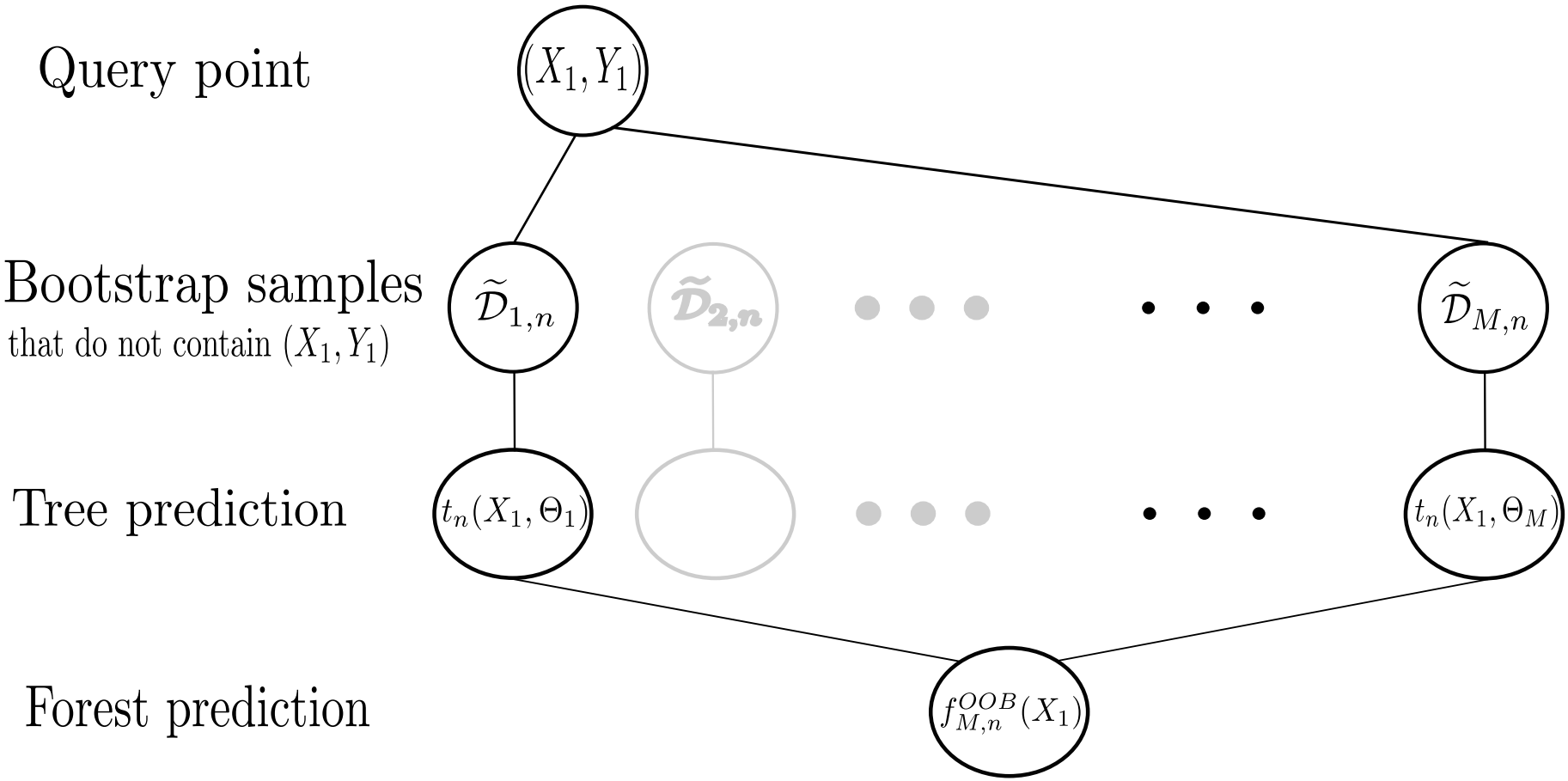
Out-of-bag procedure



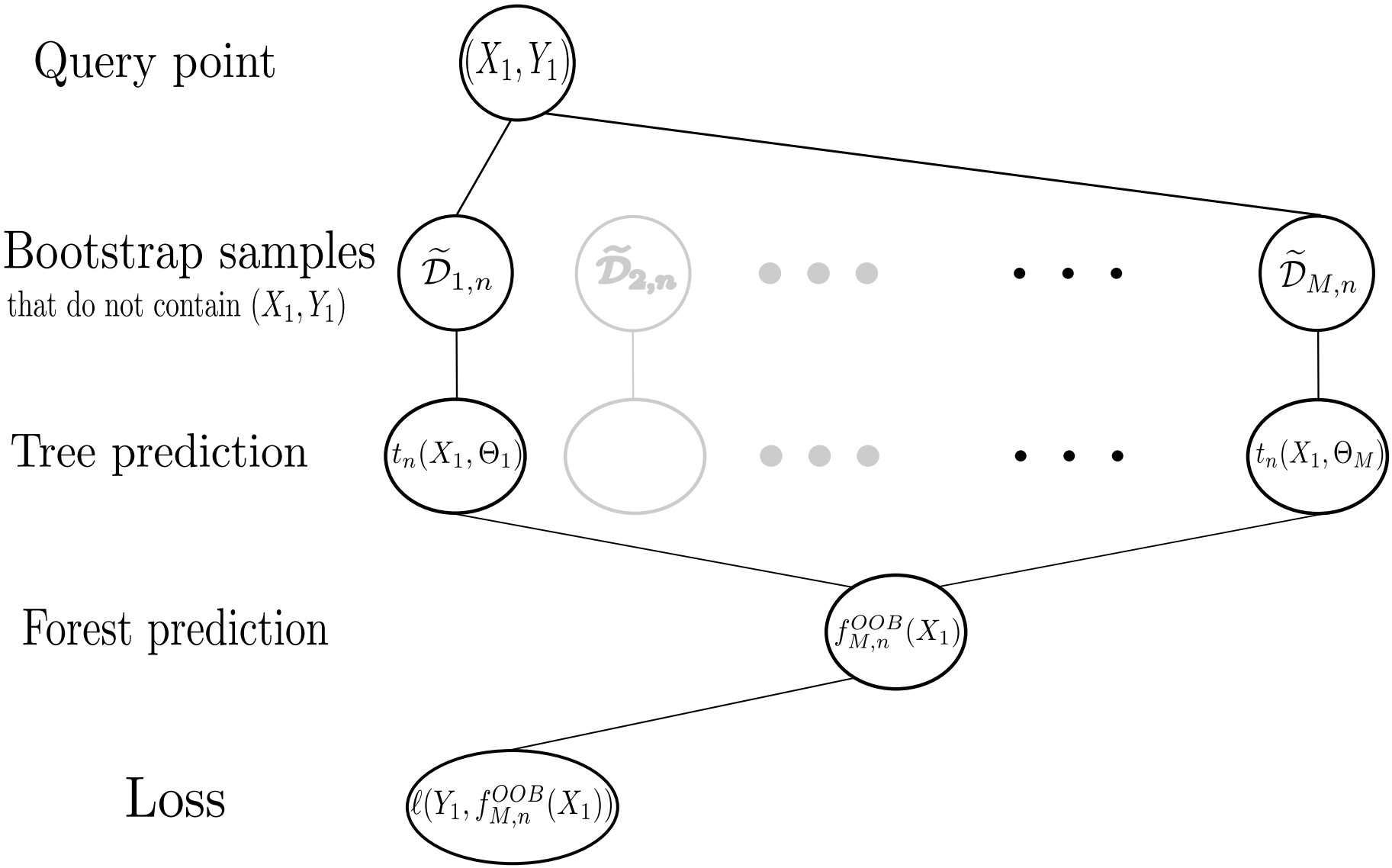
Out-of-bag procedure



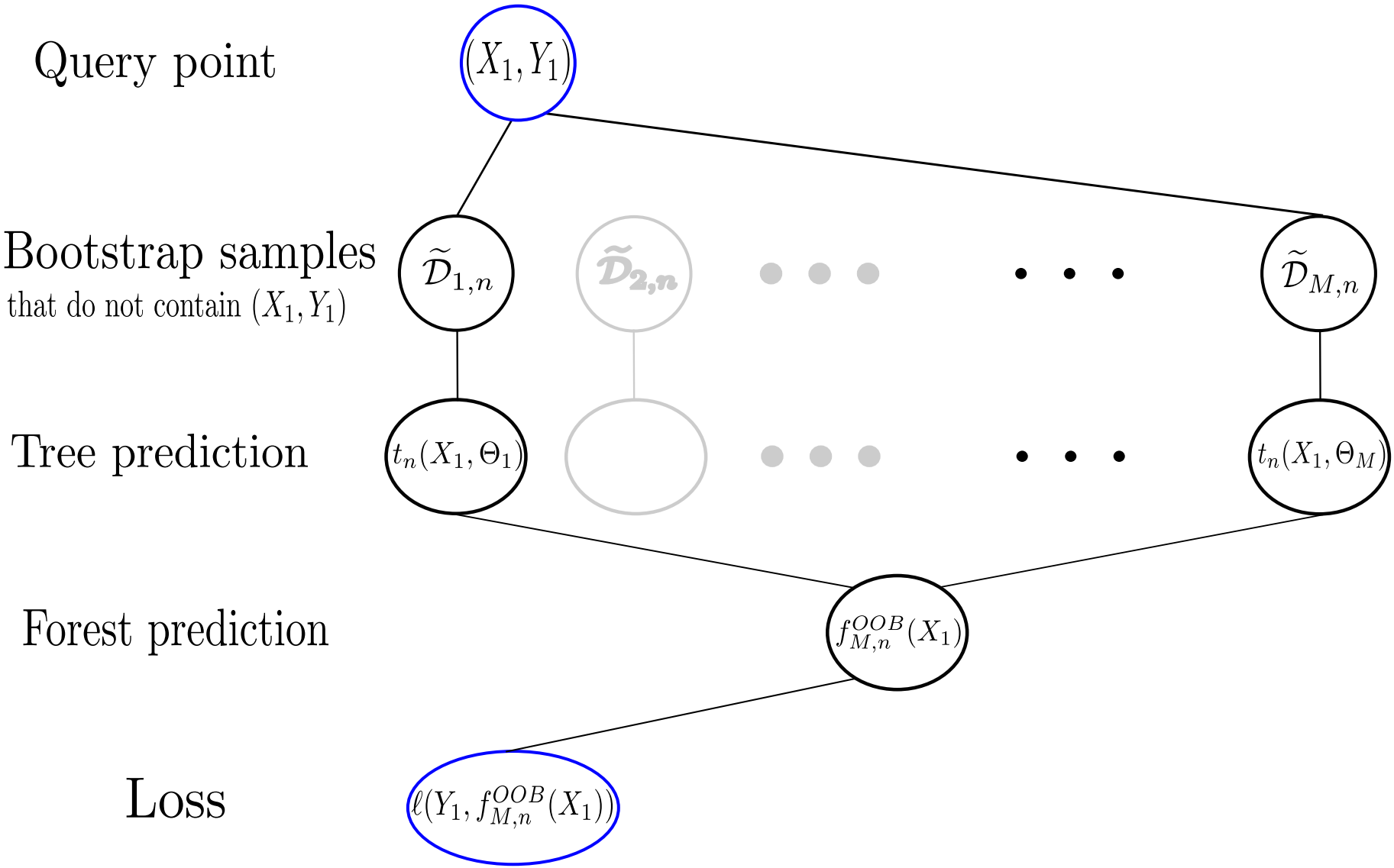
Out-of-bag procedure



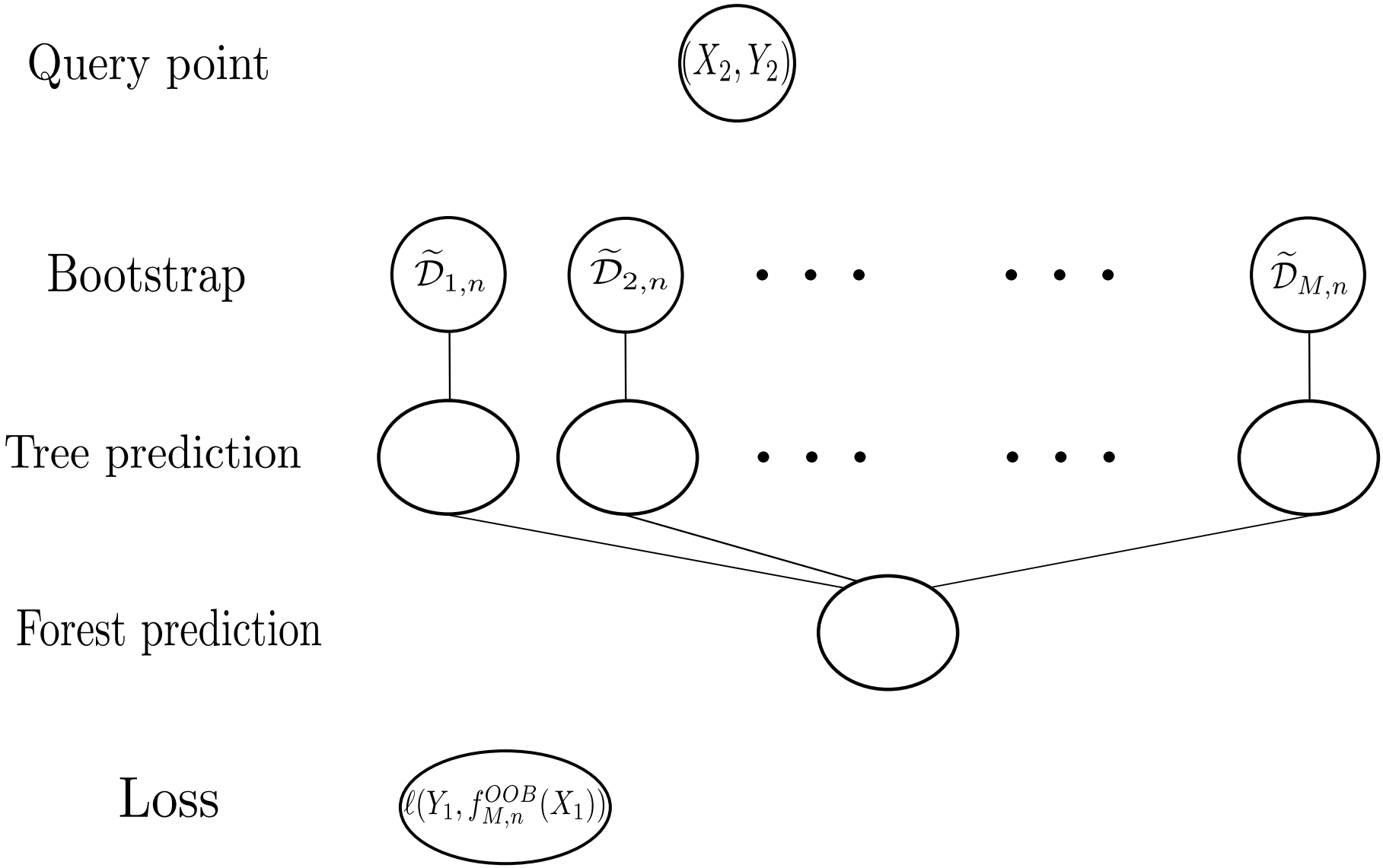
Out-of-bag procedure



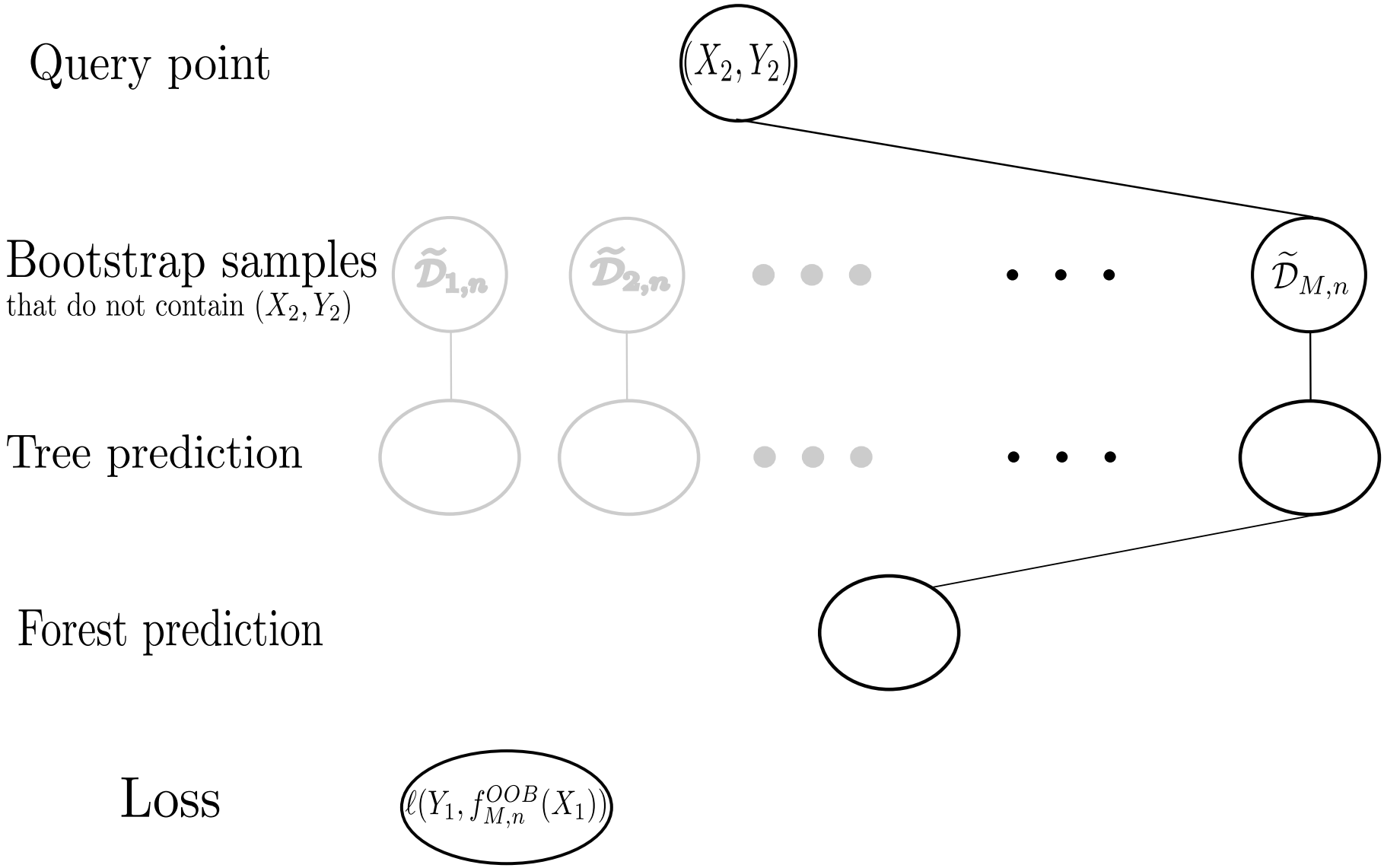
Out-of-bag procedure



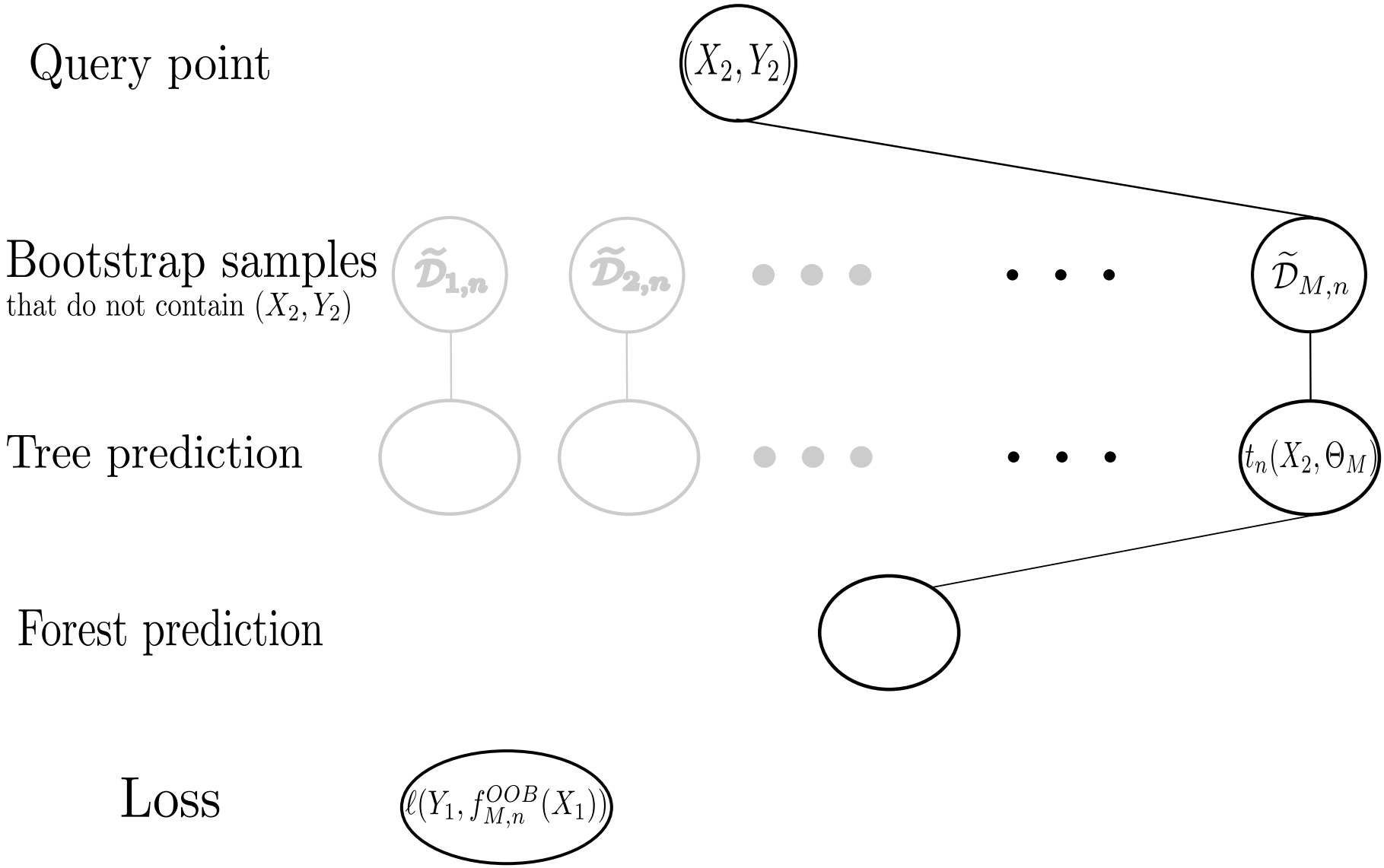
Out-of-bag procedure



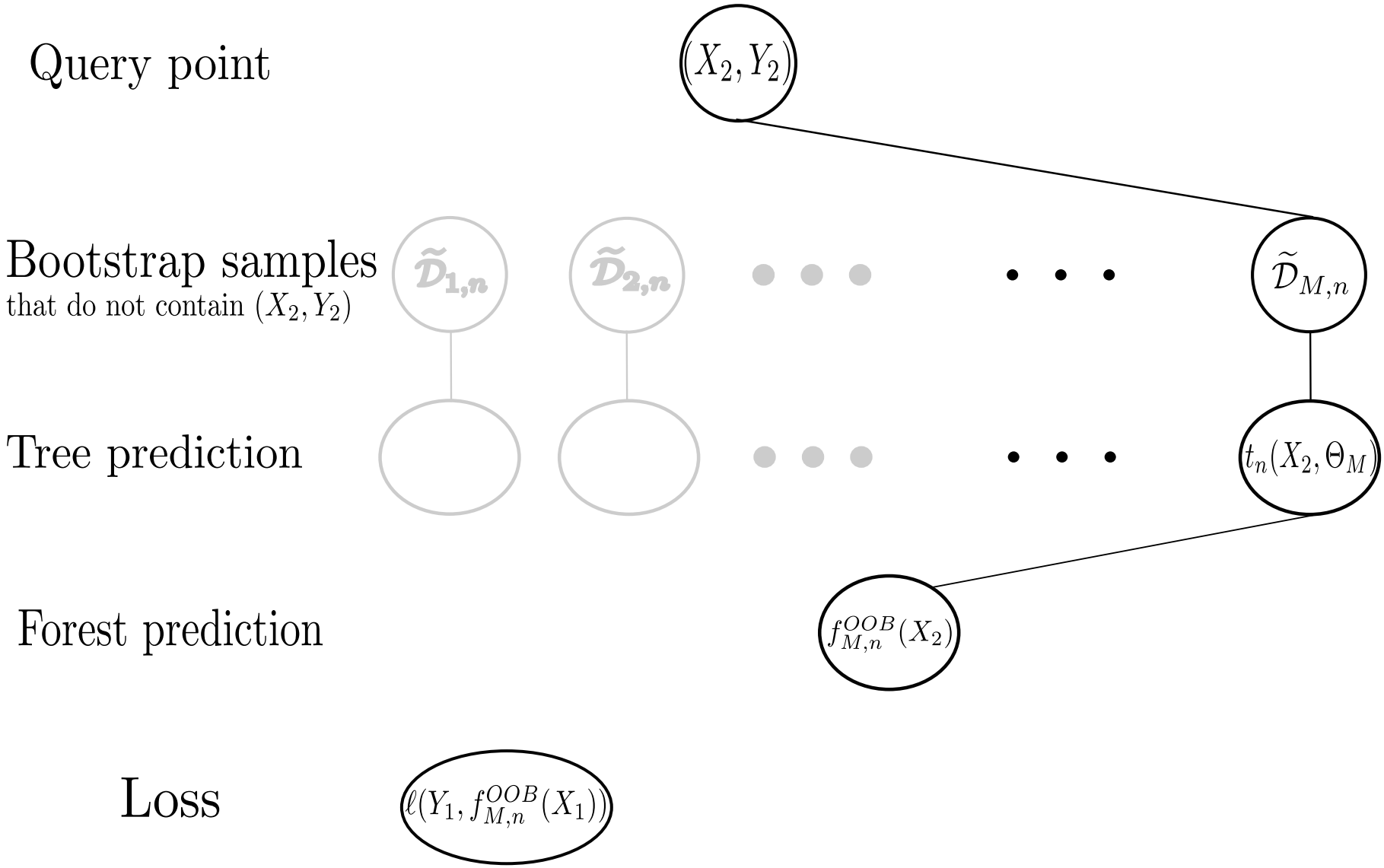
Out-of-bag procedure



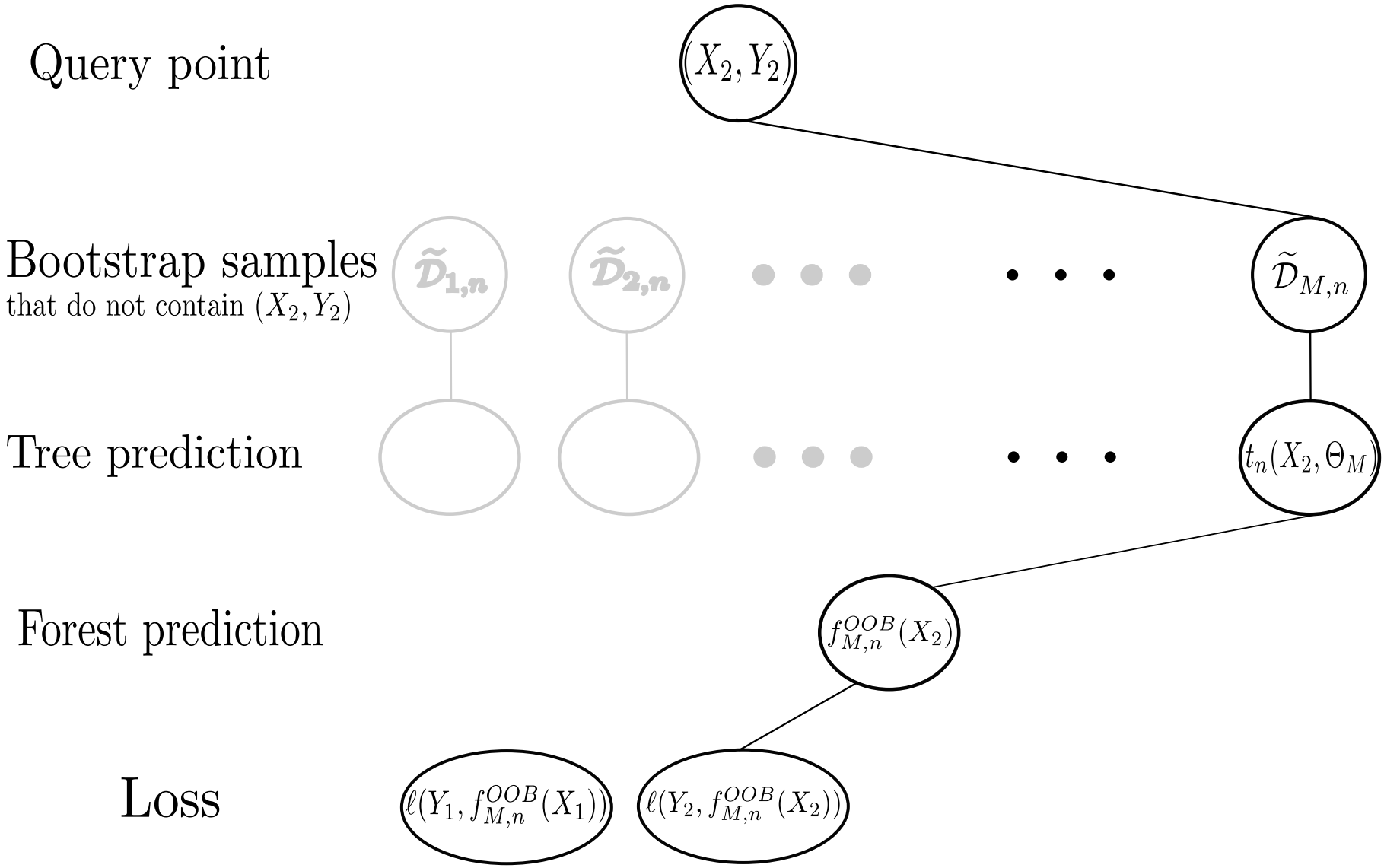
Out-of-bag procedure



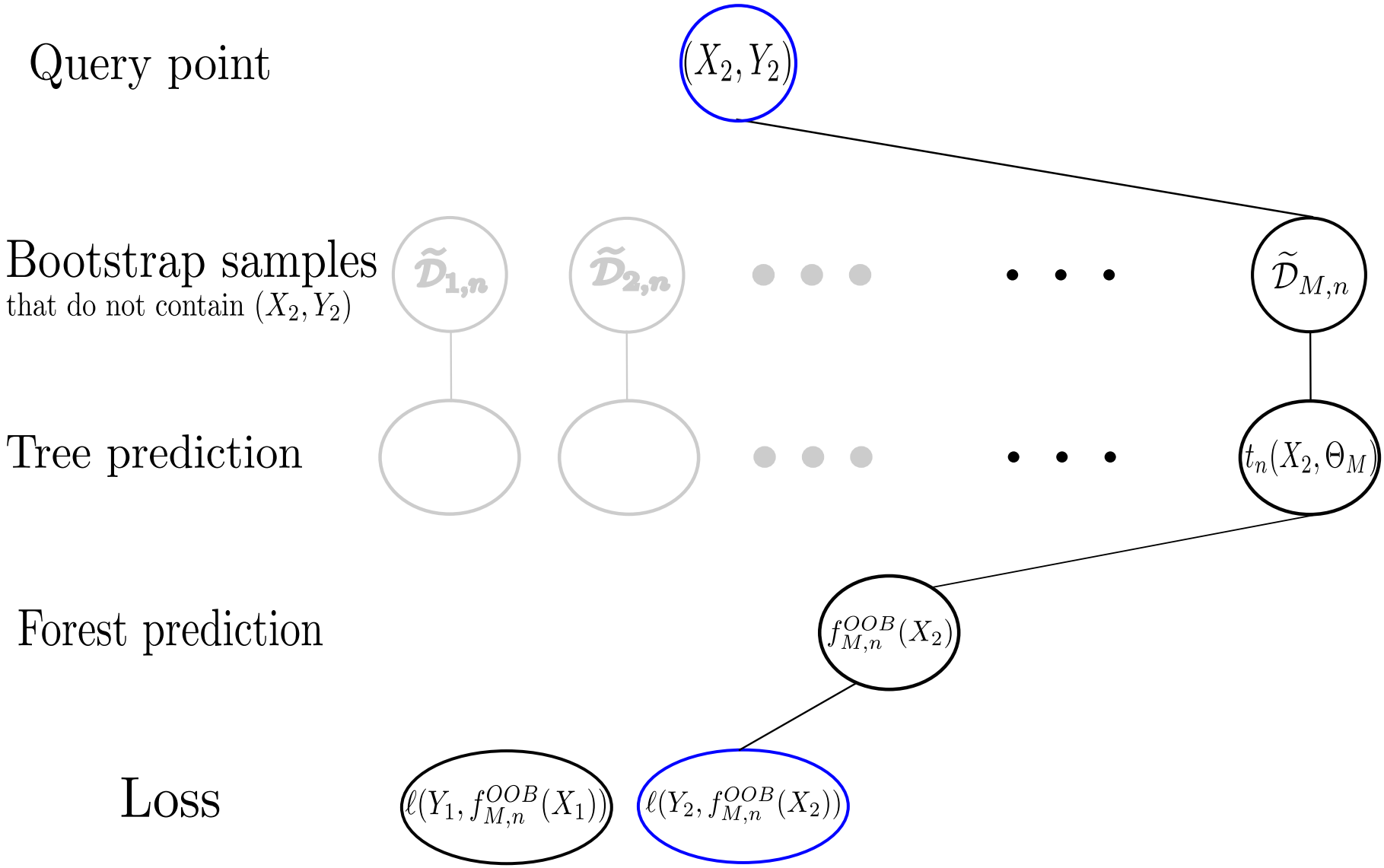
Out-of-bag procedure



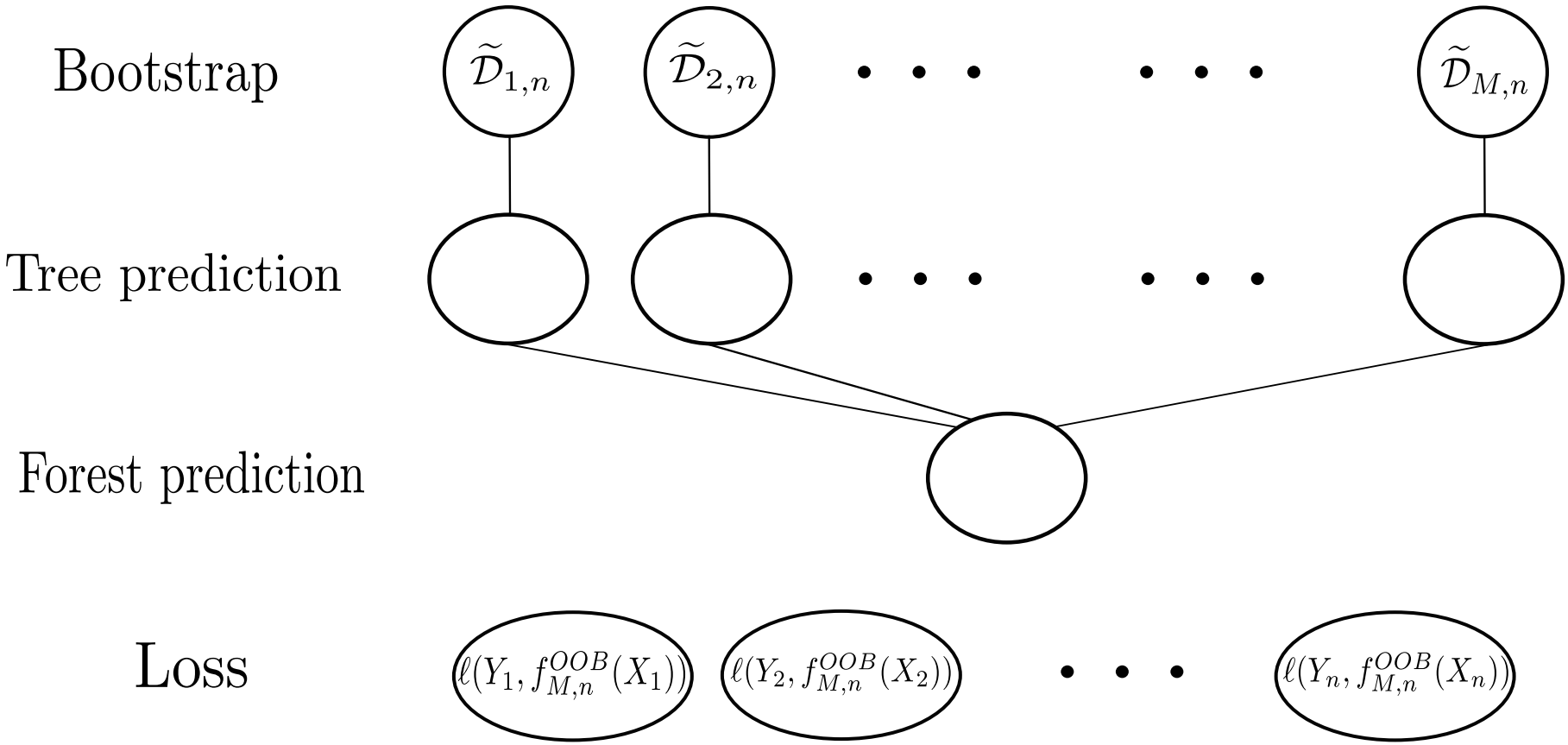
Out-of-bag procedure



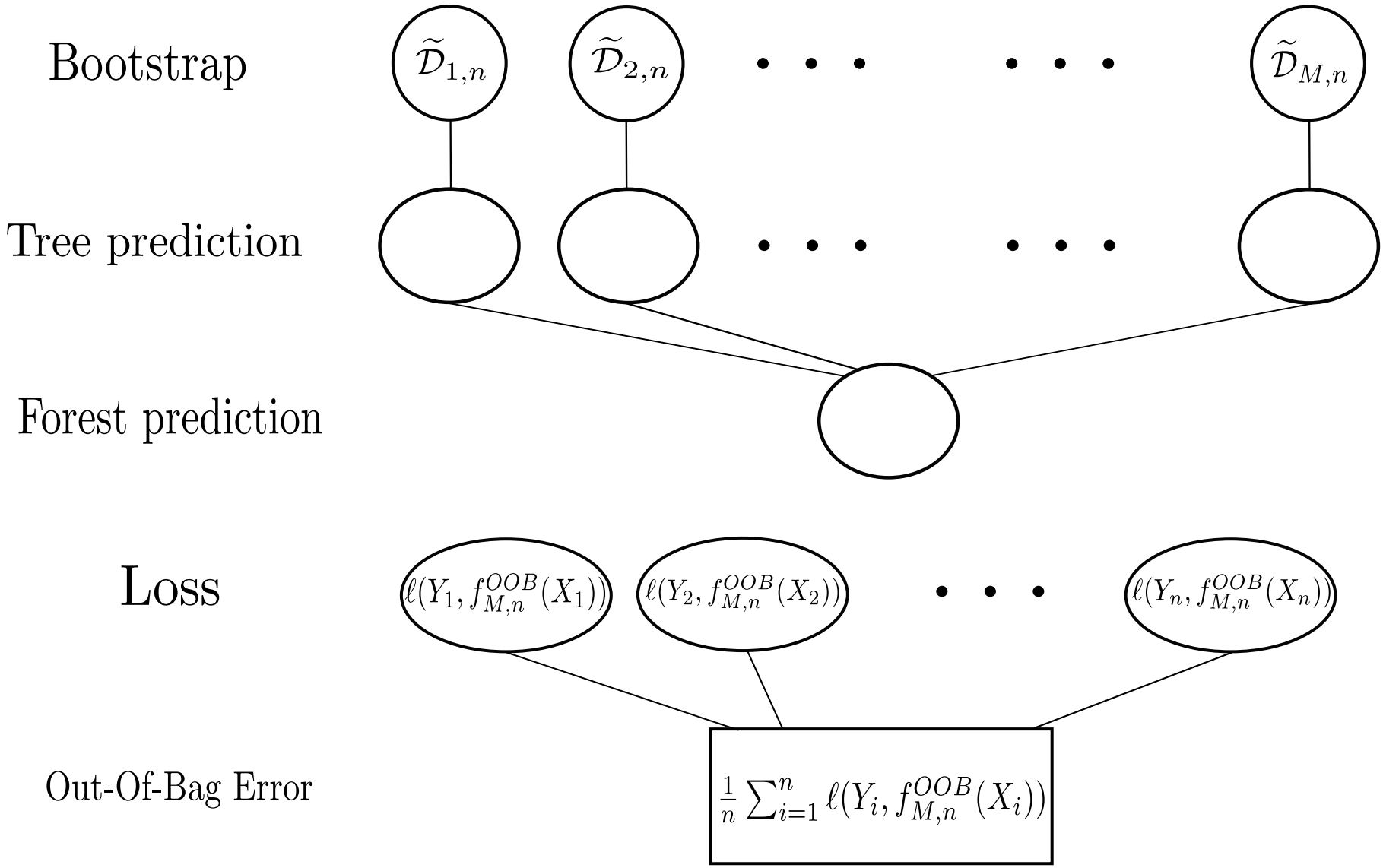
Out-of-bag procedure



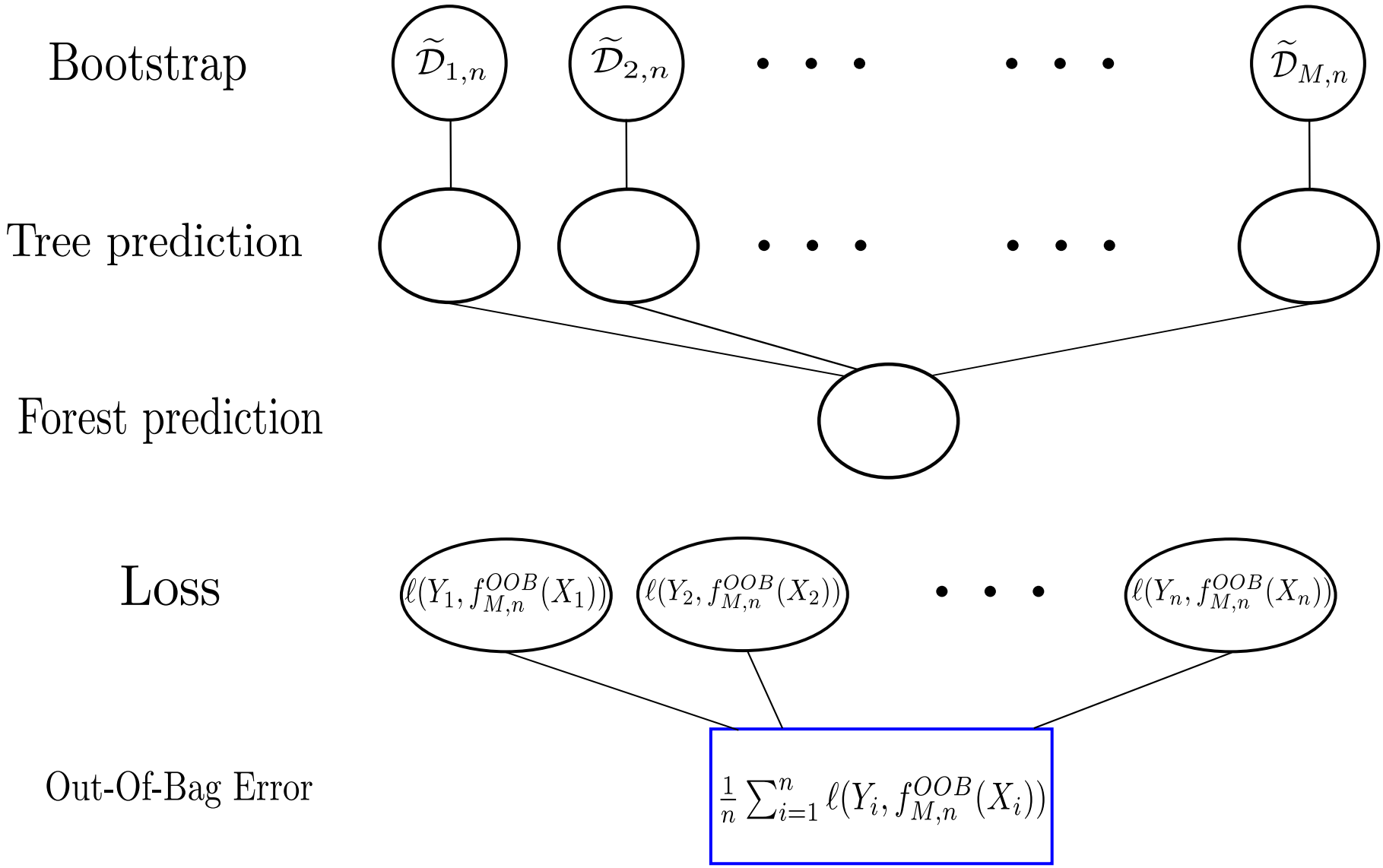
Out-of-bag procedure



Out-of-bag procedure



Out-of-bag procedure



Out-of-bag (detailed procedure)

Consider that a forest has been trained on the data set $\mathcal{D}_n$.

- For each observation $i \in \{1, \ldots, n\}$,
  - Consider the bootstrap samples that do not contain this observation, that is the set $\Lambda_{i,n} = \{m, (X_i, Y_i) \notin \widetilde{\mathcal{D}}_{m,n}\}$
  - For the trees that are not built using observation i, compute the prediction at X_i and aggregate them as

$$f_{M,n}^{(OOB)}(X_i) = \frac{1}{|\Lambda_{i,n}|} \sum_{m \in \Lambda_{n,i}} t_n(X_i, \Theta_m) \mathbb{1}_{|\Lambda_{n,i}|>0} \quad \text{in regression,}$$

$$= \underset{k \in \{1,\ldots,K\}}{\text{argmax}} \sum_{\ell \in \Lambda_{n,i}} \mathbb{1}_{t_n(X_i,\Theta_\ell)=k} \quad \text{in classification.}$$

  - Compute the associated loss

$$\ell(f_{M,n}^{(OOB)}(X_i), Y_i) = (f_{M,n}^{(OOB)}(X_i) - Y_i)^2 \quad \text{in regression,}$$

$$= \ell(f_{M,n}^{(OOB)}(X_i), Y_i) = \mathbb{1}_{f_{M,n}^{(OOB)}(X_i) \neq Y_i} \text{in classification.}$$

- Compute the Out-of-bag error by averaging the losses over all observations, that is

$$R_n^{OOB} = \frac{1}{n} \sum_{i=1}^{n} \ell(f_{M,n}^{(OOB)}(X_i), Y_i)$$

Outline

Variable importance via random forests

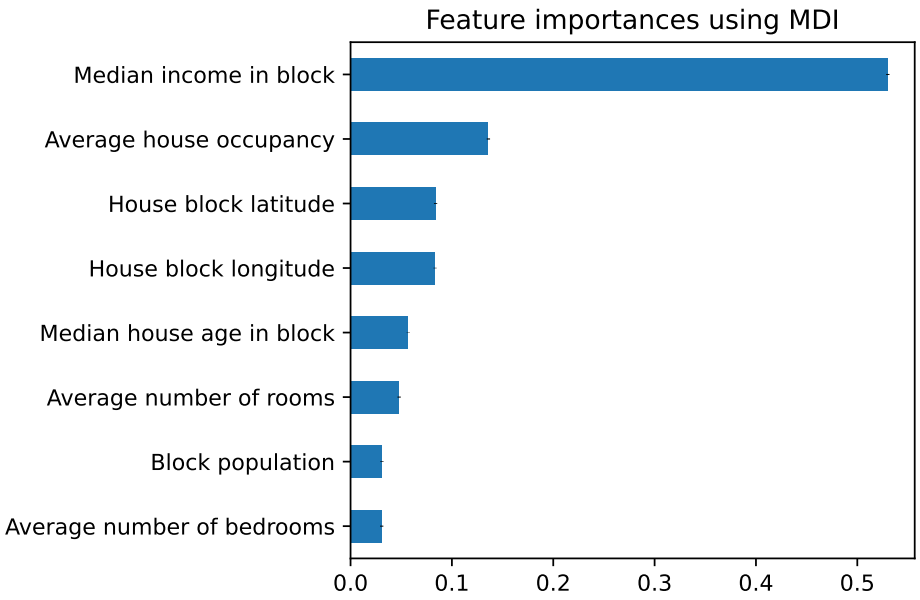


Figure: One of the two variable importance measure, Mean Decrease in Impurity (MDI) computed on the California housing data set.

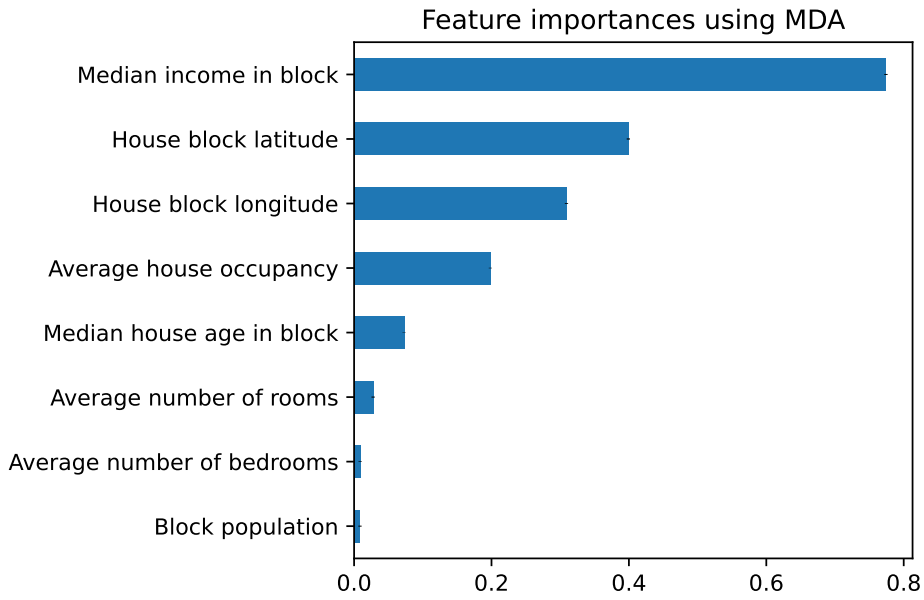
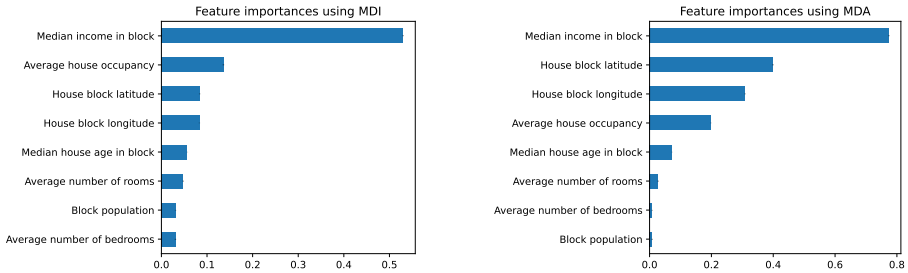


Figure: One of the two variable importance measure, Mean Decrease in Accuracy (MDA) computed on the California housing data set.

Variable importance via random forests



- Going beyond prediction to understand the black-box model
- Finding the input variables that are the most "linked" to the output
- Here the variable ranking is not exactly the same across these two different measures.

Variable importance - to what aim?

One single good variable importance measure does not exist. It always depend on what it is used for.

A simple example. Assume that $X \in \mathbb{R}^{10}$, $Y \in \mathbb{R}$ and $Y = X_1$ with $X_1 = g(X_2, \ldots, X_{10})$ for some function g.

- (Variable selection) If one is interested in finding the smallest set of variables leading to good predictive performance, the associated variable importance should be large for X_1 and null for $X_2, \ldots, X_{10}$.
- (Link identification) If one is interested in finding all variables linked to the output, the associated variable importance should be large for $X_1, \ldots, X_d$.

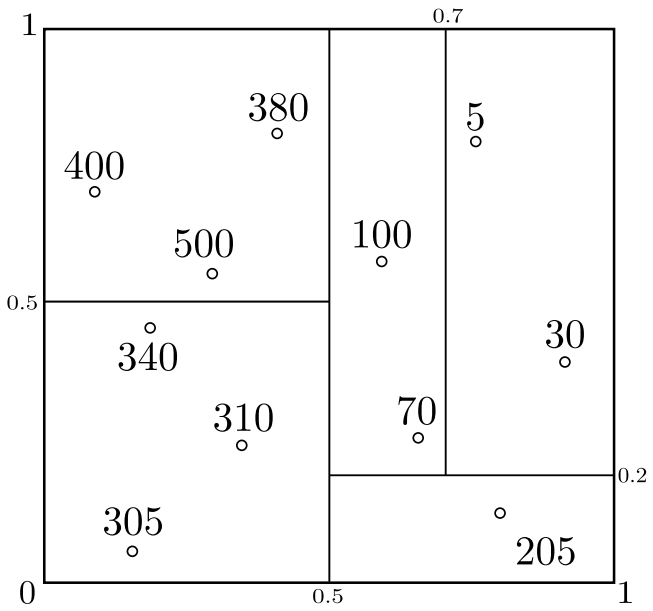
The quality of a variable importance measure depends on its final use (variable selection or link identification).

Variable importance in random forests

Two different measures often computed with random forests:

- Mean Decrease Impurity (MDI) (Breiman, 2002)
  - Tailored for decision tree methods
  - Use the decrease in impurity in each node to compute an aggregated variable importance
- Mean Decrease Accuracy (MDA) (also called *permutation importance*, see Breiman, 2001)
  - Can be used with any supervised learning algorithm (not tree specific)
  - Permute the values of a given feature in the test set and compare the resulting decrease in predictive performance.

Mean Decrease in impurity

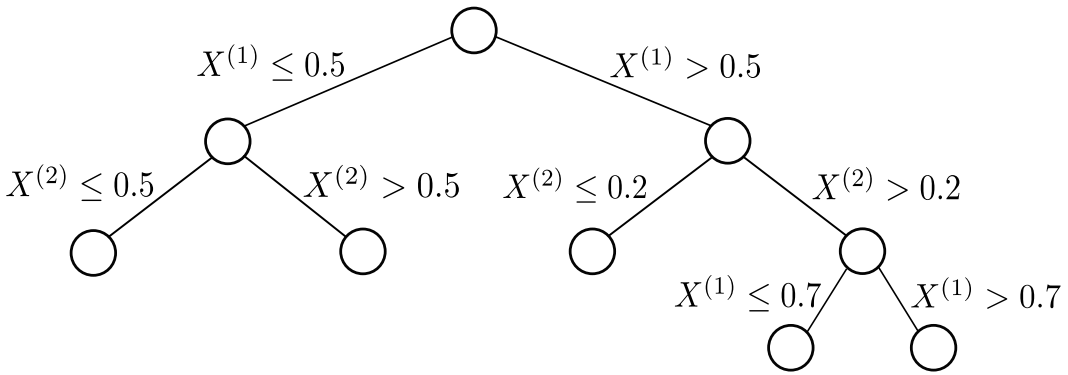


$k = 0$

$k = 1$

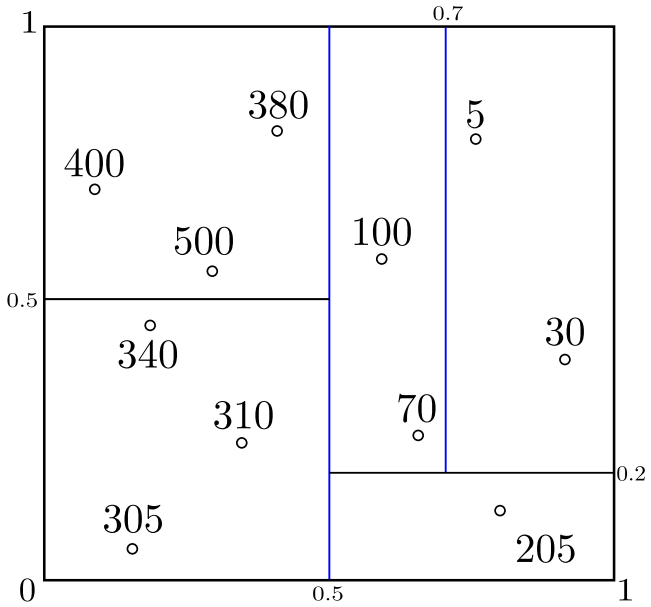
$k = 2$

$k = 3$



For this given trained tree $\mathcal{T}$, we want to evaluate the MDI of $X^{(1)}$.

Mean Decrease in impurity

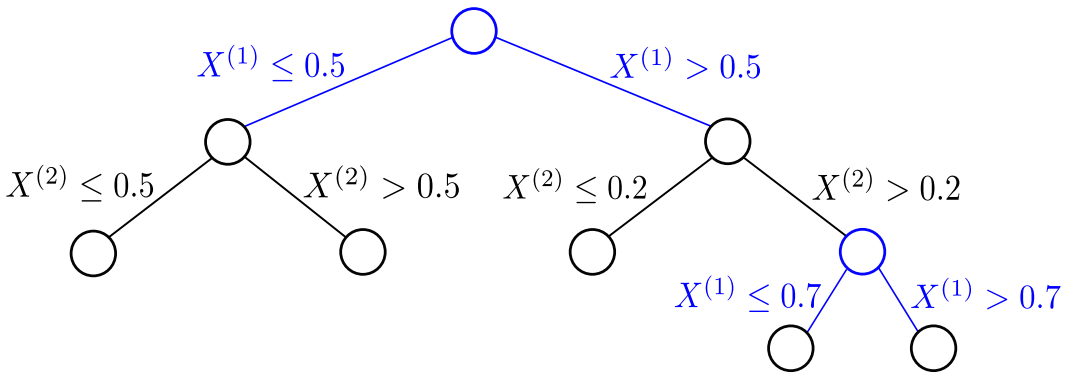


$k = 0$

$k = 1$

$k = 2$

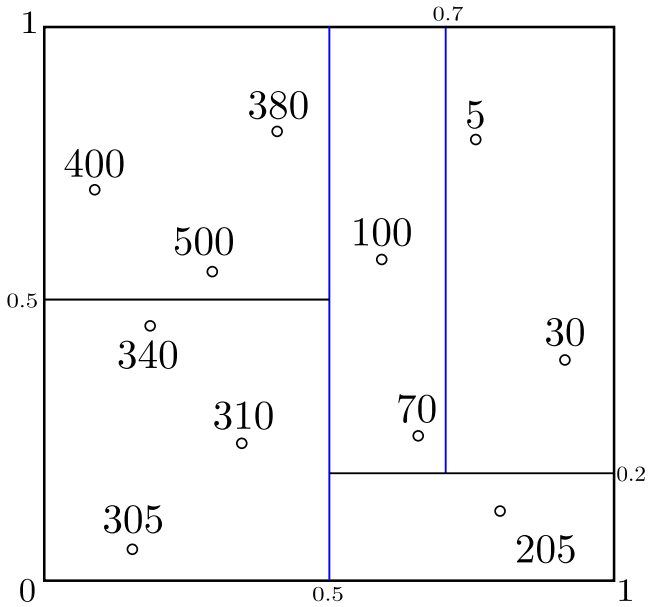
$k = 3$



For this given trained tree $\mathcal{T}$, we want to evaluate the MDI of $X^{(1)}$. We proceed as follows:

- Identify all splits that involve variable $X^{(1)}$

Mean Decrease in impurity

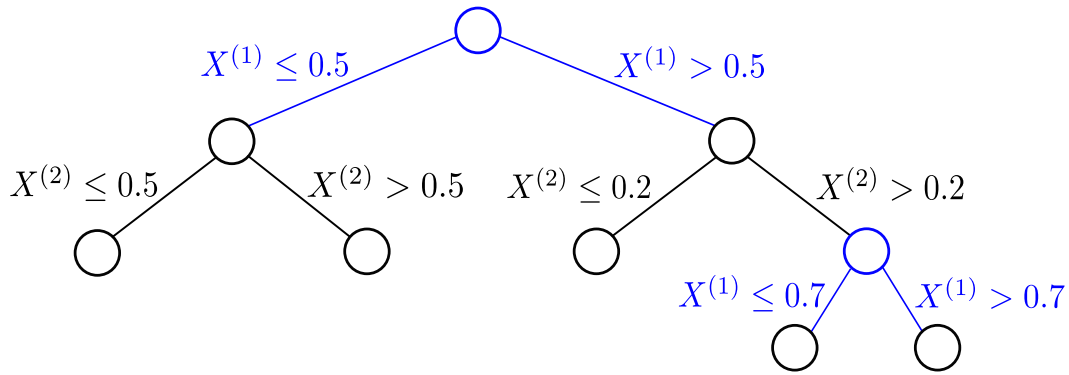


$k = 0$

$k = 1$

$k = 2$

$k = 3$



For this given trained tree $\mathcal{T}$, we want to evaluate the MDI of $X^{(1)}$. We proceed as follows:

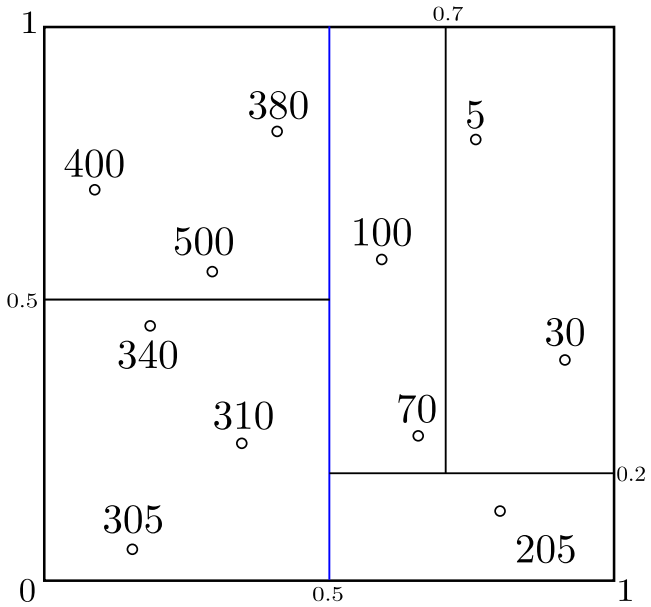
- Identify all splits that involve variable $X^{(1)}$
- For each split, compute the decrease in impurity between the parent node A and the two resulting nodes A_L and A_R:

$$\Delta Imp_n(A) = Imp_n(A) - p_{L,n} Imp_n(A_L) - p_{R,n} Imp_n(A_R),$$

where $p_{L,n}$ (resp. $p_{R,n}$) is the fraction of observations in A that fall into A_L (resp. A_R). For example,

$$Imp_{V,n}(A) = \mathbb{V}_n[Y|X \in A].$$

Mean Decrease in impurity

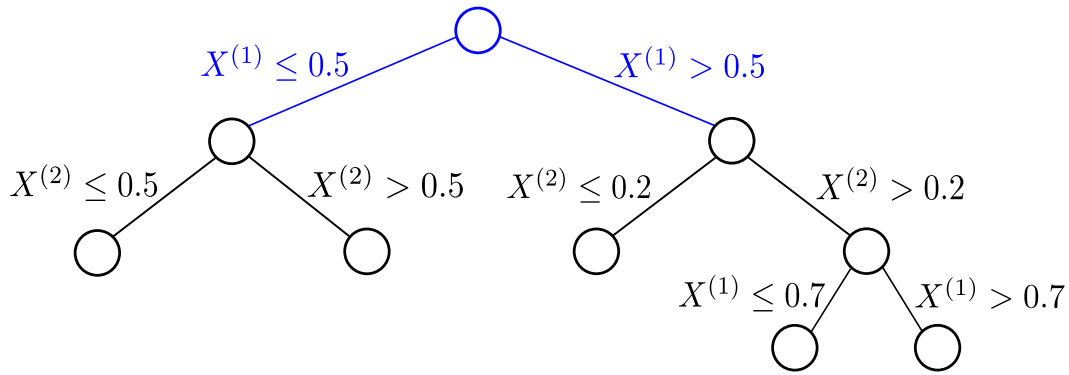


$k = 0$

$k = 1$

$k = 2$

$k = 3$



For this given trained tree $\mathcal{T}$, we want to evaluate the MDI of $X^{(1)}$. We proceed as follows:

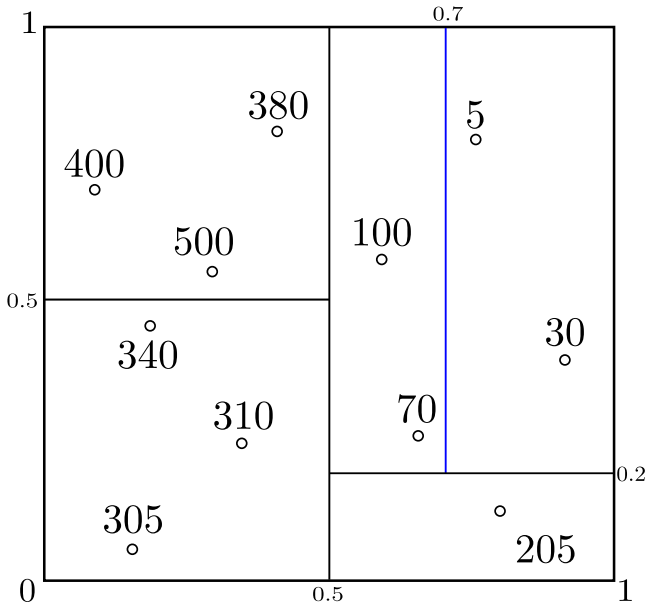
- Identify all splits that involve variable $X^{(1)}$
- For each split, compute the decrease in impurity between the parent node A and the two resulting nodes A_L and A_R:

$$\Delta Imp_n(A) = Imp_n(A) - p_{L,n} Imp_n(A_L) - p_{R,n} Imp_n(A_R),$$

where $p_{L,n}$ (resp. $p_{R,n}$) is the fraction of observations in A that fall into A_L (resp. A_R). For example,

$$Imp_{V,n}(A) = \mathbb{V}_n[Y|X \in A].$$

Mean Decrease in impurity

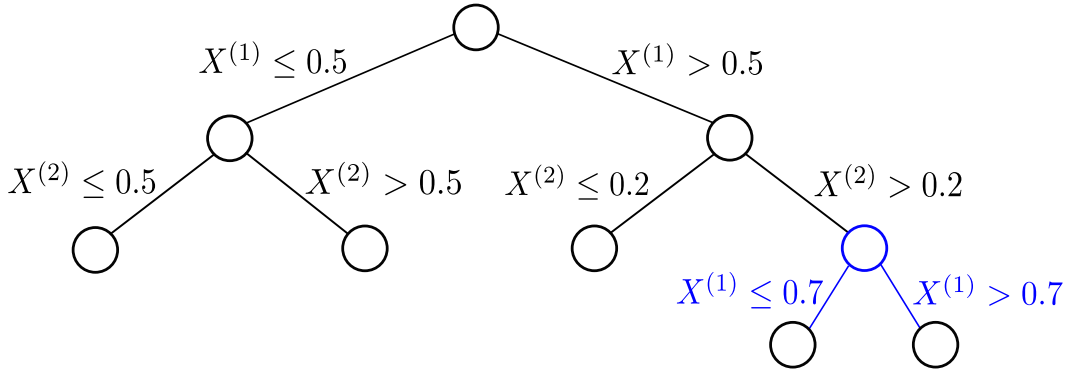


$k = 0$

$k = 1$

$k = 2$

$k = 3$



For this given trained tree $\mathcal{T}$, we want to evaluate the MDI of $X^{(1)}$. We proceed as follows:

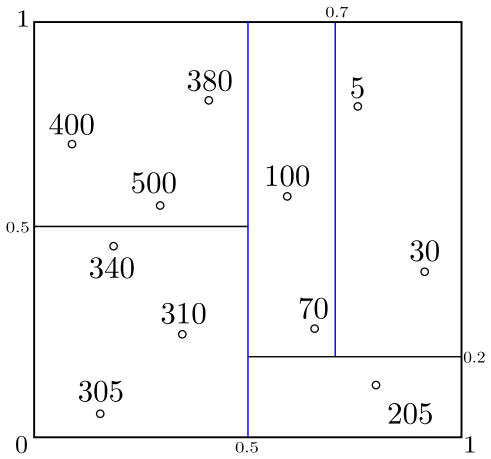
- Identify all splits that involve variable $X^{(1)}$
- For each split, compute the decrease in impurity between the parent node A and the two resulting nodes A_L and A_R:

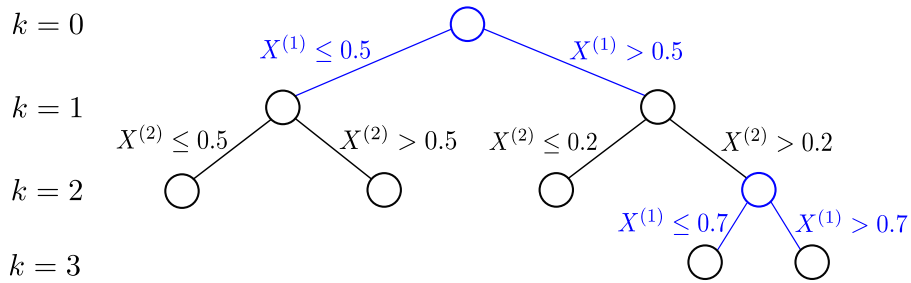
$$\Delta Imp_n(A) = Imp_n(A) - p_{L,n} Imp_n(A_L) - p_{R,n} Imp_n(A_R),$$

where $p_{L,n}$ (resp. $p_{R,n}$) is the fraction of observations in A that fall into A_L (resp. A_R). For example,

$$Imp_{V,n}(A) = \mathbb{V}_n[Y|X \in A].$$

Mean Decrease in impurity





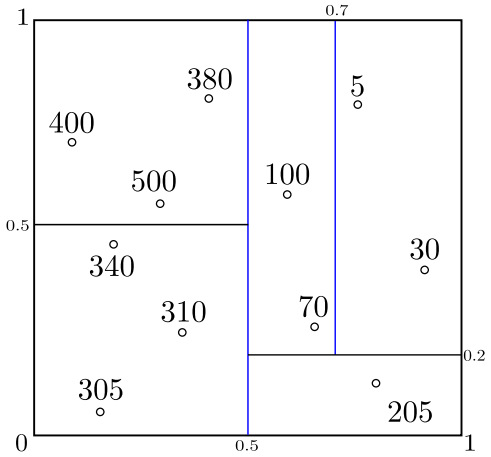
For this given trained tree $\mathcal{T}$, we want to evaluate the MDI of $X^{(1)}$. We proceed as follows:

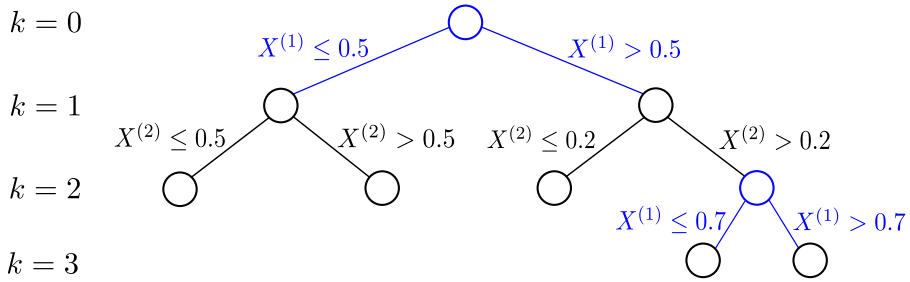
- Identify all splits that involve variable $X^{(1)}$
- For each split, compute the decrease in impurity $\Delta Imp_n(A)$ between the parent node A and the two resulting nodes A_L and A_R
- The MDI of $X^{(1)}$ computed via this tree $\mathcal{T}$ is

$$\widehat{\mathrm{MDI}}_{\mathcal{T}}(X^{(j)}) = \sum_{\substack{A \in \mathcal{T} \\ j_{n,A}=1}} p_{n,A}\ \Delta Imp_n(A), \tag{1}$$

where the sum ranges over all cells A in $\mathcal{T}$ that are split along variable j and $p_{A,n}$ is the fraction of observations falling into A

Mean Decrease in impurity





For this given trained tree $\mathcal{T}$, we want to evaluate the MDI of $X^{(1)}$. We proceed as follows:

- Identify all splits that involve variable $X^{(1)}$
- For each split, compute the decrease in impurity $\Delta Imp_n(A)$ between the parent node A and the two resulting nodes A_L and A_R
- The MDI of $X^{(1)}$ computed via this tree $\mathcal{T}$ is

$$\widehat{\mathrm{MDI}}_{\mathcal{T}}(X^{(j)}) = \sum_{\substack{A \in \mathcal{T} \\ j_{n,A}=1}} p_{n,A}\ \Delta Imp_n(A) \tag{1}$$

- The MDI of $X^{(1)}$ output by a forest is the average of the MDI of $X^{(1)}$ of each tree.

Mean Decrease in Impurity

Pros

- Easily accessible via scikit-learn as the attribute `feature_importances_` of a RandomForest object
- No extra computations needed
- Adapted to the tree building process / the predictor

Mean Decrease in Impurity

Cons

- biased towards variables with many categories (see, e.g., Strobl et al., 2007; Nicodemus, 2011), variables that possess high-category frequency (Nicodemus, 2011; Boulesteix et al., 2011), biased in presence of correlated features (Nicodemus and Malley, 2009)
- Bias related to in-sample estimation (Li et al., 2019; Zhou and Hooker, 2021) - Same observations are used to build the tree and estimate the MDI
- Bias related to fully-grown tree
- No information about the quantity it is supposed to estimate!

MDA illustration

MDA principle: decrease of accuracy of the forest when a variable is noised up

$X^{(1)}$	$X^{(2)}$	$\dots$	$X^{(j)}$	$\dots$	$X^{(p)}$	Y
2.1	4.3	$\dots$	0.1	$\dots$	2.6	2.3
1.7	4.1	$\dots$	9.2	$\dots$	3.8	0.4
3.4	9.2	$\dots$	3.2	$\dots$	3.6	10.2
5.6	1.2	$\dots$	8.2	$\dots$	4.2	9.1
8.9	6.8	$\dots$	6.7	$\dots$	2.9	4.5

Table: Example of the permutation of a dataset $\mathcal{D}_n$ for $n = 5$.

MDA illustration

MDA principle: decrease of accuracy of the forest when a variable is noised up

$X^{(1)}$	$X^{(2)}$	$\dots$	$X^{(j)}$	$\dots$	$X^{(p)}$	Y
2.1	4.3	$\dots$	0.1	$\dots$	2.6	2.3
1.7	4.1	$\dots$	9.2	$\dots$	3.8	0.4
3.4	9.2	$\dots$	3.2	$\dots$	3.6	10.2
5.6	1.2	$\dots$	8.2	$\dots$	4.2	9.1
8.9	6.8	$\dots$	6.7	$\dots$	2.9	4.5

Table: Example of the permutation of a dataset $\mathcal{D}_n$ for $n = 5$.

MDA illustration

MDA principle: decrease of accuracy of the forest when a variable is noised up

$X^{(1)}$	$X^{(2)}$	...	$X^{(j)}$	...	$X^{(p)}$	Y
2.1	4.3	...	0.1	...	2.6	2.3
1.7	4.1	...	9.2	...	3.8	0.4
3.4	9.2	...	3.2	...	3.6	10.2
5.6	1.2	...	8.2	...	4.2	9.1
8.9	6.8	...	6.7	...	2.9	4.5

$X^{(1)}$	$X^{(2)}$	...	$X^{(j)}$	...	$X^{(p)}$	Y
2.1	4.3	...	6.7	...	2.6	2.3
1.7	4.1	...	3.2	...	3.8	0.4
3.4	9.2	...	9.2	...	3.6	10.2
5.6	1.2	...	0.1	...	4.2	9.1
8.9	6.8	...	8.2	...	2.9	4.5

Table: Example of the permutation of a dataset $\mathcal{D}_n$ for $n = 5$.

MDA illustration

MDA principle: decrease of accuracy of the forest when a variable is noised up

$X^{(1)}$	$X^{(2)}$	...	$X^{(j)}$	...	$X^{(p)}$	Y
2.1	4.3	...	0.1	...	2.6	2.3
1.7	4.1	...	9.2	...	3.8	0.4
3.4	9.2	...	3.2	...	3.6	10.2
5.6	1.2	...	8.2	...	4.2	9.1
8.9	6.8	...	6.7	...	2.9	4.5

$X^{(1)}$	$X^{(2)}$	...	$X^{(j)}$	...	$X^{(p)}$	Y
2.1	4.3	...	6.7	...	2.6	2.3
1.7	4.1	...	3.2	...	3.8	0.4
3.4	9.2	...	9.2	...	3.6	10.2
5.6	1.2	...	0.1	...	4.2	9.1
8.9	6.8	...	8.2	...	2.9	4.5

quadratic error $= 13.7$
quadratic error $= 16.4$

$$\mathrm{MDA}(X^{(j)}) = 16.4 - 13.7 = 2.7$$

MDA versions

The explained variance estimate of MDA algorithms differ across implementations

Train-Test MDA: train data to fit the forest, and test data for accuracy

MDA versions

The explained variance estimate of MDA algorithms differ across implementations

Train-Test MDA: train data to fit the forest, and test data for accuracy

Out-of-bag (OOB) samples: $\mathcal{D}_n$ is bootstrap prior to the construction of each tree, leaving aside a portion of $\mathcal{D}_n$, which is not involved in the tree growing and defines the "out-of-bag" sample.

$X^{(1)}$	$X^{(2)}$	$\ldots$	$X^{(j)}$	$\ldots$	$X^{(p)}$	Y
2.1	4.3	$\ldots$	0.1	$\ldots$	2.6	2.3
1.7	4.1	$\ldots$	9.2	$\ldots$	3.8	0.4
3.4	9.2	$\ldots$	3.2	$\ldots$	3.6	10.2
5.6	1.2	$\ldots$	8.2	$\ldots$	4.2	9.1
8.9	6.8	$\ldots$	6.7	$\ldots$	2.9	4.5

Selected samples: $\Theta_\ell^{(s)} = \{1, 3, 4\}$

MDA versions

The explained variance estimate of MDA algorithms differ across implementations

Train-Test MDA: train data to fit the forest, and test data for accuracy

Out-of-bag (OOB) samples: $\mathcal{D}_n$ is bootstrap prior to the construction of each tree, leaving aside a portion of $\mathcal{D}_n$, which is not involved in the tree growing and defines the "out-of-bag" sample.

$X^{(1)}$	$X^{(2)}$	$\ldots$	$X^{(j)}$	$\ldots$	$X^{(p)}$	Y
2.1	4.3	$\ldots$	0.1	$\ldots$	2.6	2.3
1.7	4.1	$\ldots$	9.2	$\ldots$	3.8	0.4
3.4	9.2	$\ldots$	3.2	$\ldots$	3.6	10.2
5.6	1.2	$\ldots$	8.2	$\ldots$	4.2	9.1
8.9	6.8	$\ldots$	6.7	$\ldots$	2.9	4.5

OOB samples: $\{1, \ldots, n\} \setminus \Theta_\ell^{(S)} = \{2, 5\}$

Mean Decrease in Accuracy

Pros

- Can be applied to any machine learning algorithm via the function `permutation-importance` in scikit-learn
- Fast to compute (no need to retrain a forest)

Cons

- Biased in presence of correlation
- The quantity to which it converges is not the correct for either of the two objectives (designing a small model with high predictivity or finding a large set of variables linked to the output)

Take-home message on variable importance

Do not use MDI or MDA!

We do not know what quantity they are targeting

Take-home message on variable importance

Some alternatives:

- MDI
  - Out-of-sample estimation (Li et al., 2019; Zhou and Hooker, 2021; Loecher, 2022) with code in python:

    `https://github.com/ZhengzeZhou/unbiased-feature-importance`

- MDA
  - Rerun the model without a given covariate (expensive). Work for any predictive model (Williamson et al., 2021)
  - Use the tree structure to remove a variable from the model without needing to rerun it (Bénard et al., 2022)

Take-home message on variable importance

Some alternatives:

- MDI
  - Out-of-sample estimation (Li et al., 2019; Zhou and Hooker, 2021; Loecher, 2022) with code in python:
    `https://github.com/ZhengzeZhou/unbiased-feature-importance`
- MDA
  - Rerun the model without a given covariate (expensive). Work for any predictive model (Williamson et al., 2021)
  - Use the tree structure to remove a variable from the model without needing to rerun it (Bénard et al., 2022)

Anyway, remember to check the predictive performance of a model: it it is low, the model is useless and variable importances are misleading.

Outline

Outline

What is Boosting?

Boosting:

- Combining weak predictors (classification/regression) in a sequential manner to obtain an aggregated predictor better than each individual predictor
- The predictor resulting from boosting is a weighted average of some weak predictors, resulting from a learning algorithm applied to a modified dataset.

What is Boosting?

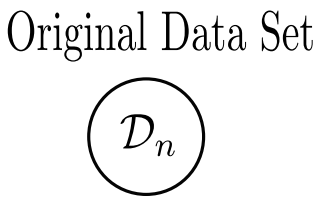
Boosting:

- Combining weak predictors (classification/regression) in a sequential manner to obtain an aggregated predictor better than each individual predictor
- The predictor resulting from boosting is a weighted average of some weak predictors, resulting from a learning algorithm applied to a modified dataset.

What do we need?

- A data set
  $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$
- A weak learning procedure
  - Decision tree (CART) $\rightarrow$ Tree Boosting
  - Linear models
  - ...
- A loss

Boosting in a scheme



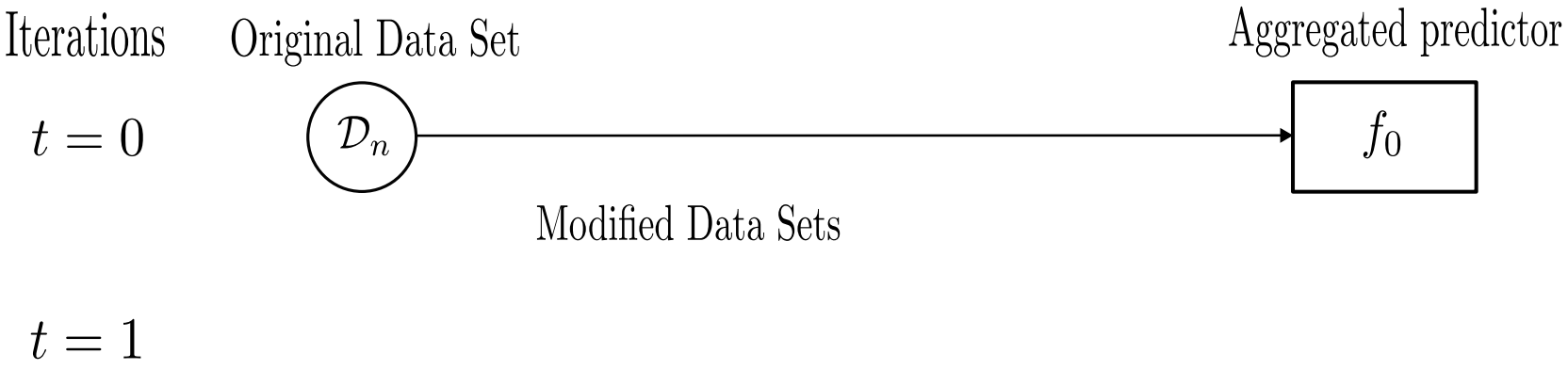
Boosting in a scheme

Iterations	Original Data Set
$t = 0$	$\mathcal{D}_n$

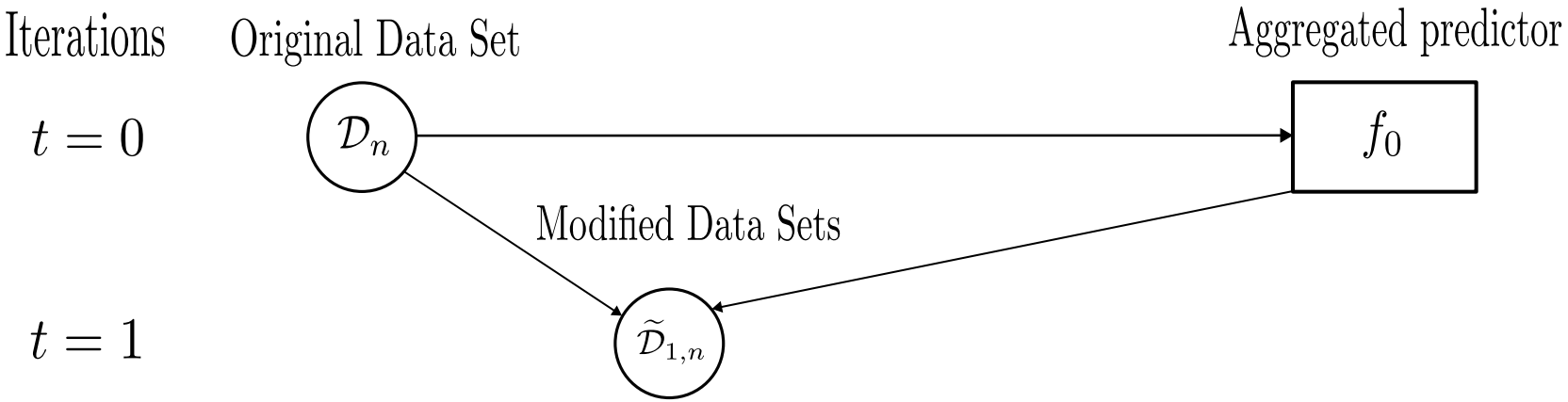
Boosting in a scheme

Boosting in a scheme

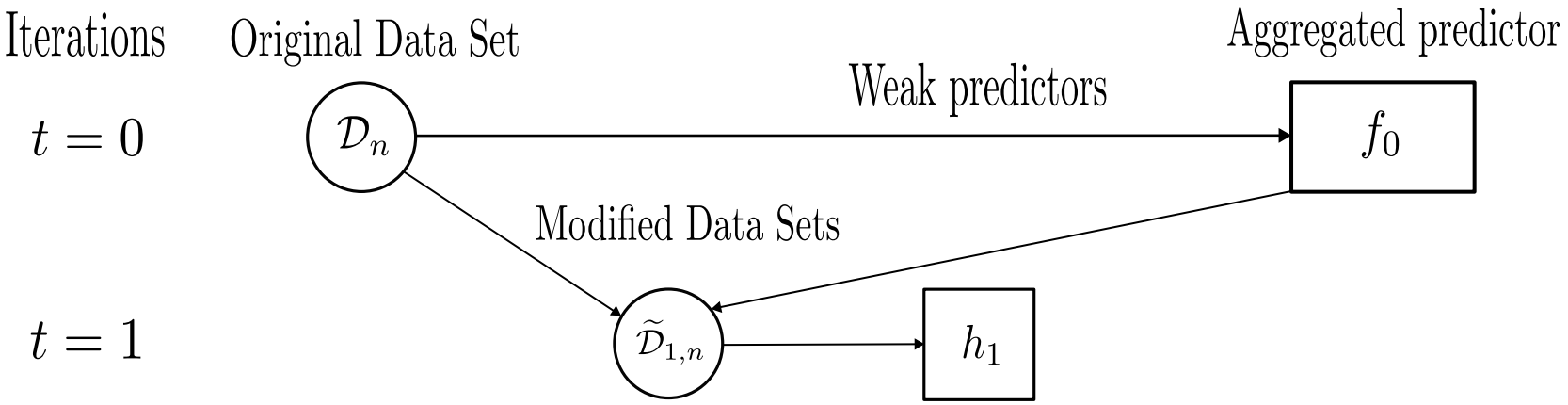
Boosting in a scheme



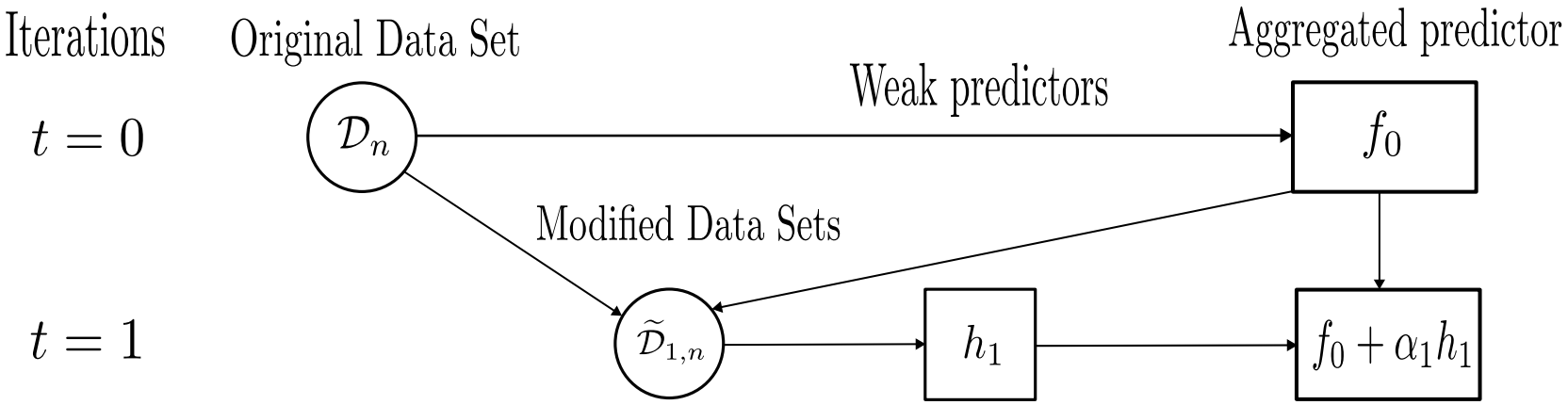
Boosting in a scheme



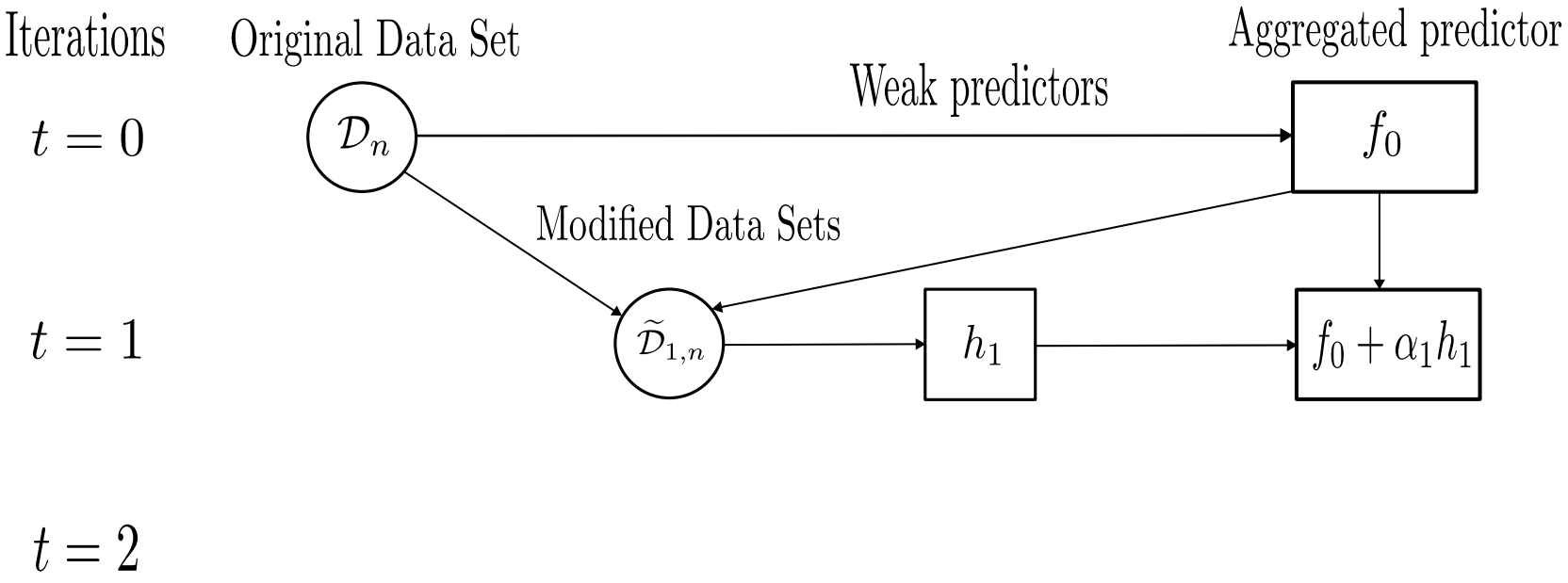
Boosting in a scheme



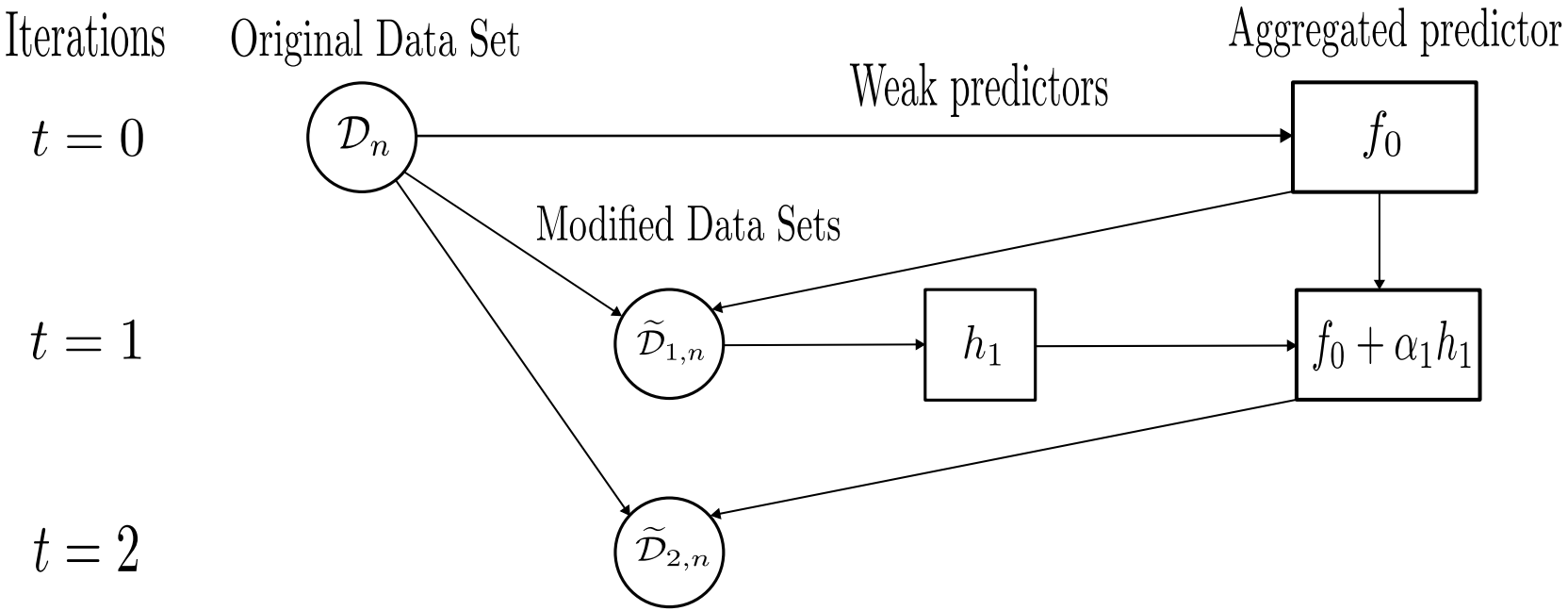
Boosting in a scheme



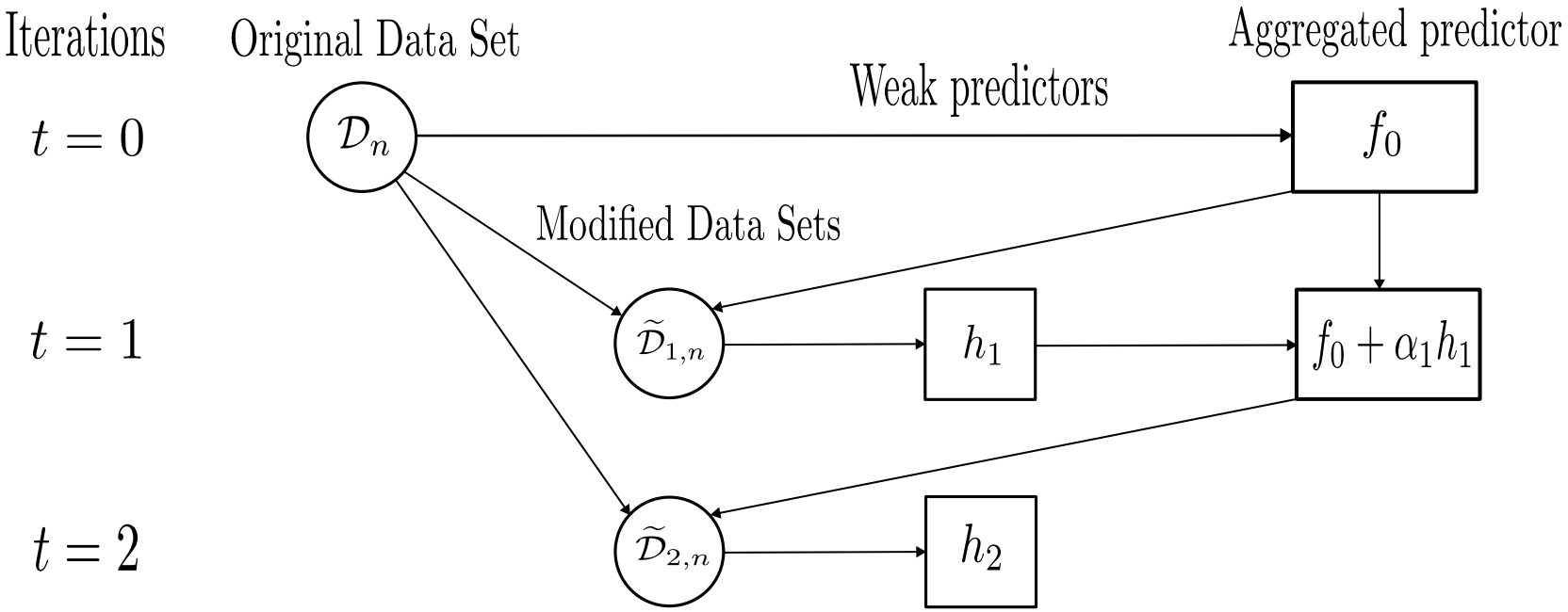
Boosting in a scheme



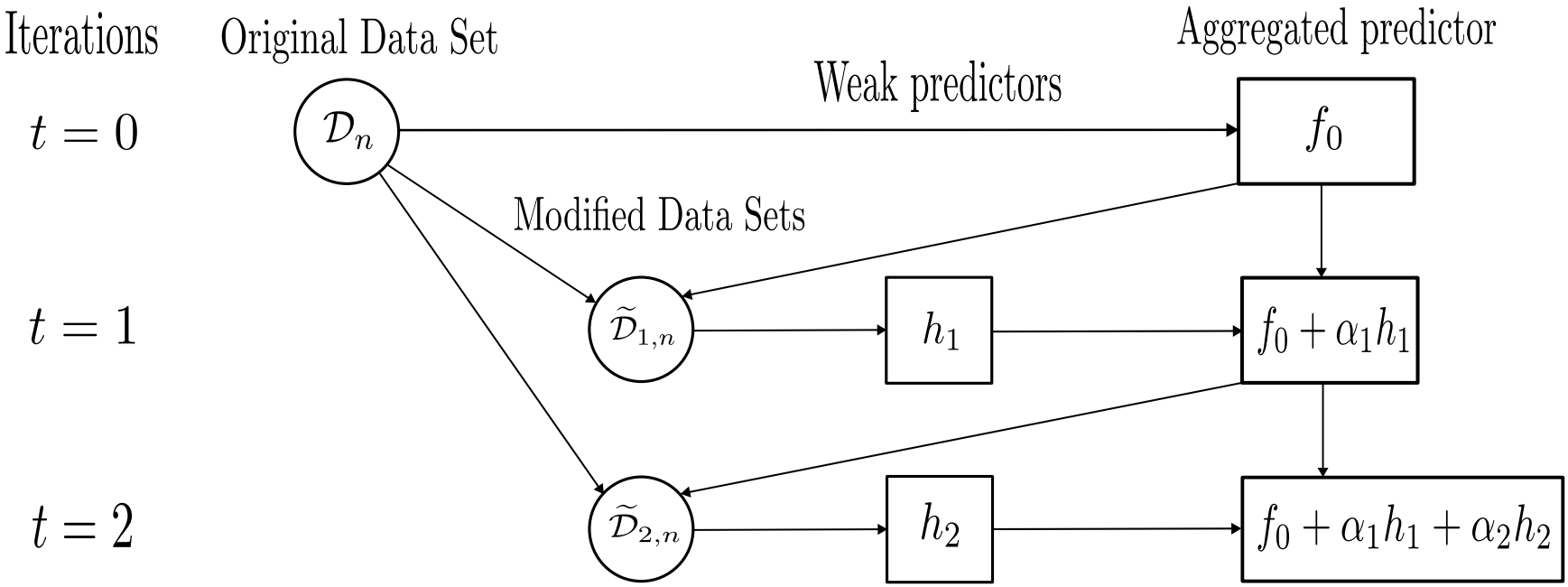
Boosting in a scheme



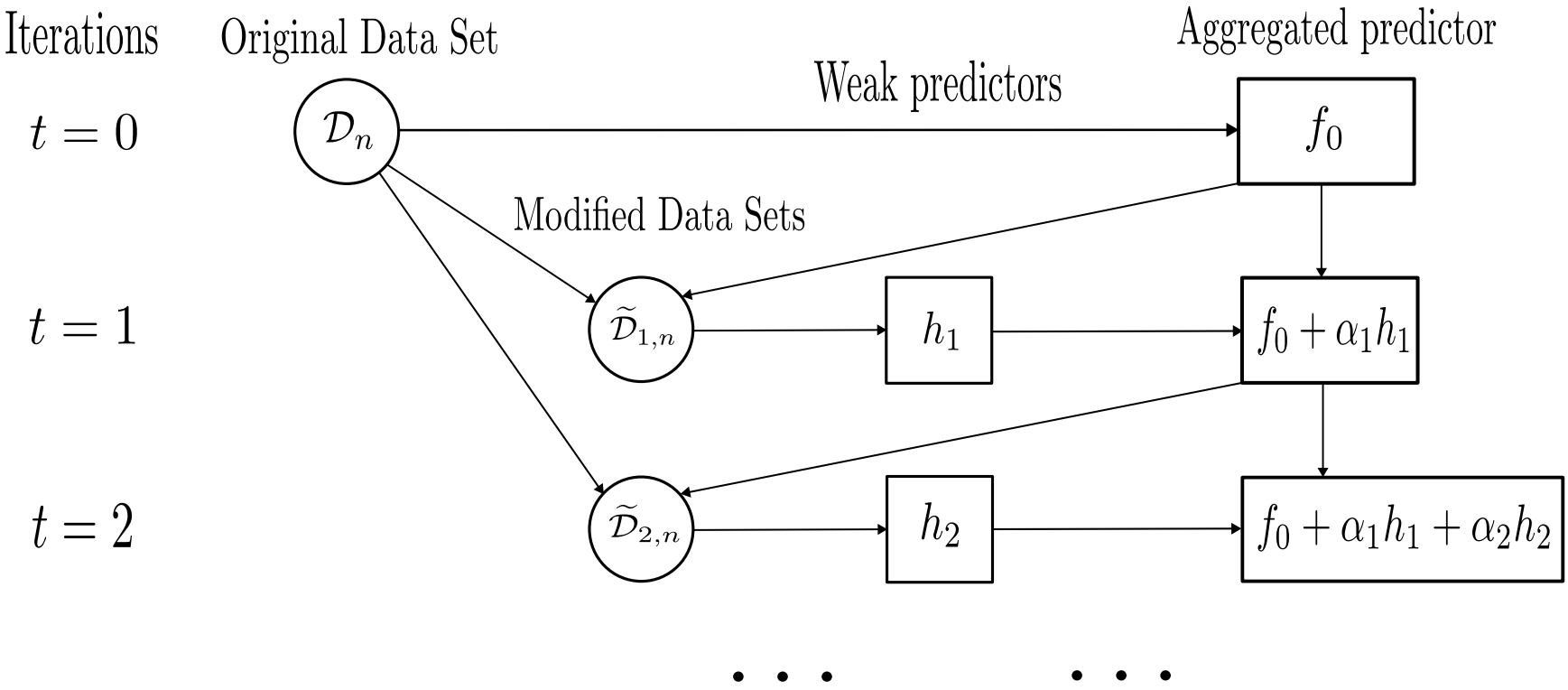
Boosting in a scheme



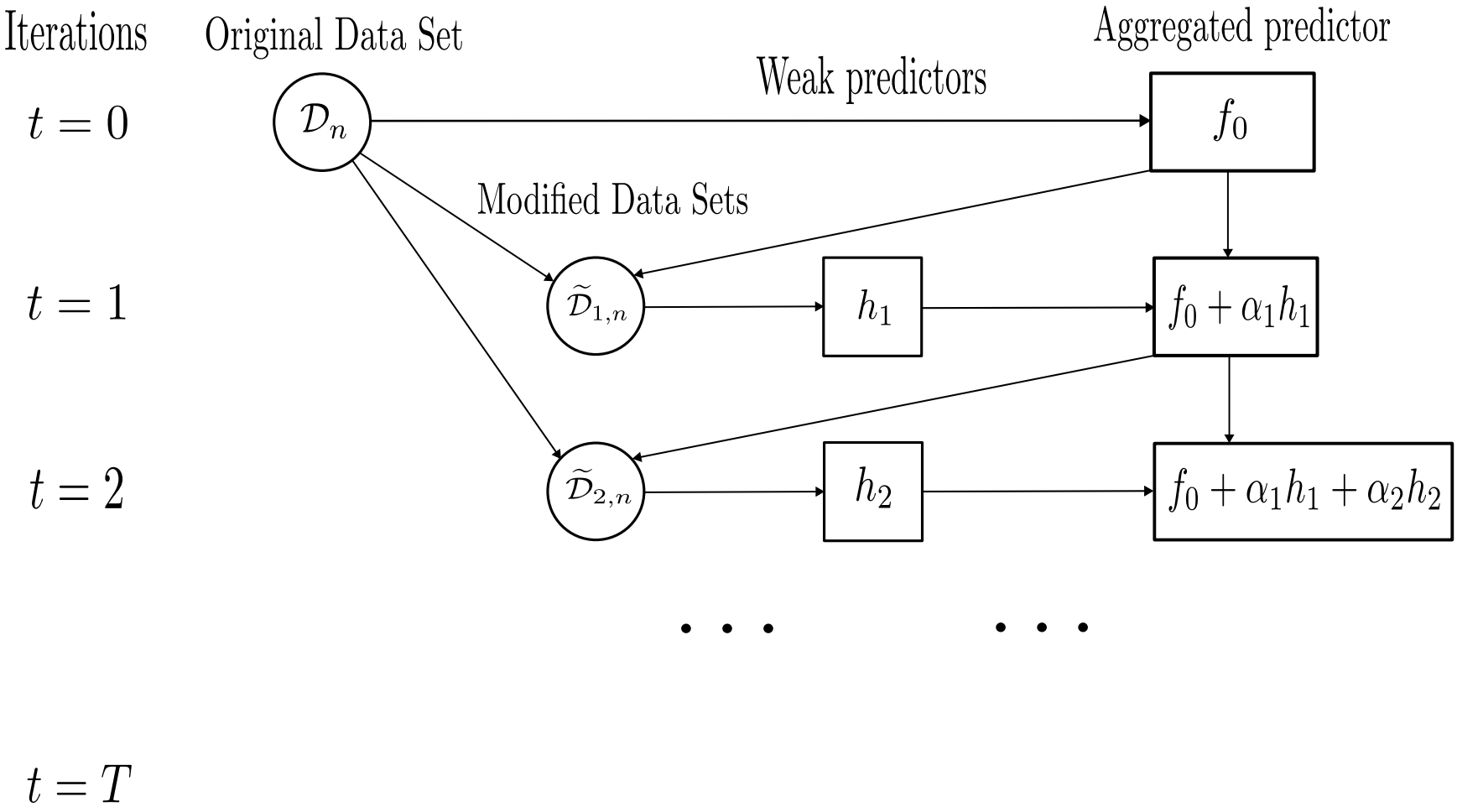
Boosting in a scheme



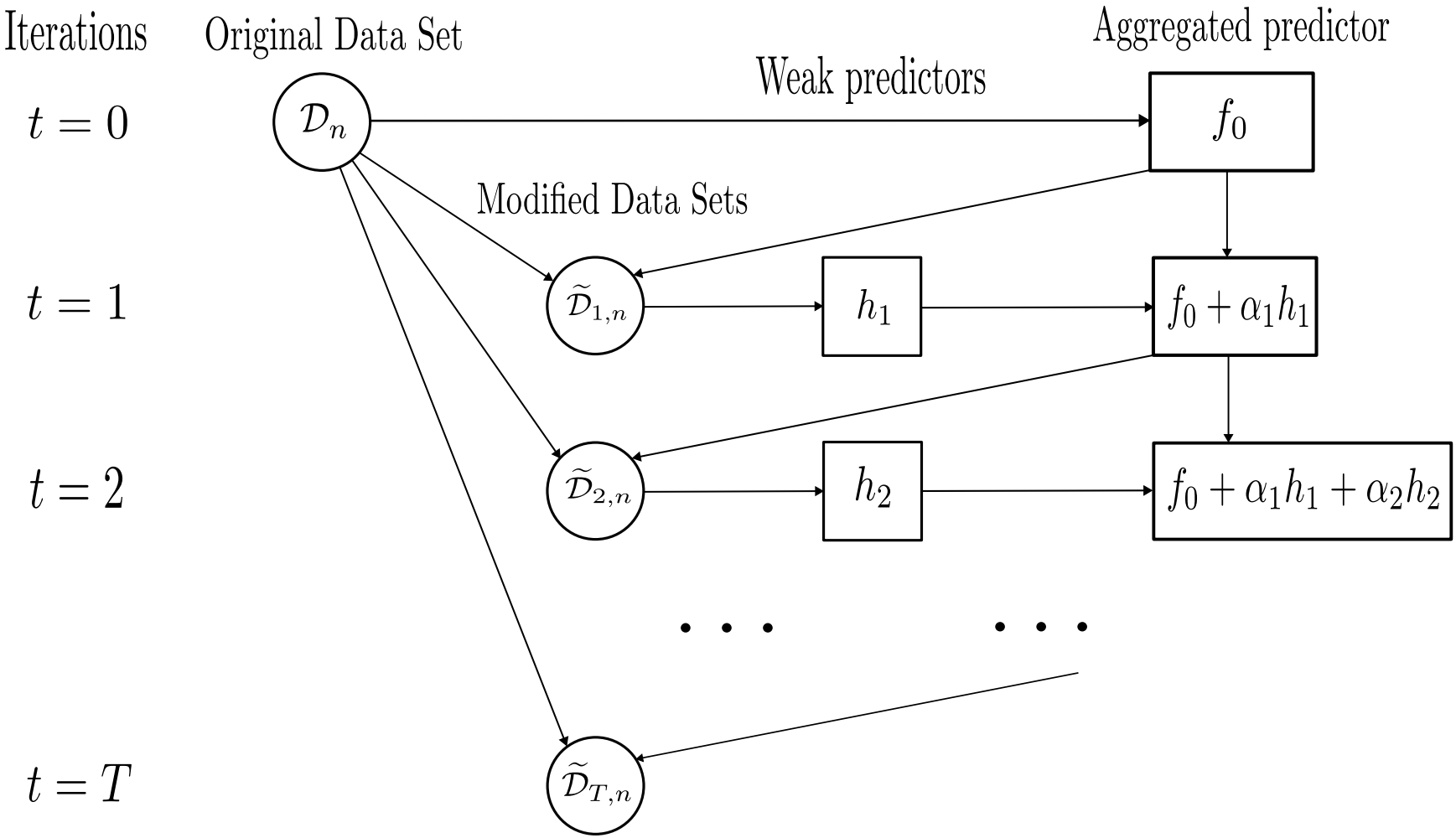
Boosting in a scheme



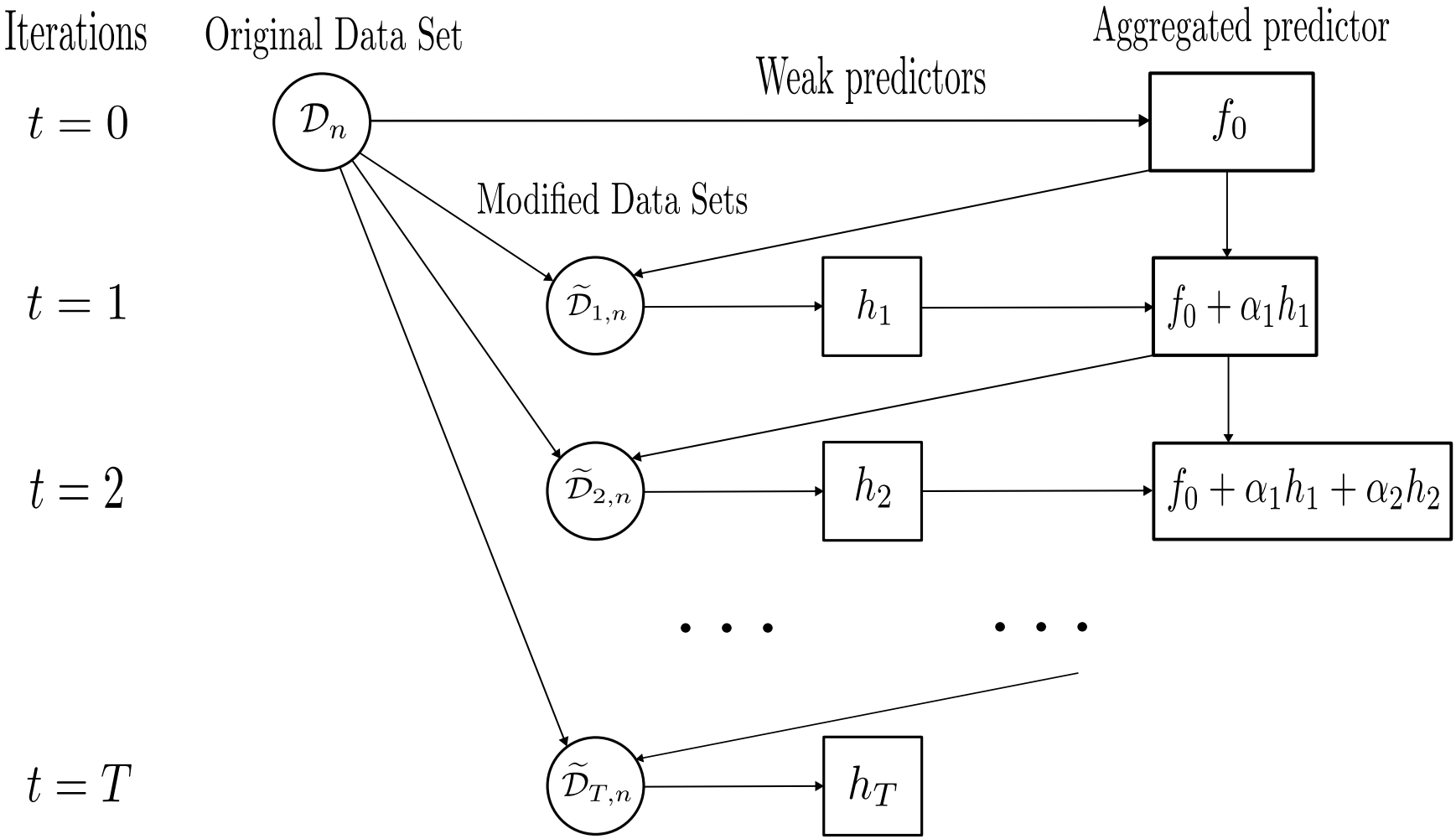
Boosting in a scheme



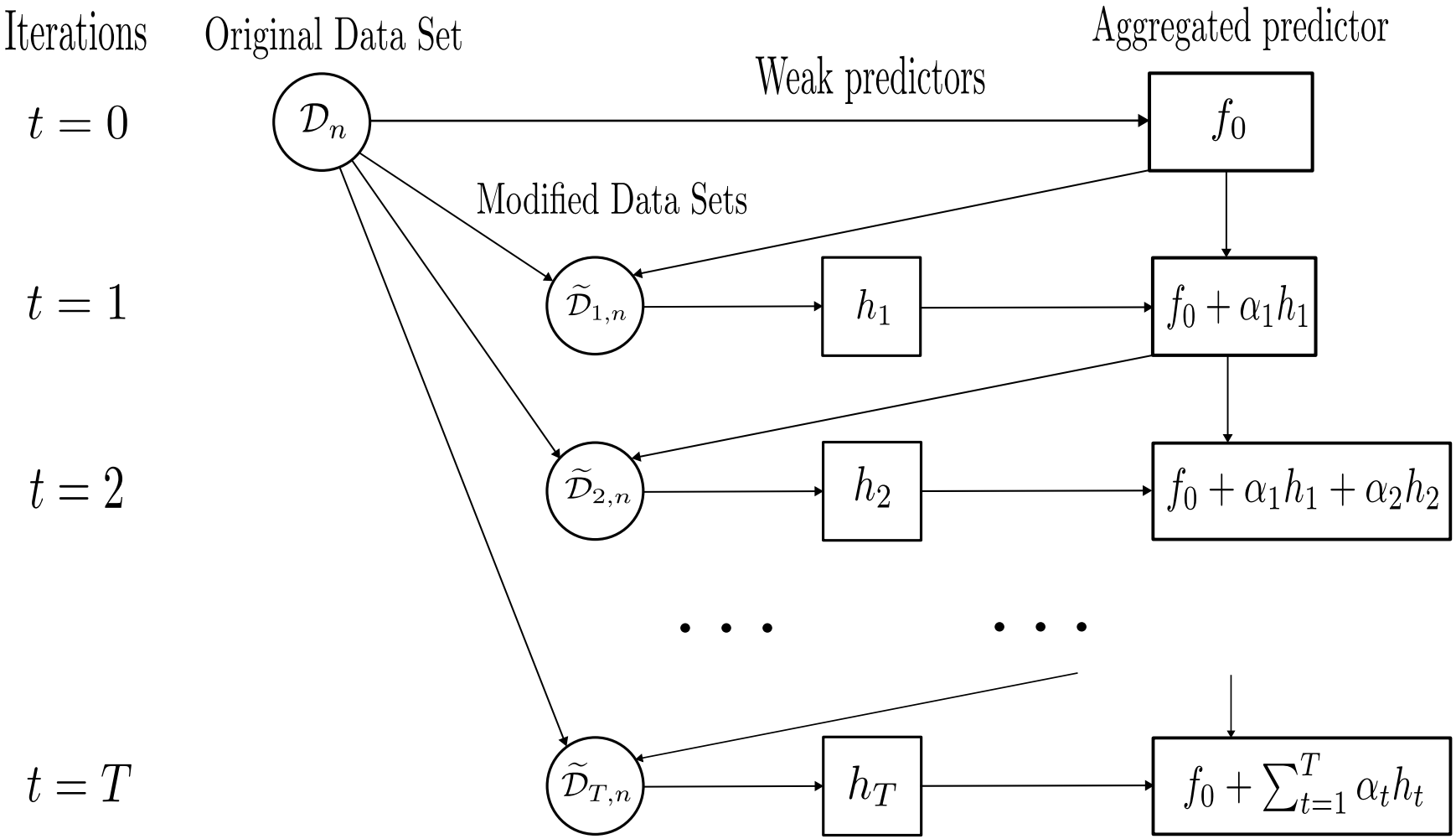
Boosting in a scheme



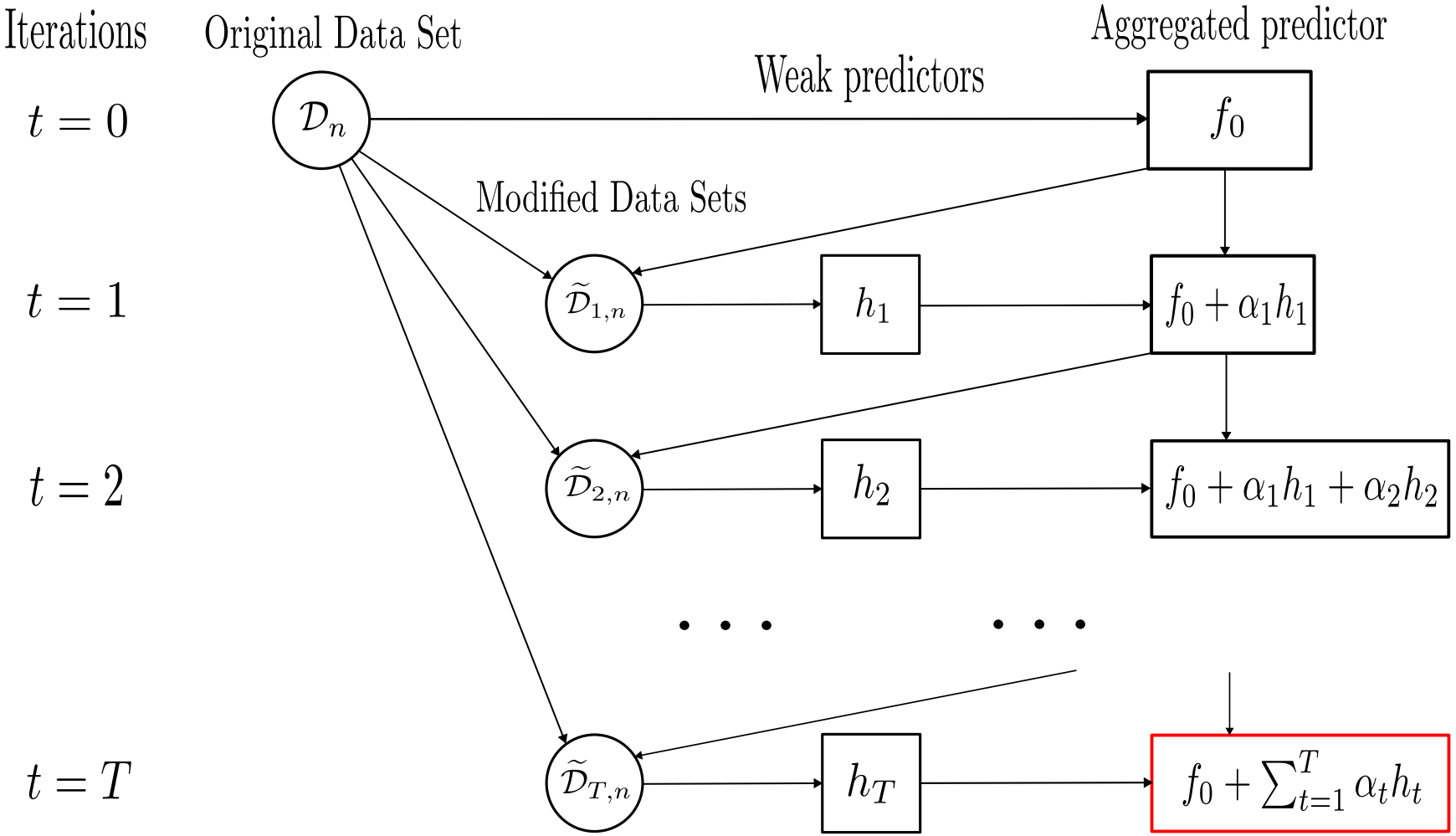
Boosting in a scheme



Boosting in a scheme



Boosting in a scheme



Outline

Generic Boosting pseudo-algorithm

Boosting algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a set $\mathcal{H}$ of weak learners, a loss ℓ.
2. Set $f = 0$
3. For $t = 1, \ldots, T$
   1. Select
   $$(h_t, \alpha_t) \in \underset{h \in \mathcal{H}, \alpha \in \mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell(y_i, f(x_i) + \alpha h(x_i)) \tag{2}$$
   2. Update $f = f + \alpha_t h_t$
4. The final predictor is given by $f = \sum_{t=1}^{T} \alpha_t h_t$ in regression (or its sign in binary classification).

Idea. At each iteration, we try to find the weak predictor that, when added to the current overall predictor, decreases the most the risk on the training set.

Generic Boosting pseudo-algorithm

Boosting algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a set $\mathcal{H}$ of weak learners, a loss ℓ.
2. Set $f = 0$
3. For $t = 1, \ldots, T$
   1. Select
   $$(h_t, \alpha_t) \in \underset{h \in \mathcal{H}, \alpha \in \mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell(y_i, f(x_i) + \alpha h(x_i)) \tag{2}$$
   2. Update $f = f + \alpha_t h_t$
4. The final predictor is given by $f = \sum_{t=1}^{T} \alpha_t h_t$ in regression (or its sign in binary classification).

Remarks. Finding minimizers in equation is difficult in general:

- Optimization on a large space - the class $\mathcal{H}$ can be infinite or very large, for example the set of all decision trees.
- Joint optimization procedure in (h, α).

Generic Boosting pseudo-algorithm

Boosting algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a set $\mathcal{H}$ of weak learners, a loss ℓ.
2. Set $f = 0$
3. For $t = 1, \ldots, T$
   1. Select
   $$(h_t, \alpha_t) \in \underset{h \in \mathcal{H}, \alpha \in \mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell(y_i, f(x_i) + \alpha h(x_i)) \tag{2}$$
   2. Update $f = f + \alpha_t h_t$
4. The final predictor is given by $f = \sum_{t=1}^{T} \alpha_t h_t$ in regression (or its sign in binary classification).

A special case to understand better the procedure: the exponential loss (Adaboost).

AdaBoost algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ with $Y_i \in \{-1, 1\}$, a set $\mathcal{H}$ of weak learners.
2. Set $f = 0$
3. For $t = 1, \ldots, T$
   1. Select
   $$(h_t, \alpha_t) \in \operatorname*{argmin}_{h,\alpha} \sum_{i=1}^{n} \exp(-y_i(f(x_i) + \alpha h(x_i))) \tag{3}$$
   2. Update $f = f + \alpha_t h_t$
4. The final predictor is given by $f = \sum_{t=1}^{T} \alpha_t h_t$ in regression (or its sign in binary classification).

AdaBoost algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ with $Y_i \in \{-1, 1\}$, a set $\mathcal{H}$ of weak learners.
2. Set $f = 0$
3. For $t = 1, \ldots, T$
   1. Select
   $$(h_t, \alpha_t) \in \operatorname*{argmin}_{h,\alpha} \sum_{i=1}^{n} \exp(-y_i(f(x_i) + \alpha h(x_i))) \tag{3}$$
   2. Update $f = f + \alpha_t h_t$
4. The final predictor is given by $f = \sum_{t=1}^{T} \alpha_t h_t$ in regression (or its sign in binary classification).

Assumptions

Assume that, for all $h \in \mathcal{H}$,

- $-h \in \mathcal{H}$ (symmetry)
- There exist $1 \leq i \neq j \leq n$ such that $h(X_i) = Y_i$ and $h(X_j) \neq Y_j$ (no perfect classifier).

Under these assumptions, AdaBoost can be rewritten as follows.

Adaboost algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a set $\mathcal{H}$ of weak learners.
2. Set $f = 0$ and $w_{1,i} = 1/n$ for all $i \in \{1, \ldots, n\}$.
3. For $t = 1, \ldots, T$
   1. Select
   $$h_t \in \operatorname*{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{n} w_{t,i} \mathbb{1}_{Y_i \neq h(X_i)} \tag{3}$$
   2. Compute
   $$\alpha_t = \frac{1}{2} \log\left(\frac{\varepsilon_t(h_t)}{1 - \varepsilon_t(h_t)}\right), \quad \text{with} \quad \varepsilon_t(h_t) = \sum_{i=1}^{n} w_{t,i} \mathbb{1}_{Y_i \neq h_t(X_i)}. \tag{4}$$
   3. Set $w_{t+1,i} = w_{t,i} \exp(-y_i \alpha_t h_t(x_i))/Z_{t+1}$, where Z_{t+1} is such that $\sum_{i=1}^{n} w_{t+1,i} = 1$.
   4. Update $f = f + \alpha_t h_t$
4. The final predictor is given by $f = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t\right)$.

Adaboost algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a set $\mathcal{H}$ of weak learners.
2. Set $f = 0$ and $w_{1,i} = 1/n$ for all $i \in \{1, \ldots, n\}$.
3. For $t = 1, \ldots, T$
   1. Select
   $$h_t \in \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} w_{t,i} \mathbb{1}_{Y_i \neq h(X_i)} \tag{3}$$
   2. Compute
   $$\alpha_t = \frac{1}{2} \log\left(\frac{\varepsilon_t(h_t)}{1 - \varepsilon_t(h_t)}\right), \quad \text{with} \quad \varepsilon_t(h_t) = \sum_{i=1}^{n} w_{t,i} \mathbb{1}_{Y_i \neq h_t(X_i)}. \tag{4}$$
   3. Set $w_{t+1,i} = w_{t,i} \exp(-y_i \alpha_t h_t(x_i))/Z_{t+1}$, where Z_{t+1} is such that $\sum_{i=1}^{n} w_{t+1,i} = 1$.
   4. Update $f = f + \alpha_t h_t$
4. The final predictor is given by $f = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t\right)$.

- Each observation receives an initial weight $w_{1,i} = 1/n$
- The weak classifier that minimizes the $0-1$ loss on the weighted sample is selected
- Its coefficient α is computed based on its error
- The aggregated classifier is computed and weights are updated.

Adaboost algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a set $\mathcal{H}$ of weak learners.
2. Set $f = 0$ and $w_{1,i} = 1/n$ for all $i \in \{1, \ldots, n\}$.
3. For $t = 1, \ldots, T$
   1. Select
   $$h_t \in \operatorname*{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{n} w_{t,i} \mathbb{1}_{Y_i \neq h(X_i)} \tag{3}$$
   2. Compute
   $$\alpha_t = \frac{1}{2} \log\left(\frac{\varepsilon_t(h_t)}{1 - \varepsilon_t(h_t)}\right), \quad \text{with} \quad \varepsilon_t(h_t) = \sum_{i=1}^{n} w_{t,i} \mathbb{1}_{Y_i \neq h_t(X_i)}. \tag{4}$$
   3. Set $w_{t+1,i} = w_{t,i} \exp(-y_i \alpha_t h_t(x_i))/Z_{t+1}$, where Z_{t+1} is such that $\sum_{i=1}^{n} w_{t+1,i} = 1$.
   4. Update $f = f + \alpha_t h_t$
4. The final predictor is given by $f = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t\right)$.

In practice, finding the best weak predictor (equation (3)) might be difficult. We thus simply fit an algorithm on the weighted training sample instead.

Adaboost algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a weak learning procedure (e.g., decision trees of depth one trained with CART).
2. Set $f = 0$ and $w_{1,i} = 1/n$ for all $i \in \{1, \ldots, n\}$.
3. For $t = 1, \ldots, T$
   1. Fit a weak learning algorithm (e.g., CART trees of depth 1, stumps) to the training set with weights $w_{t,i}$. Denote by h_t the predictor.
   2. Compute
   $$\alpha_t = \frac{1}{2} \log \left(\frac{\varepsilon_t(h_t)}{1 - \varepsilon_t(h_t)} \right), \quad \text{with} \quad \varepsilon_t(h_t) = \sum_{i=1}^{n} w_{t,i} \mathbb{1}_{Y_i \neq h_t(X_i)}. \tag{3}$$
   3. Set $w_{t+1,i} = w_{t,i} \exp(-y_i \alpha_t h_t(x_i))/Z_{t+1}$, where Z_{t+1} is such that $\sum_{i=1}^{n} w_{t+1,i} = 1$.
   4. Update $f = f + \alpha_t h_t$
4. The final predictor is given by $f = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t \right)$.

AdaBoost / Gradient Boosting

- Adaboost among state-of-the-art methods for binary classification (2003 Gödel Prize, see Freund and Schapire, 1995)
- Can be adapted to multiclass (Hastie et al., 2009) and regression (Drucker, 1997) via the function `AdaBoostClassifier` and `AdaBoostRegressor` in scikit-learn.
- Beware: it was claimed that AdaBoost does not overfit but this is not true! Hyperparameters need to be chosen carefully to prevent overfitting.

AdaBoost / Gradient Boosting

Pros:

- Powerful classifier
- Deterministic strategy: two runs of Adaboost leads to the same classifier.

Cons:

- Sensible to noise / outliers in the data

  $\rightarrow$ High importance is given to incorrectly classified observations due to the *exponential* loss

Going back to the general case

Boosting algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a set $\mathcal{H}$ of weak learners, a loss ℓ.
2. Set $f_0 = 0$
3. For $t = 1, \ldots, T$
   1. Select
   $$(h_t, \alpha_t) \in \underset{h \in \mathcal{H}, \alpha \in \mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell(y_i, f_{t-1}(x_i) + \alpha h(x_i)) \tag{4}$$
   2. Update $f_t = f_{t-1} + \alpha_t h_t$
4. The final predictor is $f_T = f_0 + \sum_{t=1}^{T} \alpha_t h_t$ in regression (or its sign in binary classification).

Remarks. At each step, we try to find the base/weak predictor h_t that reduces the most the training set error of the aggregated predictor.

Going back to the general case

Boosting algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \dots, (X_n, Y_n)\}$, a set $\mathcal{H}$ of weak learners, a loss ℓ.
2. Set $f_0 = 0$
3. For $t = 1, \dots, T$
   1. Select
   $$(h_t, \alpha_t) \in \underset{h \in \mathcal{H}, \alpha \in \mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell(y_i, f_{t-1}(x_i) + \alpha h(x_i)) \tag{4}$$
   2. Update $f_t = f_{t-1} + \alpha_t h_t$
4. The final predictor is $f_T = f_0 + \sum_{t=1}^{T} \alpha_t h_t$ in regression (or its sign in binary classification).

Problem. Finding minimizers in equation (4) is difficult in general:

- Optimization on a large space - the class $\mathcal{H}$ can be infinite or very large, for example the set of all decision trees.
- Joint optimization procedure in (h, α).

Going back to the general case

Boosting algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a set $\mathcal{H}$ of weak learners, a loss ℓ.
2. Set $f_0 = 0$
3. For $t = 1, \ldots, T$
   1. Select
   $$(h_t, \alpha_t) \in \underset{h \in \mathcal{H}, \alpha \in \mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell(y_i, f_{t-1}(x_i) + \alpha h(x_i)) \tag{4}$$
   2. Update $f_t = f_{t-1} + \alpha_t h_t$
4. The final predictor is $f_T = f_0 + \sum_{t=1}^{T} \alpha_t h_t$ in regression (or its sign in binary classification).

Solution. Consider a first-order approximation

$$\ell(y_i, f_{t-1}(x_i) + \alpha h(x_i)) \simeq \ell(y_i, f_{t-1}(x_i)) + \alpha h(x_i) \left[\frac{\partial \ell(y_i, z)}{\partial z} \right]_{z = f_{t-1}(x_i)}. \tag{5}$$

that is, solving

$$h_t \in \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} h(x_i) \left[\frac{\partial \ell(y_i, z)}{\partial z} \right]_{z = f_{t-1}(x_i)}. \tag{6}$$

Going back to the general case

Boosting algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a set $\mathcal{H}$ of weak learners, a loss ℓ.
2. Set $f_0 = 0$
3. For $t = 1, \ldots, T$
   1. Select
   $$h_t \in \operatorname*{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{n} h(x_i) \left[\frac{\partial \ell(y_i, z)}{\partial z} \right]_{z = f_{t-1}(x_i)}. \tag{4}$$
   2. Select
   $$\alpha_t \in \operatorname*{argmin}_{\alpha \in \mathbb{R}} \sum_{i=1}^{n} \ell(y_i, f_{t-1}(x_i) + \alpha h_t(x_i)). \tag{5}$$
   3. Update $f_t = f_{t-1} + \alpha_t h_t$
4. The final predictor is $f_T = f_0 + \sum_{t=1}^{T} \alpha_t h_t$ in regression (or its sign in binary classification).

Going back to the general case

Boosting algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a set $\mathcal{H}$ of weak learners, a loss ℓ.
2. Set $f_0 = 0$
3. For $t = 1, \ldots, T$
   1. Select h_t in $\mathcal{H}$ such that, for all $i \in \{1, \ldots, n\}$,
   $$h_t(x_i) \simeq -\left[\frac{\partial \ell(y_i, z)}{\partial z}\right]_{z=f_{t-1}(x_i)}. \tag{4}$$
   2. Select
   $$\alpha_t \in \underset{\alpha \in \mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell(y_i, f_{t-1}(x_i) + \alpha h_t(x_i)). \tag{5}$$
   3. Update $f_t = f_{t-1} + \alpha_t h_t$
4. The final predictor is $f_T = f_0 + \sum_{t=1}^{T} \alpha_t h_t$ in regression (or its sign in binary classification).

Going back to the general case

Gradient Boosting algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a set $\mathcal{H}$ of weak learners, a loss ℓ.
2. Set $f_0 = 0$
3. For $t = 1, \ldots, T$
   1. For all $i \in \{1, \ldots, n\}$, compute the gradient
   $$r_i = -\left[\frac{\partial \ell(y_i, z)}{\partial z}\right]_{z=f_{t-1}(x_i)}. \tag{4}$$
   2. Denote by h_t the predictor, obtained by fitting the weak learning procedure to the *residual* dataset $(x_1, r_1), \ldots, (x_n, r_n)$.
   3. Select
   $$\alpha_t \in \operatorname*{argmin}_{\alpha \in \mathbb{R}} \sum_{i=1}^{n} \ell(y_i, f_{t-1}(x_i) + \alpha h_t(x_i)). \tag{5}$$
   4. Update $f_t = f_{t-1} + \alpha_t h_t$
4. The final predictor is $f_T = f_0 + \sum_{t=1}^{T} \alpha_t h_t$ in regression (or its sign in binary classification).

Going back to the general case

Gradient Boosting algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a set $\mathcal{H}$ of weak learners, a loss ℓ.
2. Set $f \in \text{argmin}_{c \in \mathbb{R}} \sum_{i=1}^{n} \ell(y_i, c)$
3. For $t = 1, \ldots, T$
   1. For all $i \in \{1, \ldots, n\}$, compute the gradient
   $$r_i = - \left[\frac{\partial \ell(y_i, z)}{\partial z} \right]_{z = f_{t-1}(x_i)}. \tag{4}$$
   2. Denote by h_t the predictor, obtained by fitting the weak learning procedure to the *residual* dataset $(x_1, r_1), \ldots, (x_n, r_n)$.
   3. Via Backtracking line search, find an approximated solution
   $$\alpha_t \in \underset{\alpha \in \mathbb{R}}{\text{argmin}} \sum_{i=1}^{n} \ell(y_i, f_{t-1}(x_i) + \alpha h_t(x_i)). \tag{5}$$
   4. Update $f_t = f_{t-1} + \alpha_t h_t$
4. The final predictor is $f_T = f_0 + \sum_{t=1}^{T} \alpha_t h_t$ in regression (or its sign in binary classification).

Going back to the general case

Gradient Boosting algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a set $\mathcal{H}$ of weak learners, a loss ℓ.
2. Set $f \in \text{argmin}_{c \in \mathbb{R}} \sum_{i=1}^{n} \ell(y_i, c)$
3. For $t = 1, \ldots, T$
   1. For all $i \in \{1, \ldots, n\}$, compute the gradient
   $$r_i = - \left[\frac{\partial \ell(y_i, z)}{\partial z} \right]_{z = f_{t-1}(x_i)}. \tag{4}$$
   2. Denote by h_t the predictor, obtained by fitting the weak learning procedure to the *residual* dataset $(x_1, r_1), \ldots, (x_n, r_n)$.
   3. Via Backtracking line search, find an approximated solution
   $$\alpha_t \in \underset{\alpha \in \mathbb{R}}{\text{argmin}} \sum_{i=1}^{n} \ell(y_i, f_{t-1}(x_i) + \alpha h_t(x_i)). \tag{5}$$
   4. Update $f_t = f_{t-1} + \nu \alpha_t h_t$
4. The final predictor is $f_T = f_0 + \nu \sum_{t=1}^{T} \alpha_t h_t$ in regression (or its sign in binary classification).

Gradient boosting. Improving the performance of a base/weak learning algorithm (e.g. decision trees) by successively fitting it to a modified training set. Here, outputs of the training set are replaced by the negative gradient of the loss at each iteration.

A simple example: Gradient boosting with quadratic loss

Now, consider the Gradient Boosting algorithm with the square loss: $\ell(y, z) = (y - z)^2$.

Gradient Boosting algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a weak learning procedure.
2. Set $f \in \text{argmin}_{c \in \mathbb{R}} \sum_{i=1}^{n} \ell(y_i, c)$
3. For $t = 1, \ldots, T$
   1. For all $i \in \{1, \ldots, n\}$, compute the gradient
   $$r_i = - \left[\frac{\partial \ell(y_i, z)}{\partial z} \right]_{z = f_{t-1}(x_i)}. \tag{6}$$
   2. Denote by h_t the predictor, obtained by fitting the weak learning procedure to the *residual* dataset $(x_1, r_1), \ldots, (x_n, r_n)$.
   3. Via Backtracking line search, find an approximated solution
   $$\alpha_t \in \underset{\alpha \in \mathbb{R}}{\text{argmin}} \sum_{i=1}^{n} \ell(y_i, f_{t-1}(x_i) + \alpha h_t(x_i)). \tag{7}$$
   4. Update $f_t = f_{t-1} + \nu \alpha_t h_t$
4. $f_T = f_0 + \nu \sum_{t=1}^{T} \alpha_t h_t$ in regression (or its sign in binary classification).

A simple example: Gradient boosting with quadratic loss

Now, consider the Gradient Boosting algorithm with the square loss: $\ell(y, z) = (y - z)^2$. The pseudo residuals are given by

$$r_i = -\left[\frac{\partial \ell(y_i, z)}{\partial z}\right]_{z=f_{t-1}(x_i)}. \tag{6}$$

Gradient Boosting algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a weak learning procedure.
2. Set $f \in \text{argmin}_{c \in \mathbb{R}} \sum_{i=1}^n \ell(y_i, c)$
3. For $t = 1, \ldots, T$
   1. For all $i \in \{1, \ldots, n\}$, compute the gradient
   $$r_i = -\left[\frac{\partial \ell(y_i, z)}{\partial z}\right]_{z=f_{t-1}(x_i)}. \tag{7}$$
   2. Denote by h_t the predictor, obtained by fitting the weak learning procedure to the *residual* dataset $(x_1, r_1), \ldots, (x_n, r_n)$.
   3. Via Backtracking line search, find an approximated solution
   $$\alpha_t \in \underset{\alpha \in \mathbb{R}}{\text{argmin}} \sum_{i=1}^n \ell(y_i, f_{t-1}(x_i) + \alpha h_t(x_i)). \tag{8}$$
   4. Update $f_t = f_{t-1} + \nu \alpha_t h_t$
4. $f_T = f_0 + \nu \sum_{t=1}^T \alpha_t h_t$ in regression (or its sign in binary classification).

A simple example: Gradient boosting with quadratic loss

Now, consider the Gradient Boosting algorithm with the square loss: $\ell(y, z) = (y - z)^2$. The pseudo residuals are given by

$$r_i = 2(y_i - f_{t-1}(x_i)). \tag{6}$$

Gradient Boosting algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a weak learning procedure.
2. Set $f \in \text{argmin}_{c \in \mathbb{R}} \sum_{i=1}^{n} \ell(y_i, c)$
3. For $t = 1, \ldots, T$
   1. For all $i \in \{1, \ldots, n\}$, compute the gradient
   $$r_i = - \left[\frac{\partial \ell(y_i, z)}{\partial z} \right]_{z = f_{t-1}(x_i)}. \tag{7}$$
   2. Denote by h_t the predictor, obtained by fitting the weak learning procedure to the *residual* dataset $(x_1, r_1), \ldots, (x_n, r_n)$.
   3. Via Backtracking line search, find an approximated solution
   $$\alpha_t \in \underset{\alpha \in \mathbb{R}}{\text{argmin}} \sum_{i=1}^{n} \ell(y_i, f_{t-1}(x_i) + \alpha h_t(x_i)). \tag{8}$$
   4. Update $f_t = f_{t-1} + \nu \alpha_t h_t$
4. $f_T = f_0 + \nu \sum_{t=1}^{T} \alpha_t h_t$ in regression (or its sign in binary classification).

A simple example: Gradient boosting with quadratic loss

Now, consider the Gradient Boosting algorithm with the square loss: $\ell(y, z) = (y - z)^2$. The pseudo residuals are given by

$$r_i = 2(y_i - f_{t-1}(x_i)). \tag{6}$$

Gradient Boosting algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a weak learning procedure.
2. Set $f \in \text{argmin}_{c \in \mathbb{R}} \sum_{i=1}^{n} \ell(y_i, c)$
3. For $t = 1, \ldots, T$
   1. For all $i \in \{1, \ldots, n\}$, compute the gradient
   $$r_i = 2(y_i - f_{t-1}(x_i)). \tag{7}$$
   2. Denote by h_t the predictor, obtained by fitting the weak learning procedure to the *residual* dataset $(x_1, r_1), \ldots, (x_n, r_n)$.
   3. Via Backtracking line search, find an approximated solution
   $$\alpha_t \in \underset{\alpha \in \mathbb{R}}{\text{argmin}} \sum_{i=1}^{n} \ell(y_i, f_{t-1}(x_i) + \alpha h_t(x_i)). \tag{8}$$
   4. Update $f_t = f_{t-1} + \nu \alpha_t h_t$
4. $f_T = f_0 + \nu \sum_{t=1}^{T} \alpha_t h_t$ in regression (or its sign in binary classification).

A simple example: Gradient boosting with quadratic loss

Now, consider the Gradient Boosting algorithm with the square loss: $\ell(y, z) = (y - z)^2$.

Gradient Boosting algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a weak learning procedure.
2. Set $f \in \operatorname{argmin}_{c \in \mathbb{R}} \sum_{i=1}^{n} \ell(y_i, c)$
3. For $t = 1, \ldots, T$
   1. For all $i \in \{1, \ldots, n\}$, compute the gradient
   $$r_i = 2(y_i - f_{t-1}(x_i)). \tag{6}$$
   2. Denote by h_t the predictor, obtained by fitting the weak learning procedure to the *residual* dataset $(x_1, r_1), \ldots, (x_n, r_n)$.
   3. Via Backtracking line search, find an approximated solution
   $$\alpha_t \in \operatorname*{argmin}_{\alpha \in \mathbb{R}} \sum_{i=1}^{n} \ell(y_i, f_{t-1}(x_i) + \alpha h_t(x_i)). \tag{7}$$
   4. Update $f_t = f_{t-1} + \nu \alpha_t h_t$
4. $f_T = f_0 + \nu \sum_{t=1}^{T} \alpha_t h_t$ in regression (or its sign in binary classification).

Gradient Boosting with quadratic loss amounts to iteratively fitting a weak learner to the residuals of the current predictor.

Outline

Gradient Boosting Decision Tree (GBDT)

GBDT algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a weak learning procedure, a loss ℓ.
2. Set $f \in \text{argmin}_{c \in \mathbb{R}} \sum_{i=1}^{n} \ell(y_i, c)$
3. For $t = 1, \ldots, T$
   1. For all $i \in \{1, \ldots, n\}$, compute the gradient
   $$r_i = - \left[\frac{\partial \ell(y_i, z)}{\partial z} \right]_{z = f_{t-1}(x_i)}. \tag{8}$$
   2. Denote by h_t the predictor, obtained by fitting a weak learning procedure to the *residual* dataset $(x_1, r_1), \ldots, (x_n, r_n)$.
   3. Via Backtracking line search, find an approximated solution
   $$\alpha_t \in \underset{\alpha \in \mathbb{R}}{\text{argmin}} \sum_{i=1}^{n} \ell(y_i, f_{t-1}(x_i) + \alpha h_t(x_i)). \tag{9}$$
   4. Update $f_t = f_{t-1} + \nu \alpha_t h_t$
4. The final predictor is f_T in regression (or its sign in binary classification).

Also called "Multiple additive regression trees" (MARS, see Friedman and Meulman, 2003).

Gradient Boosting Decision Tree (GBDT)

GBDT algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a weak learning procedure (shallow trees), a loss ℓ.
2. Set $f \in \text{argmin}_{c \in \mathbb{R}} \sum_{i=1}^{n} \ell(y_i, c)$
3. For $t = 1, \ldots, T$
   1. For all $i \in \{1, \ldots, n\}$, compute the gradient
   $$r_i = - \left[\frac{\partial \ell(y_i, z)}{\partial z} \right]_{z = f_{t-1}(x_i)}. \tag{8}$$
   2. Denote by h_t the predictor, obtained by fitting a shallow tree to the *residual* dataset $(x_1, r_1), \ldots, (x_n, r_n)$.
   3. Via Backtracking line search, find an approximated solution
   $$\alpha_t \in \underset{\alpha \in \mathbb{R}}{\text{argmin}} \sum_{i=1}^{n} \ell(y_i, f_{t-1}(x_i) + \alpha h_t(x_i)). \tag{9}$$
   4. Update $f_t = f_{t-1} + \nu \alpha_t h_t$
4. The final predictor is f_T in regression (or its sign in binary classification).

Also called "Multiple additive regression trees" (MARS, see Friedman and Meulman, 2003).

Gradient Boosting Decision Tree (GBDT)

GBDT algorithm

1. Inputs: a dataset $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, a weak learning procedure (shallow trees), a loss ℓ.
2. Set $f \in \text{argmin}_{c \in \mathbb{R}} \sum_{i=1}^{n} \ell(y_i, c)$
3. For $t = 1, \ldots, T$
   1. For all $i \in \{1, \ldots, n\}$, compute the gradient
   $$r_i = - \left[\frac{\partial \ell(y_i, z)}{\partial z} \right]_{z = f_{t-1}(x_i)}. \tag{8}$$
   2. Denote by h_t the predictor, obtained by fitting a shallow tree to the *residual* dataset $(x_1, r_1), \ldots, (x_n, r_n)$. Let us denote by $R_1, \ldots, R_j$ the leaves of h_t.
   3. Via direct computations, select
   $$\alpha_t \in \underset{\alpha \in \mathbb{R}^J}{\text{argmin}} \sum_{i=1}^{n} \ell(y_i, f_{t-1}(x_i) + \sum_{j=1}^{J} \alpha_j \mathbb{1}_{x_i \in R_j}). \tag{9}$$
   4. Update $\quad f_t(x) = f_{t-1}(x) + \nu \sum_{j=1}^{J} \alpha_{j,t} \mathbb{1}_{x_i \in R_j}$.
4. The final predictor is f_T in regression (or its sign in binary classification).

Also called "Multiple additive regression trees" (MARS, see Friedman and Meulman, 2003).

Parameters in Gradient Boosting Decision Tree (GBDT)

Functions `GradientBoostingClassifier` and `GradientBoostingRegressor` in scikit-learn.

Parameters in Gradient Boosting Decision Tree (GBDT)

Optimization/Statistical complexity trade-off.

- Number of boosted trees T.

  $\rightarrow$ Controls both the statistical complexity of the final predictor (how many trees are aggregated) *and* the number of iterations in the optimization procedure (as adding a tree can be seen as a gradient descent step).

- The shrinkage parameter ν.

  $\rightarrow$ It helps to prevent overfitting in the early iterations. It can be seen as the learning rate of the boosting procedure. The smaller ν, the more iterations are needed to converge. Related to the optimization procedure

$\Rightarrow$ Heuristic: fix ν to a small value (typically 0.1) and $T = 100$ (or optimize T via early stopping).

Parameters in Gradient Boosting Decision Tree (GBDT)

Tree structure.

- Same parameters as in CART
- No split randomization is performed, i.e. `max-features` $= d$
- Maximal depth is set to `max-depth` $= 3$
- `square-loss` in regression / `log-loss` in classification (as in logistic/multinomial regression).

Gradient Boosting

Pros:

- State-of-the-art algorithm for supervised learning with tabular data
- Can handle regression and classification tasks for various loss functions
- Can handle continous and discrete features
- Deterministic strategy: two runs leads to the same predictor.

Cons:

- Can be computationally expensive - may require a large number of trees
- May overfit if too many trees are used (early stopping can be used to prevent overfitting)
- May be sensible to outliers / noise in the data

XGBoost

- Stands for eXtreme Gradient Boosting (Chen and Guestrin, 2016)
- State-of-the-art methods on tabular data sets (often better than random forests)

XGBoost

Differences with Gradient Boosting

- Second-order approximation of the loss with a penalty term (number of leaves + leaf values): assuming that f_{t-1} has been built, the loss at step t is

$$\underset{h}{\operatorname{argmin}} \sum_{i=1}^{n} \underbrace{\ell(y_i, f_{t-1}(x_i) + h(x_i))}_{\text{Replaced by a 2nd-order approx.}} + \Omega(h)$$

  $\rightarrow$ New objective function to build a boosted tree
- Feature discretization based on second-order statistics
- Feature subsampling can be used as in random forest (in each node) or prior to the tree construction.
- Computationally more efficient (handling sparse data, parallel and distributed computing)

XGBoost

Benefits/drawbacks

- Computationally efficient
- Can be applied to large-scale data set due to the new split finding scheme
- High predictive acuracy on most tabular data sets.

Variable importance in Gradient Boosting

The same variable importances as that in Random Forests can be computed:

- Mean Decrease in Impurity (MDI) for each tree, which is then averaged (with equal weights even with shrinkage) over trees
- Mean Decrease in Accuracy which is computed on the final boosted predictor.

Variable importance in Gradient Boosting

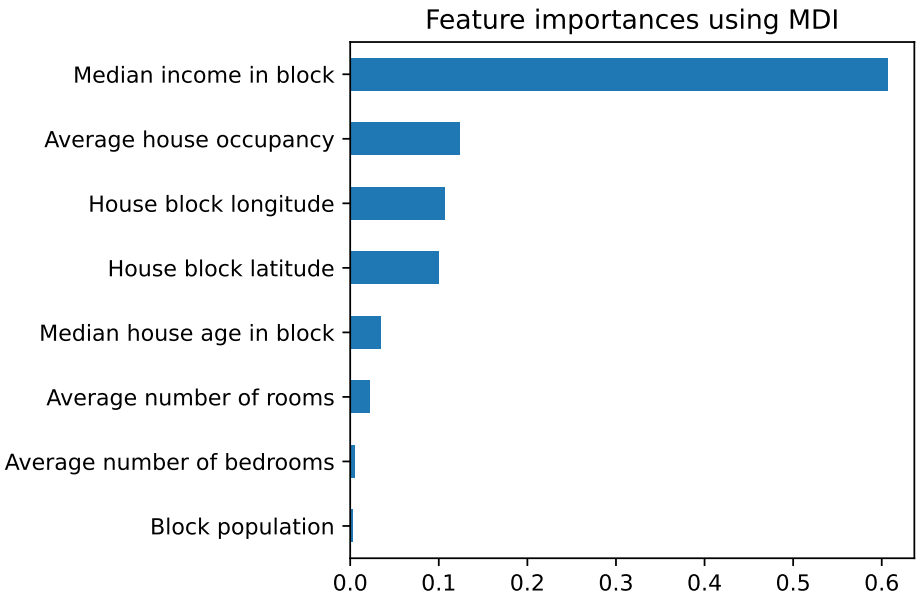


Figure: MDI computed with Gradient Boosting on the California housing data set.

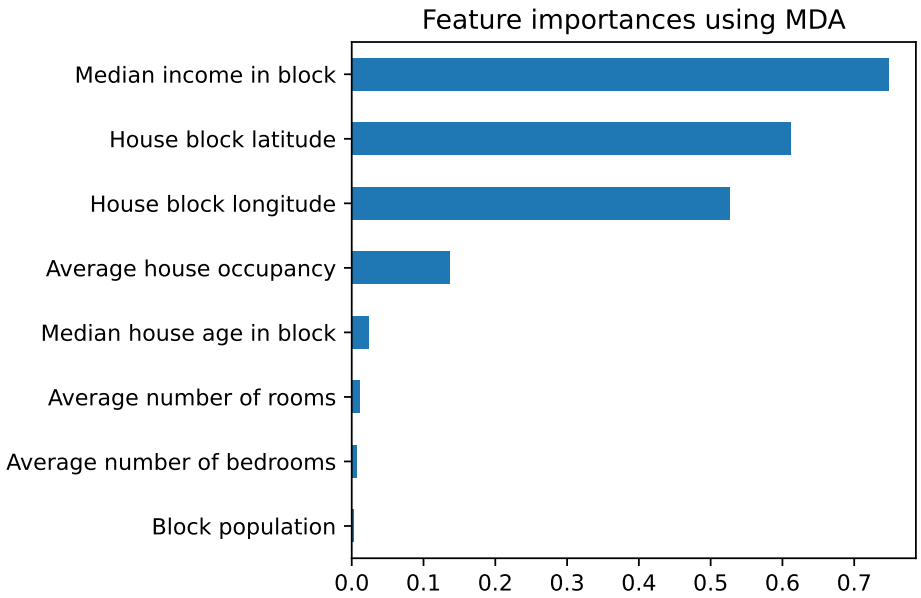


Figure: MDA computed with Gradient Boosting on the California housing data set.

[BDS22] C. Bénard, S. Da Veiga, and E. Scornet. "MDA for random forests: inconsistency, and a practical solution via the Sobol-MDA". In: *Biometrika* (2022).

[Bou+11] A.-L. Boulesteix et al. "Random forest Gini importance favours SNPs with large minor allele frequency: impact, sources and recommendations". In: *Briefings in Bioinformatics* 13 (2011), pp. 292–304.

[Bre01] L. Breiman. "Random forests". In: *Machine Learning* 45 (2001), pp. 5–32.

[Bre02] L. Breiman. "Manual on setting up, using, and understanding random forests v3. 1". In: *Statistics Department University of California Berkeley, CA, USA* 1 (2002), p. 58.

[CG16] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[Dru97] Harris Drucker. "Improving regressors using boosting techniques". In: *Icml*. Vol. 97. Citeseer. 1997, pp. 107–115.

[FM03] Jerome H Friedman and Jacqueline J Meulman. "Multiple additive regression trees with application in epidemiology". In: *Statistics in medicine* 22.9 (2003), pp. 1365–1381.

[FS95] Yoav Freund and Robert E Schapire. "A desicion-theoretic generalization of on-line learning and an application to boosting". In: *Computational Learning Theory: Second European Conference, EuroCOLT'95 Barcelona, Spain, March 13–15, 1995 Proceedings 2*. Springer. 1995, pp. 23–37.

[Has+09] Trevor Hastie et al. "Multi-class adaboost". In: *Statistics and its Interface* 2.3 (2009), pp. 349–360.

[Li+19] X. Li et al. "A debiased mdi feature importance measure for random forests". In: *Advances in Neural Information Processing Systems*. 2019, pp. 8049–8059.

[Loe22] Markus Loecher. "Unbiased variable importance for random forests". In: *Communications in Statistics-Theory and Methods* 51.5 (2022), pp. 1413–1425.

[Nic11] K. K. Nicodemus. "Letter to the editor: On the stability and ranking of predictors from random forest variable importance measures". In: *Briefings in bioinformatics* 12.4 (2011), pp. 369–373.

[NM09] K. K. Nicodemus and J. D. Malley. "Predictor correlation impacts machine learning algorithms: implications for genomic studies". In: *Bioinformatics* 25.15 (2009), pp. 1884–1890.

[Str+07] C. Strobl et al. “Bias in random forest variable importance measures: Illustrations, sources and a solution”. In: *BMC bioinformatics* 8.1 (2007), p. 25.

[Wil+21] Brian D Williamson et al. “A general framework for inference on algorithm-agnostic variable importance”. In: *Journal of the American Statistical Association* (2021), pp. 1–14.

[ZH21] Zhengze Zhou and Giles Hooker. “Unbiased measurement of feature importance in tree-based methods”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 15.2 (2021), pp. 1–21.