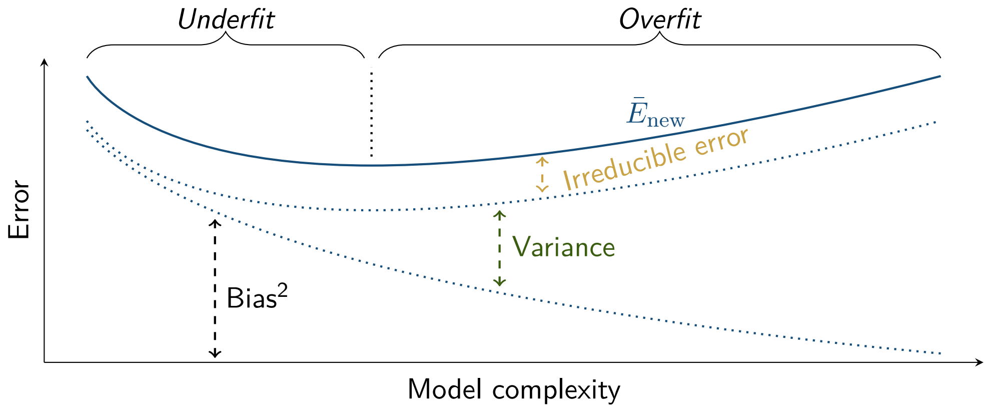
Lecture 6 – Tree-based methods, Bagging and Boosting

UPPSALA UNIVERSITET

David Sumpter
Division of Systems and Control
Department of Information Technology
Uppsala University.

Summary of Lecture 5



Finding a balanced fit (neither over- nor underfit) is called the
the bias-variance tradeoff.

Contents – Lecture 6

1. Classification and regression trees (CART)
2. Bagging – *a general variance reduction technique*
3. Random forests
4. Boosting

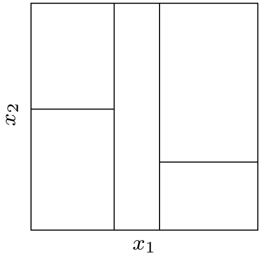
The idea behind tree-based methods

In both regression and classification settings we seek a function $\widehat{y}(\mathbf{x})$ which maps the input $\mathbf{x}$ into a prediction.

The idea behind tree-based methods

In both regression and classification settings we seek a function $\widehat{y}(\mathbf{x})$ which maps the input $\mathbf{x}$ into a prediction.

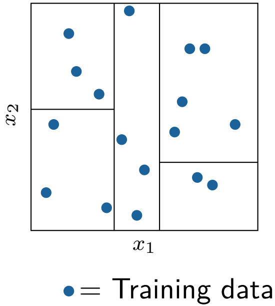
One **flexible** way of designing this function is to partition the input space into disjoint regions and fit a simple model in each region.



The idea behind tree-based methods

In both regression and classification settings we seek a function $\widehat{y}(\mathbf{x})$ which maps the input $\mathbf{x}$ into a prediction.

One **flexible** way of designing this function is to partition the input space into disjoint regions and fit a simple model in each region.



- **Classification:** Majority vote within the region.
- **Regression:** Mean of training data within the region.

Finding the partition

The key challenge in using this strategy is to find a good partition.

Even if we restrict our attention to seemingly simple regions (e.g. “boxes”), finding an ***optimal*** partition w.r.t. minimizing the training error is ***computationally infeasible!***

UPPSALA UNIVERSITET

Finding the partition

The key challenge in using this strategy is to find a good partition.

Even if we restrict our attention to seemingly simple regions (e.g. "boxes"), finding an ***optimal*** partition w.r.t. minimizing the training error is ***computationally infeasible!***

Instead, we use a "greedy" approach: **recursive binary splitting**.

1. Select one input variable x_j and a cut-point s. Partition the input space into two half-spaces,

$$\{\mathbf{x} : x_j < s\} \qquad \{\mathbf{x} : x_j \geq s\}.$$

Finding the partition

The key challenge in using this strategy is to find a good partition.

Even if we restrict our attention to seemingly simple regions (e.g. "boxes"), finding an ***optimal*** partition w.r.t. minimizing the training error is ***computationally infeasible!***

Instead, we use a "greedy" approach: **recursive binary splitting**.

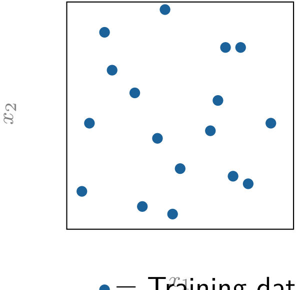
1. Select one input variable x_j and a cut-point s. Partition the input space into two half-spaces,

$$\{\mathbf{x} : x_j < s\} \qquad \{\mathbf{x} : x_j \geq s\}.$$

2. Repeat this splitting for each region until some stopping criterion is met (e.g., no region contains more than 5 training data points).

Recursive binary splitting

Partitioning of input space

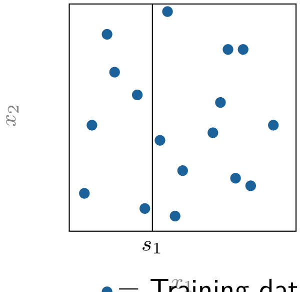


● = Training data

Tree representation

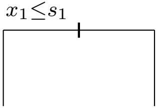
Recursive binary splitting

Partitioning of input space



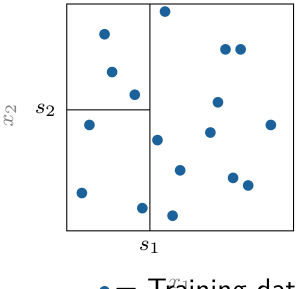
$\bullet$ = Training data

Tree representation



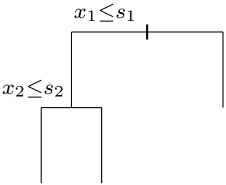
Recursive binary splitting

Partitioning of input space



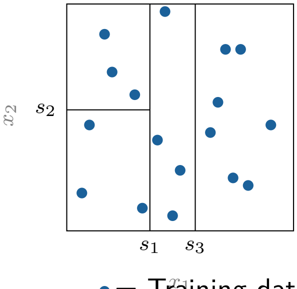
$\bullet =$ Training data

Tree representation



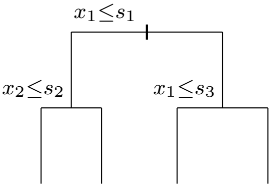
Recursive binary splitting

Partitioning of input space



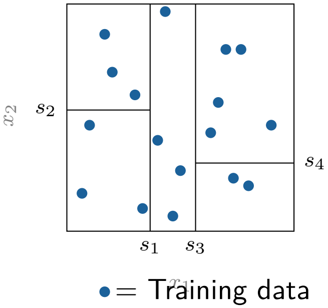
● = Training data

Tree representation

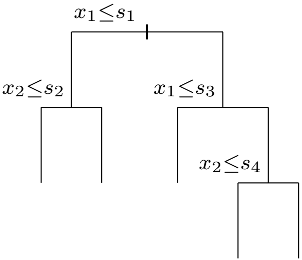


Recursive binary splitting

Partitioning of input space

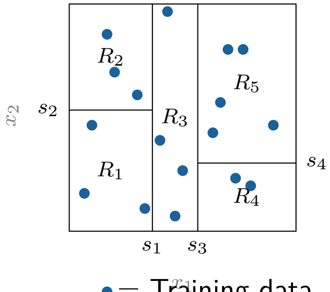


Tree representation



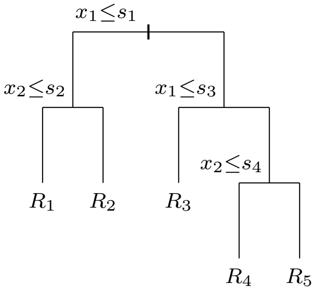
Recursive binary splitting

Partitioning of input space



$\bullet$ = Training data

Tree representation



UPPSALA UNIVERSITET

Regression trees (I/II)

Once the input space is partitioned into L regions, $R_1, R_2, \dots, R_L$ the prediction model is

$$\widehat{y}_\star = \sum_{\ell=1}^{L} \widehat{y}_\ell \mathbb{I}\{\mathbf{x}_\star \in R_\ell\},$$

where $\mathbb{I}\{\mathbf{x}_\star \in R_\ell\}$ is the indicator function

$$\mathbb{I}\{\mathbf{x}_\star \in R_\ell\} = \begin{cases} 1 & \text{if } \mathbf{x}_\star \in R_\ell \\ 0 & \text{if } \mathbf{x}_\star \notin R_\ell \end{cases}$$

and $\widehat{y}_\ell$ is a constant prediction within each region.

Regression trees (I/II)

Once the input space is partitioned into L regions, $R_1, R_2, \dots, R_L$ the prediction model is

$$\widehat{y}_\star = \sum_{\ell=1}^{L} \widehat{y}_\ell \mathbb{I}\{\mathbf{x}_\star \in R_\ell\},$$

where $\mathbb{I}\{\mathbf{x}_\star \in R_\ell\}$ is the indicator function

$$\mathbb{I}\{\mathbf{x}_\star \in R_\ell\} = \begin{cases} 1 & \text{if } \mathbf{x}_\star \in R_\ell \\ 0 & \text{if } \mathbf{x}_\star \notin R_\ell \end{cases}$$

and $\widehat{y}_\ell$ is a constant prediction within each region.
For regression trees we use

$$\widehat{y}_\ell = \text{avarage}\{y_i : \mathbf{x}_i \in R_\ell\}$$

Regression trees (II/II) - How do we find the partition?

Recursive binary splitting is **greedy** - each split is made in order to minimize the loss **without looking ahead** at future splits.

UPPSALA UNIVERSITET

Regression trees (II/II) - How do we find the partition?

Recursive binary splitting is **greedy** - each split is made in order to minimize the loss **without looking ahead** at future splits.

For any j and s, define

$$R_1(j,s) = \{\mathbf{x} \,|\, x_j < s\} \qquad \text{and} \qquad R_2(j,s) = \{\mathbf{x} \,|\, x_j \geq s\}.$$

Regression trees (II/II) - How do we find the partition?

Recursive binary splitting is **greedy** - each split is made in order to minimize the loss **without looking ahead** at future splits.

For any j and s, define

$$R_1(j,s) = \{\mathbf{x} \,|\, x_j < s\} \qquad \text{and} \qquad R_2(j,s) = \{\mathbf{x} \,|\, x_j \geq s\}.$$

We then seek (j,s) that minimize

$$\sum_{i:\mathbf{x}_i \in R_1(j,s)} (y_i - \widehat{y}_1(j,s))^2 + \sum_{i:\mathbf{x}_i \in R_2(j,s)} (y_i - \widehat{y}_2(j,s))^2$$

where

$$\widehat{y}_1 = \text{avarage}\{y_i : \mathbf{x}_i \in R_1(j,s)\}$$
$$\widehat{y}_2 = \text{avarage}\{y_i : \mathbf{x}_i \in R_2(j,s)\}$$

UPPSALA UNIVERSITET

Regression trees (II/II) - How do we find the partition?

Recursive binary splitting is **greedy** - each split is made in order to minimize the loss **without looking ahead** at future splits.

For any j and s, define

$$R_1(j,s) = \{\mathbf{x} \,|\, x_j < s\} \qquad \text{and} \qquad R_2(j,s) = \{\mathbf{x} \,|\, x_j \geq s\}.$$

We then seek (j,s) that minimize

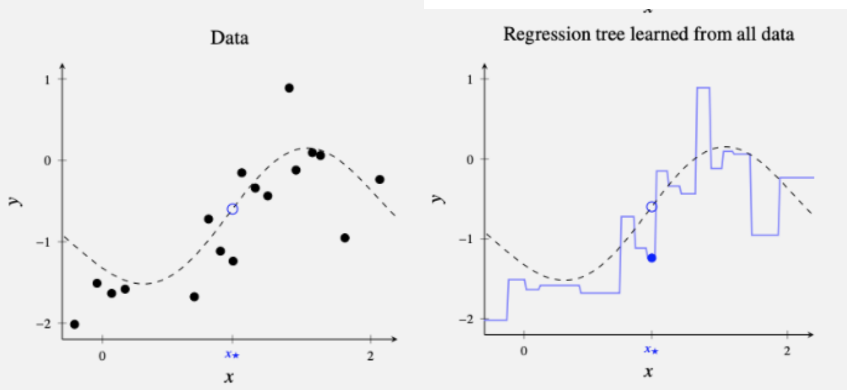
$$\sum_{i:\mathbf{x}_i \in R_1(j,s)} (y_i - \widehat{y}_1(j,s))^2 + \sum_{i:\mathbf{x}_i \in R_2(j,s)} (y_i - \widehat{y}_2(j,s))^2$$

where

$$\widehat{y}_1 = \text{avarage}\{y_i : \mathbf{x}_i \in R_1(j,s)\}$$
$$\widehat{y}_2 = \text{avarage}\{y_i : \mathbf{x}_i \in R_2(j,s)\}$$

This optimization problem is easily solved by "brute force" by evaluating all possible splits.

Example: Regression trees



Classification trees

Classification trees are constructed similarly to regression trees, but with *two differences*.

Classification trees

Classification trees are constructed similarly to regression trees, but with *two differences*.

Firstly, the class prediction for each region is based on the proportion of data points from each class in that region.

Classification trees

Classification trees are constructed similarly to regression trees, but with *two differences*.

Firstly, the class prediction for each region is based on the proportion of data points from each class in that region. Let

$$\widehat{\pi}_{\ell m} = \frac{1}{n_\ell} \sum_{i:\mathbf{x}_i \in R_\ell} \mathbb{I}\{y_i = m\}$$

be the proportion of training data points in the lth region that belong to the mth class.

Classification trees

Classification trees are constructed similarly to regression trees, but with *two differences*.

Firstly, the class prediction for each region is based on the proportion of data points from each class in that region. Let

$$\widehat{\pi}_{\ell m} = \frac{1}{n_\ell} \sum_{i:\mathbf{x}_i \in R_\ell} \mathbb{I}\{y_i = m\}$$

be the proportion of training data points in the lth region that belong to the mth class. Then we approximate

$$p(y = m \,|\, \mathbf{x}_\star) \approx \sum_{\ell=1}^{L} \widehat{\pi}_{\ell m} \mathbb{I}\{\mathbf{x}_\star \in R_\ell\}$$

Classification trees

Secondly, the squared loss used to construct the tree needs to be replaced by a measure suitable to categorical outputs.

Classification trees

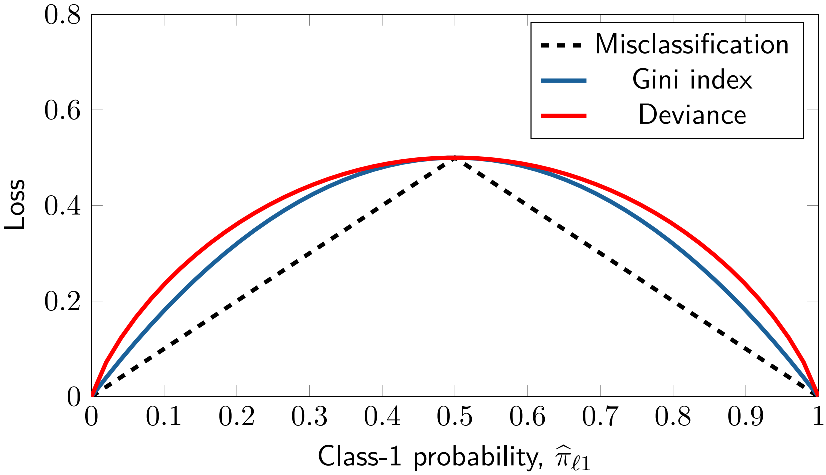
Secondly, the squared loss used to construct the tree needs to be replaced by a measure suitable to categorical outputs.

Three common error measures are,

$$
\begin{aligned}
\text{Misclassification error:} \quad & 1 - \max_m \widehat{\pi}_{\ell m} \\
\text{Entropy/deviance:} \quad & -\textstyle\sum_{m=1}^{M} \widehat{\pi}_{\ell m} \log \widehat{\pi}_{\ell m} \\
\text{Gini index:} \quad & \textstyle\sum_{m=1}^{M} \widehat{\pi}_{\ell m}(1 - \widehat{\pi}_{\ell m})
\end{aligned}
$$

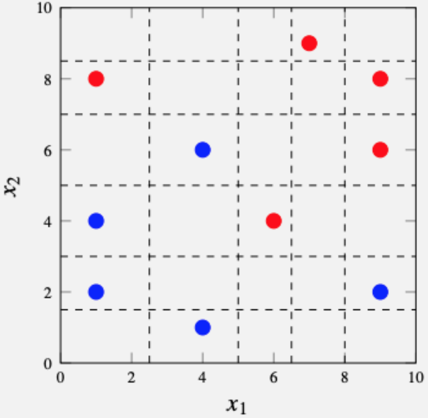
Classification error measures

For a binary classification problem ($M = 2$)



Example using entropy

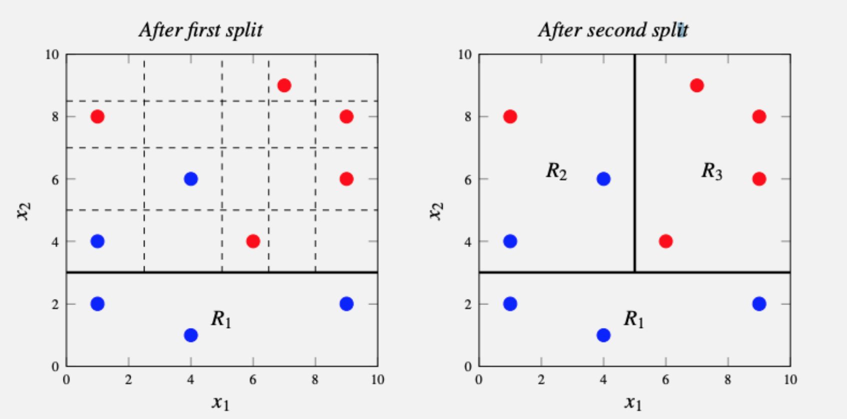
x_1	x_2	y
9.0	2.0	Blue
1.0	4.0	Blue
4.0	6.0	Blue
4.0	1.0	Blue
1.0	2.0	Blue
1.0	8.0	Red
6.0	4.0	Red
7.0	9.0	Red
9.0	8.0	Red
9.0	6.0	Red



Example using entropy

Split (R_1)	n_1	$\widehat{\pi}_{1B}$	$\widehat{\pi}_{1R}$	Q_1	n_2	$\widehat{\pi}_{2B}$	$\widehat{\pi}_{2R}$	Q_2	$n_1Q_1 + n_2Q_2$
$x_1 < 2.5$	3	2/3	1/3	0.64	7	3/7	4/7	0.68	6.69
$x_1 < 5.0$	5	4/5	1/5	0.50	5	1/5	4/5	0.50	5.00
$x_1 < 6.5$	6	4/6	2/6	0.64	4	1/4	3/4	0.56	6.07
$x_1 < 8.0$	7	4/7	3/7	0.68	3	1/3	2/3	0.64	6.69
$x_2 < 1.5$	1	1/1	0/1	0.00	9	4/9	5/9	0.69	6.18
$x_2 < 3.0$	3	3/3	0/3	0.00	7	2/7	5/7	0.60	**4.18**
$x_2 < 5.0$	5	4/5	1/5	0.50	5	1/5	4/5	0.06	5.00
$x_2 < 7.0$	7	5/7	2/7	0.60	3	0/3	3/3	0.00	**4.18**
$x_2 < 8.5$	9	5/9	4/9	0.69	1	0/1	1/1	0.00	6.18

Example using entropy



Improving CART

The flexibility/complexity of classification and regression trees (CART) is decided by the tree depth.

- ! To obtain a **small bias** the tree need to be grown deep,
- ! but this results in a **high variance!**

The performance of (simple) CARTs is often unsatisfactory!

UPPSALA UNIVERSITET

Improving CART

The flexibility/complexity of classification and regression trees (CART) is decided by the tree depth.

! To obtain a **small bias** the tree need to be grown deep,

! but this results in a **high variance!**

The performance of (simple) CARTs is often unsatisfactory!

To improve the practical performance:

- Bagging and Random Forests
- Boosted trees

Bagging (I/II)

For now, assume that we have access to B **independent** datasets $\mathcal{T}^1, \dots, \mathcal{T}^B$. We can then train a separate deep tree $\widehat{y}^b(\mathbf{x})$ for each dataset, $1, \dots, B$.

Bagging (I/II)

For now, assume that we have access to B **independent** datasets $\mathcal{T}^1, \dots, \mathcal{T}^B$. We can then train a separate deep tree $\widehat{y}^b(\mathbf{x})$ for each dataset, $1, \dots, B$.

- Each $\widehat{y}^b(\mathbf{x})$ has a **low bias** but **high variance**

Bagging (I/II)

For now, assume that we have access to B **independent** datasets $\mathcal{T}^1, \dots, \mathcal{T}^B$. We can then train a separate deep tree $\widehat{y}^b(\mathbf{x})$ for each dataset, $1, \dots, B$.

- Each $\widehat{y}^b(\mathbf{x})$ has a **low bias** but **high variance**
- By averaging

$$\widehat{y}_{\text{bag}}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} \widehat{y}^b(\mathbf{x})$$

the bias is kept small, but variance is reduced by a factor B!

UPPSALA UNIVERSITET

Bagging (II/II)

Obvious problem We only have access to one training dataset.

UPPSALA UNIVERSITET

Bagging (II/II)

Obvious problem We only have access to one training dataset.

<u>**Solution**</u> Bootstrap the data!

- Sample n times with replacement from the original training data $\mathcal{T} = \{\mathbf{x}_i, y_i\}_{i=1}^n$

Bagging (II/II)

Obvious problem We only have access to one training dataset.

Solution Bootstrap the data!

- Sample n times with replacement from the original training data $\mathcal{T} = \{\mathbf{x}_i, y_i\}_{i=1}^n$
- Repeat B times to generate B "bootstrapped" training datasets $\widetilde{\mathcal{T}}^1, \ldots, \widetilde{\mathcal{T}}^B$

Bagging (II/II)

Obvious problem We only have access to one training dataset.

Solution Bootstrap the data!

- Sample n times with replacement from the original training data $\mathcal{T} = \{\mathbf{x}_i, y_i\}_{i=1}^n$
- Repeat B times to generate B "bootstrapped" training datasets $\widetilde{\mathcal{T}}^1, \dots, \widetilde{\mathcal{T}}^B$

UPPSALA UNIVERSITET

Bagging (II/II)

Obvious problem We only have access to one training dataset.

Solution Bootstrap the data!

- Sample n times with replacement from the original training data $\mathcal{T} = \{\mathbf{x}_i, y_i\}_{i=1}^n$
- Repeat B times to generate B "bootstrapped" training datasets $\widetilde{\mathcal{T}}^1, \dots, \widetilde{\mathcal{T}}^B$

For each bootstrapped dataset $\widetilde{\mathcal{T}}^b$ we train a tree $\widetilde{y}^b(\mathbf{x})$.

Bagging (II/II)

Obvious problem We only have access to one training dataset.

Solution Bootstrap the data!

- Sample n times with replacement from the original training data $\mathcal{T} = \{\mathbf{x}_i, y_i\}_{i=1}^n$
- Repeat B times to generate B "bootstrapped" training datasets $\widetilde{\mathcal{T}}^1, \dots, \widetilde{\mathcal{T}}^B$

For each bootstrapped dataset $\widetilde{\mathcal{T}}^b$ we train a tree $\widetilde{y}^b(\mathbf{x})$. Averaging these,

$$\widetilde{y}^b_{\text{bag}}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} \widetilde{y}^b(\mathbf{x})$$

is called "bootstrap aggregation", or bagging.

Bagging - Toy example

ex) Assume that we have a training set

$$\mathcal{T} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3), (\mathbf{x}_4, y_4)\}.$$

Bagging - Toy example

ex) Assume that we have a training set

$$\mathcal{T} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3), (\mathbf{x}_4, y_4)\}.$$

We generate, say, $B = 3$ datasets by bootstrapping:

$$\widetilde{\mathcal{T}}^1 = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3), (\mathbf{x}_3, y_3)\}$$
$$\widetilde{\mathcal{T}}^2 = \{(\mathbf{x}_1, y_1), (\mathbf{x}_4, y_4), (\mathbf{x}_4, y_4), (\mathbf{x}_4, y_4)\}$$
$$\widetilde{\mathcal{T}}^3 = \{(\mathbf{x}_1, y_1), (\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_2, y_2)\}$$

Bagging - Toy example

ex) Assume that we have a training set

$$\mathcal{T} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3), (\mathbf{x}_4, y_4)\}.$$

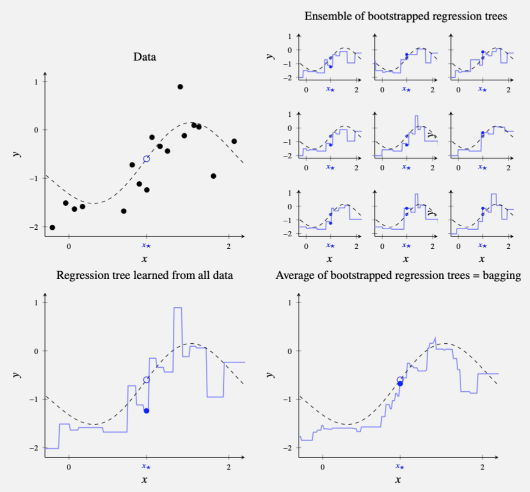
We generate, say, $B = 3$ datasets by bootstrapping:

$$\widetilde{\mathcal{T}}^1 = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3), (\mathbf{x}_3, y_3)\}$$
$$\widetilde{\mathcal{T}}^2 = \{(\mathbf{x}_1, y_1), (\mathbf{x}_4, y_4), (\mathbf{x}_4, y_4), (\mathbf{x}_4, y_4)\}$$
$$\widetilde{\mathcal{T}}^3 = \{(\mathbf{x}_1, y_1), (\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_2, y_2)\}$$

Compute $B = 3$ (deep) regression trees $\tilde{y}^1(\mathbf{x})$, $\tilde{y}^2(\mathbf{x})$ and $\tilde{y}^3(\mathbf{x})$, one for each dataset $\widetilde{\mathcal{T}}^1$, $\widetilde{\mathcal{T}}^2$, and $\widetilde{\mathcal{T}}^3$, and average

$$\tilde{y}_{\text{bag}}(\mathbf{x}) = \frac{1}{3}\sum_{b=1}^{3} \tilde{y}^b(\mathbf{x})$$

Example: Regression trees



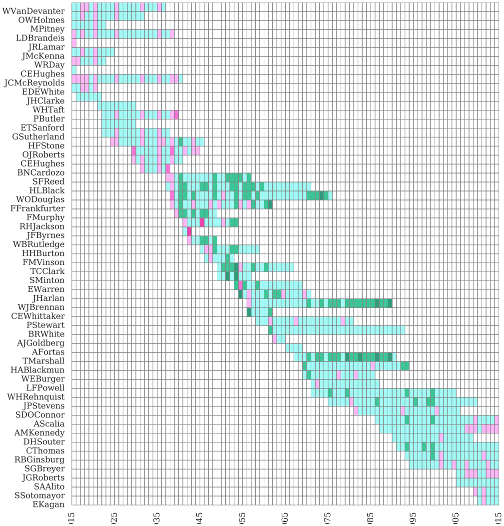
Application) Predicting US Supreme Court behavior

Random forest classifier built on SCDB data[1] to predict the votes of Supreme Court justices:

$$Y \in \{\texttt{affirm}, \texttt{reverse}, \texttt{other}\}$$

Result: 70% correct classifications

D. M. Katz, M. J. Bommarito II and J. Blackman. **A General Approach for Predicting the Behavior of the Supreme Court of the United States**. *arXiv.org*, arXiv:1612.03473v2, January 2017.



[1] `http://supremecourtdatabase.org`

Application) Predicting US Supreme Court behavior

Not only have random forests proven to be "unreasonably effective" in a wide array of supervised learning contexts, but in our testing, random forests outperformed other common approaches including support vector machines [. . .] and feedforward artificial neural network models such as multi-layer perceptron

— Katz, Bommarito II and Blackman (arXiv:1612.03473v2)

Random forests

Bagging can drastically improve the performance of CART!

However, the B bootstrapped dataset are ***correlated***
$\Rightarrow$ the variance reduction due to averaging is diminished.

[2] Proposed by Leo Breiman, inventor of random forests.

UPPSALA UNIVERSITET

Random forests

Bagging can drastically improve the performance of CART!

However, the B bootstrapped dataset are ***correlated***
$\Rightarrow$ the variance reduction due to averaging is diminished.

Idea: De-correlate the B trees by randomly perturbing each tree.

[2] Proposed by Leo Breiman, inventor of random forests.

Random forests

Bagging can drastically improve the performance of CART!

However, the B bootstrapped dataset are ***correlated***
$\Rightarrow$ the variance reduction due to averaging is diminished.

Idea: De-correlate the B trees by randomly perturbing each tree.

A **random forest** is constructed by bagging, but for each split in each tree only a ***random subset*** of $q \leq p$ inputs are considered as splitting variables.

Rule of thumb: $q = \sqrt{p}$ for classification trees and $q = p/3$ for regression trees.[2]

[2] Proposed by Leo Breiman, inventor of random forests.

Random forest pseudo-code

Algorithm Random forest for regression

1. For $b = 1$ to B *(can run in parallel)*

Random forest pseudo-code

Algorithm Random forest for regression

1. For $b = 1$ to B *(can run in parallel)*
   (a) Draw a bootstrap data set $\widetilde{\mathcal{T}}$ of size n from $\mathcal{T}$.

Random forest pseudo-code

Algorithm Random forest for regression

1. For $b = 1$ to B *(can run in parallel)*
   (a) Draw a bootstrap data set $\widetilde{\mathcal{T}}$ of size n from $\mathcal{T}$.
   (b) Grow a regression tree by repeating the following steps until a minimum node size is reached:

Random forest pseudo-code

Algorithm Random forest for regression

1. For $b = 1$ to B *(can run in parallel)*
   (a) Draw a bootstrap data set $\widetilde{\mathcal{T}}$ of size n from $\mathcal{T}$.
   (b) Grow a regression tree by repeating the following steps until a minimum node size is reached:
      i. Select q out of the p input variables uniformly at random.

Random forest pseudo-code

Algorithm Random forest for regression

1. For $b = 1$ to B *(can run in parallel)*
   (a) Draw a bootstrap data set $\widetilde{\mathcal{T}}$ of size n from $\mathcal{T}$.
   (b) Grow a regression tree by repeating the following steps until a minimum node size is reached:
      i. Select q out of the p input variables uniformly at random.
      ii. Find the variable x_j among the q selected, and the corresponding split point s, that minimizes the squared error.

Random forest pseudo-code

Algorithm Random forest for regression

1. For $b = 1$ to B *(can run in parallel)*
   (a) Draw a bootstrap data set $\widetilde{\mathcal{T}}$ of size n from $\mathcal{T}$.
   (b) Grow a regression tree by repeating the following steps until a minimum node size is reached:
      i. Select q out of the p input variables uniformly at random.
      ii. Find the variable x_j among the q selected, and the corresponding split point s, that minimizes the squared error.
      iii. Split the node into two children with $\{x_j \leq s\}$ and $\{x_j > s\}$.

UPPSALA UNIVERSITET

Random forest pseudo-code

Algorithm Random forest for regression

1. For $b = 1$ to B *(can run in parallel)*
   (a) Draw a bootstrap data set $\widetilde{\mathcal{T}}$ of size n from $\mathcal{T}$.
   (b) Grow a regression tree by repeating the following steps until a minimum node size is reached:
      i. Select q out of the p input variables uniformly at random.
      ii. Find the variable x_j among the q selected, and the corresponding split point s, that minimizes the squared error.
      iii. Split the node into two children with $\{x_j \leq s\}$ and $\{x_j > s\}$.
2. Final model is the average the B ensemble members,

$$\widehat{y}_\star^{\text{rf}} = \frac{1}{B} \sum_{b=1}^{B} \widetilde{y}_\star^b.$$

Random forests

The random input selection used in random forests:

- ▼ increases the bias, but often very slowly
- ▼ adds to the variance (σ^2) of each tree
- ▲ reduces the correlation between the trees

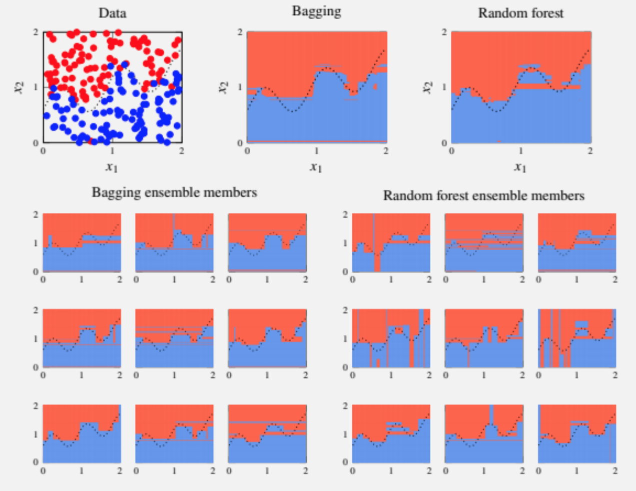
Random forests

The random input selection used in random forests:

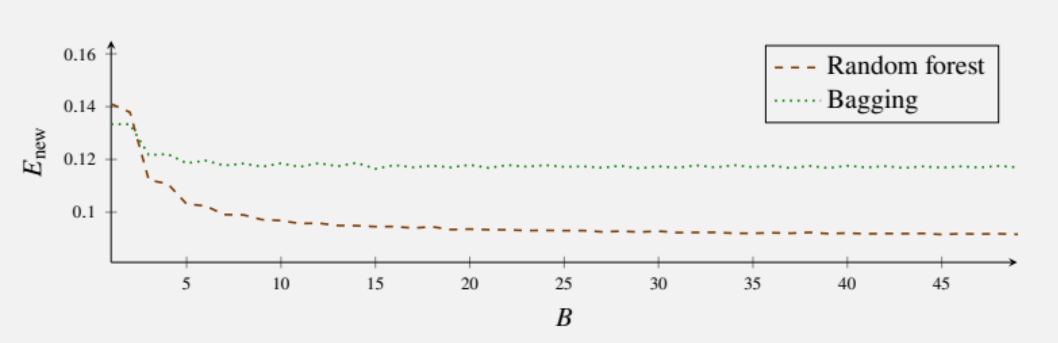
- ▼ increases the bias, but often very slowly
- ▼ adds to the variance (σ^2) of each tree
- ▲ reduces the correlation between the trees

The reduction in correlation is typically the dominant effect $\Rightarrow$ there is an overall reduction in MSE!

ex)Random forest on regression data



ex) Random forest on regression data



Advantages of random forests

Random forests have several **computational advantages**:

- Easy to do in parallel!
- Using $q < p$ potential split variables reduces the computational cost of each split.
- We ***could*** bootstrap fewer than n, say $\sqrt{n}$, data points when creating $\widetilde{\mathcal{T}}^b$ — very useful for "big data" problems.

Advantages of random forests

Random forests have several **computational advantages**:

- Easy to do in parallel!
- Using $q < p$ potential split variables reduces the computational cost of each split.
- We ***could*** bootstrap fewer than n, say $\sqrt{n}$, data points when creating $\widetilde{\mathcal{T}}^b$ — very useful for "big data" problems.

Advantages of random forests

Random forests have several **computational advantages**:

- Easy to do in parallel!
- Using $q < p$ potential split variables reduces the computational cost of each split.
- We ***could*** bootstrap fewer than n, say $\sqrt{n}$, data points when creating $\widetilde{\mathcal{T}}^b$ — very useful for "big data" problems.

UPPSALA UNIVERSITET

Advantages of random forests

Random forests have several **computational advantages**:

- Easy to do in parallel!
- Using $q < p$ potential split variables reduces the computational cost of each split.
- We ***could*** bootstrap fewer than n, say $\sqrt{n}$, data points when creating $\widetilde{\mathcal{T}}^b$ — very useful for "big data" problems.

. . . and they also come with some other benefits:

- Often works well off-the-shelf – few tuning parameters
- Requires little or no input preparation
- Implicit input selection

ex) Automatic music generation

ALYSIA: automated music generation using random forests.

- User specifies the lyrics
- ALYSIA generates accompanying music via
  - *rythm model*
  - *melody model*
- Trained on a corpus of pop songs.

https://www.youtube.com/watch?v=whgudcj82_I
https://www.withalysia.com/

M. Ackerman and D. Loker. **Algorithmic Songwriting with ALYSIA**. *In: Correia J., Ciesielski V., Liapis A. (eds) Computational Intelligence in Music, Sound, Art and Design. EvoMUSART*, 2017.

Boosting

Even a simple (classification or regression) model can typically capture some aspects of the input-output relationship.

Can we then learn an ***ensemble*** of "weak models", each describing some part of this relationship, and combine these into one "strong model"?

Boosting

Even a simple (classification or regression) model can typically capture some aspects of the input-output relationship.

Can we then learn an ***ensemble*** of "weak models", each describing some part of this relationship, and combine these into one "strong model"?

Boosting:

- ***Sequentially*** learns an ensemble of ***weak models***.

Boosting

Even a simple (classification or regression) model can typically capture some aspects of the input-output relationship.

Can we then learn an ***ensemble*** of "weak models", each describing some part of this relationship, and combine these into one "strong model"?

Boosting:

- ***Sequentially*** learns an ensemble of ***weak models***.
- Combine these into one ***strong model***.

Boosting

Even a simple (classification or regression) model can typically capture some aspects of the input-output relationship.

Can we then learn an ***ensemble*** of “weak models”, each describing some part of this relationship, and combine these into one “strong model”?

Boosting:

- ***Sequentially*** learns an ensemble of ***weak models***.
- Combine these into one ***strong model***.
- General strategy – can in principle be used to improve any supervised learning algorithm.

Boosting

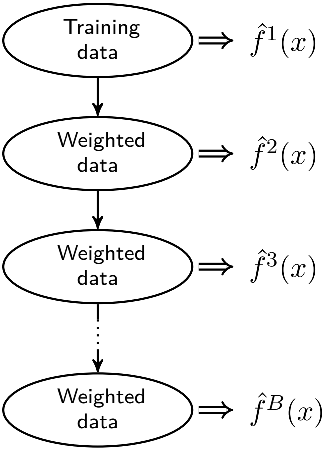
Even a simple (classification or regression) model can typically capture some aspects of the input-output relationship.

Can we then learn an ***ensemble*** of "weak models", each describing some part of this relationship, and combine these into one "strong model"?

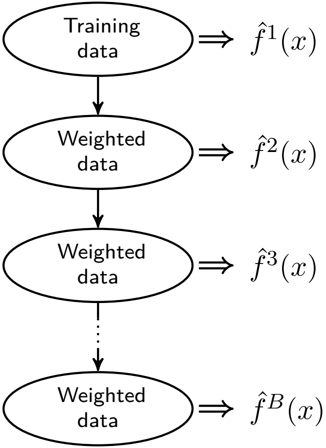
Boosting:

- ***Sequentially*** learns an ensemble of ***weak models***.
- Combine these into one ***strong model***.
- General strategy – can in principle be used to improve any supervised learning algorithm.
- One of the most successful machine learning ideas!

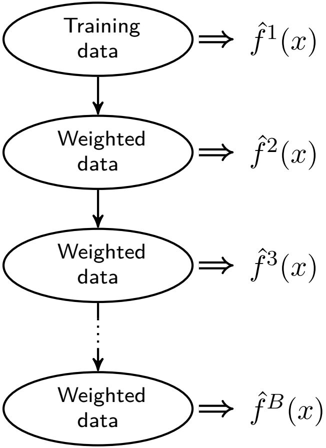
Boosting


Training data
$\Longrightarrow \hat{f}^1(x)$
Weighted data
$\Longrightarrow \hat{f}^2(x)$
Weighted data
$\Longrightarrow \hat{f}^3(x)$
Weighted data
$\Longrightarrow \hat{f}^B(x)$

Boosting

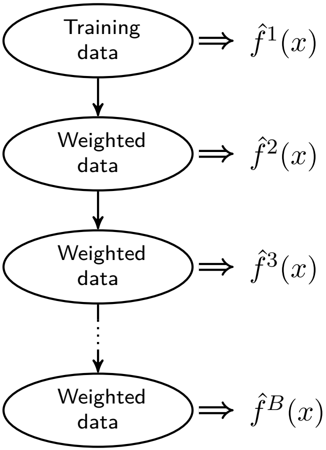

Training
data
$\Longrightarrow \hat{f}^1(x)$
Weighted
data
$\Longrightarrow \hat{f}^2(x)$
Weighted
data
$\Longrightarrow \hat{f}^3(x)$
$\vdots$
Weighted
data
$\Longrightarrow \hat{f}^B(x)$

Boosting

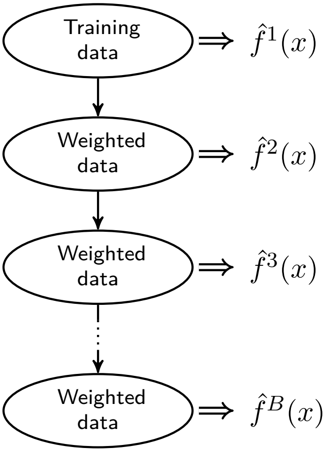


Training data
$\Longrightarrow \hat{f}^1(x)$
Weighted data
$\Longrightarrow \hat{f}^2(x)$
Weighted data
$\Longrightarrow \hat{f}^3(x)$
Weighted data
$\Longrightarrow \hat{f}^B(x)$

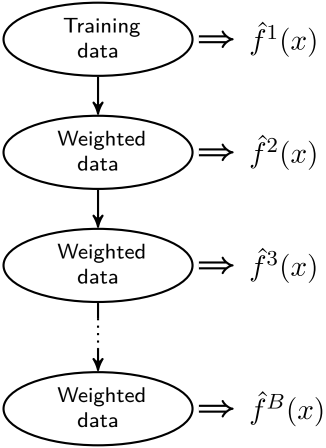
Boosting


Training
data
$\Rightarrow \hat{f}^1(x)$
Weighted
data
$\Rightarrow \hat{f}^2(x)$
Weighted
data
$\Rightarrow \hat{f}^3(x)$
Weighted
data
$\Rightarrow \hat{f}^B(x)$

Boosting


Training data
$\Rightarrow \hat{f}^1(x)$
Weighted data
$\Rightarrow \hat{f}^2(x)$
Weighted data
$\Rightarrow \hat{f}^3(x)$
Weighted data
$\Rightarrow \hat{f}^B(x)$

Boosting


Training data
Weighted data
Weighted data
Weighted data
$\hat{f}^1(x)$
$\hat{f}^2(x)$
$\hat{f}^3(x)$
$\hat{f}^B(x)$

Binary classification

We will restrict our attention to binary classification.

- Class labels are -1 and 1, i.e. $y \in \{-1, 1\}$.
- We have access to some (weak) base classifier, e.g. a classification tree.

Note. Using labels -1 and 1 is mathematically convenient as it allows us to express a majority vote between B classifiers $\widehat{G}^1(\mathbf{x}), \ldots, \widehat{G}^B(\mathbf{x})$ as

$$\text{sign}\left(\sum_{b=1}^{B} \widehat{G}^b(\mathbf{x})\right) = \begin{cases} +1 & \text{if more plus-votes than minus-votes,} \\ -1 & \text{if more minus-votes than plus-votes.} \end{cases}$$

Boosting procedure (for classification)

Boosting procedure:

1. Assign weights $w_i^1 = 1/n$ to all data points.

UPPSALA UNIVERSITET

Boosting procedure (for classification)

Boosting procedure:

1. Assign weights $w_i^1 = 1/n$ to all data points.
2. For $b = 1$ to B
   (a) Train a weak classifier $\widehat{G}^{(b)}(\mathbf{x})$ on the **weighted training data** $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$.

UPPSALA UNIVERSITET

Boosting procedure (for classification)

Boosting procedure:

1. Assign weights $w_i^1 = 1/n$ to all data points.
2. For $b = 1$ to B
   (a) Train a weak classifier $\widehat{G}^{(b)}(\mathbf{x})$ on the **weighted training data** $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$.
   (b) *Update the weights* $\{w_i^{b+1}\}_{i=1}^n$ *from* $\{w_i^b\}_{i=1}^n$*:*
      i. Increase weights for all points misclassified by $\widehat{G}^{(b)}(\mathbf{x})$.
      ii. Decrease weights for all points correctly classified by $\widehat{G}^{(b)}(\mathbf{x})$.

UPPSALA UNIVERSITET

Boosting procedure (for classification)

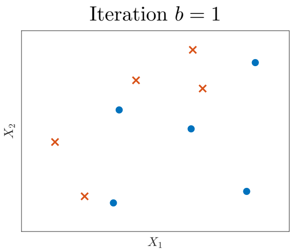
Boosting procedure:

1. Assign weights $w_i^1 = 1/n$ to all data points.
2. For $b = 1$ to B
   (a) Train a weak classifier $\widehat{G}^{(b)}(\mathbf{x})$ on the **weighted training data** $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$.
   (b) *Update the weights* $\{w_i^{b+1}\}_{i=1}^n$ *from* $\{w_i^b\}_{i=1}^n$*:*
      i. Increase weights for all points misclassified by $\widehat{G}^{(b)}(\mathbf{x})$.
      ii. Decrease weights for all points correctly classified by $\widehat{G}^{(b)}(\mathbf{x})$.

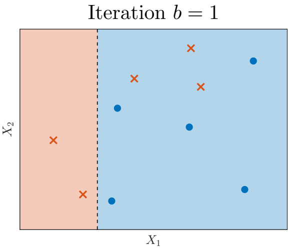
The predictions of the B classifiers, $\widehat{G}^{(1)}(\mathbf{x}), \ldots, \widehat{G}^{(B)}(\mathbf{x})$, are combined using a **weighted** majority vote:

$$\widehat{G}^B_{\text{boost}}(\mathbf{x}) = \text{sign}\left(\sum_{b=1}^{B} \alpha^{(b)} \widehat{G}^{(b)}(\mathbf{x})\right).$$

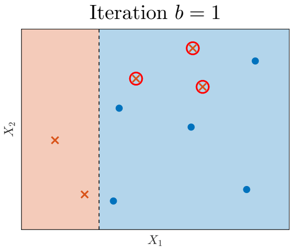
ex) Boosting illustration



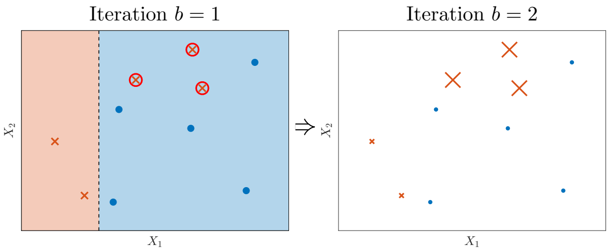
ex) Boosting illustration



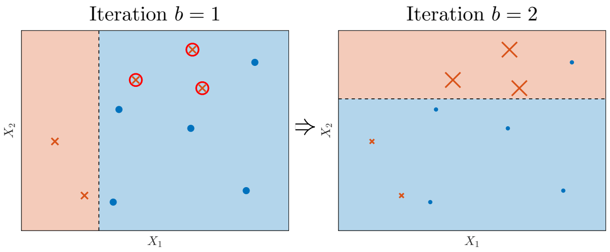
ex) Boosting illustration



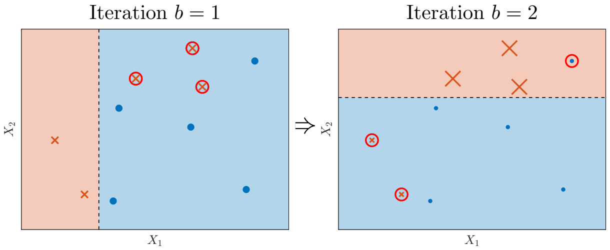
ex) Boosting illustration



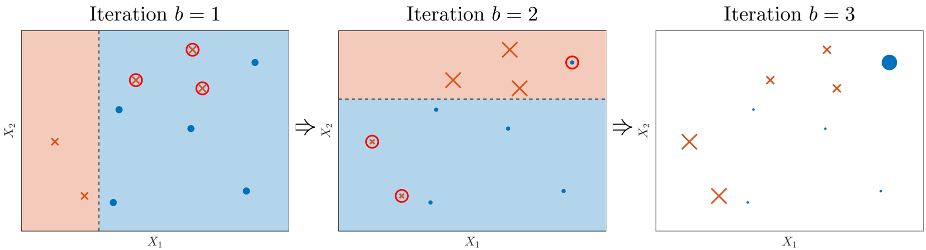
ex) Boosting illustration



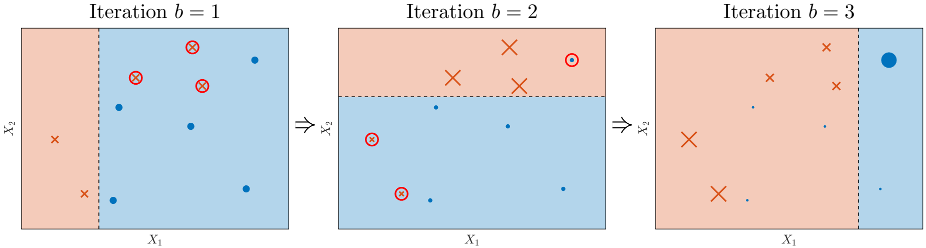
ex) Boosting illustration



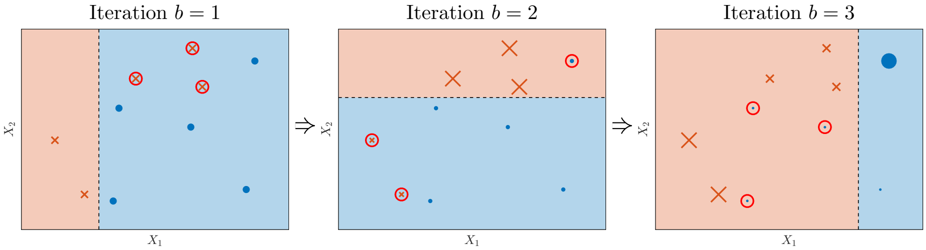
ex) Boosting illustration



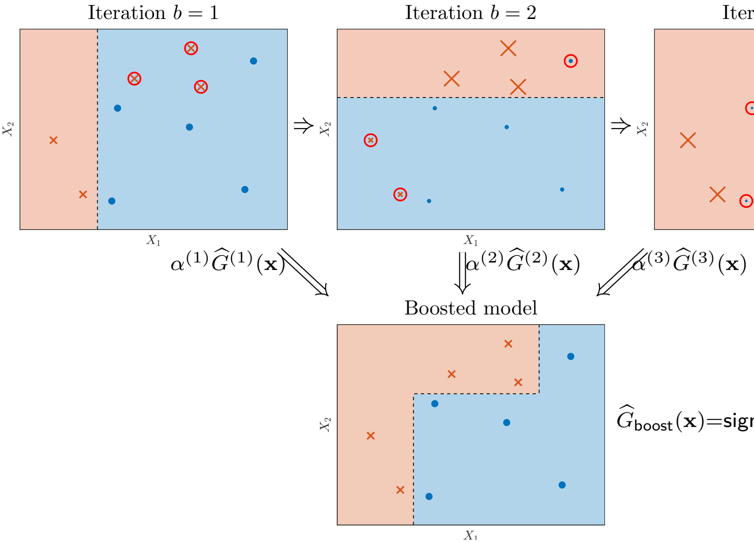
ex) Boosting illustration



ex) Boosting illustration



ex) Boosting illustration



$$\widehat{G}_{\text{boost}}(\mathbf{x}) = \text{sign}\left(\sum_{b=1}^{3} \alpha^{(b)}\widehat{G}^{(b)}(\mathbf{x})\right)$$

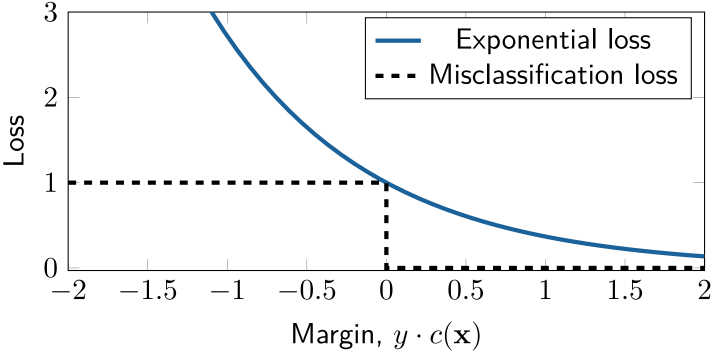
The technical details. . .

Q1: How do we reweight the data?

Q2: How are the coefficients $\alpha^{(1)}, \dots, \alpha^{(B)}$ computed?

Exponential loss

Loss functions for binary classifier $\widehat{G}(\mathbf{x}) = \text{sign}(c(\mathbf{x}))$.



Exponential loss function $L(y, c(\mathbf{x})) = \exp(-y \cdot c(\mathbf{x}))$ plotted vs. **margin** $y \cdot c(\mathbf{x})$. The misclassification loss $\mathbb{I}\{y \neq \widehat{G}(\mathbf{x})\} = \mathbb{I}\{y \cdot c(\mathbf{x}) < 0\}$ is plotted as comparison.

AdaBoost pseudo-code

1. Assign weights $w_i^1 = 1/n$ to all data points.
2. For $b = 1$ to B
   (a) Train a weak classifier $\widehat{G}^{(b)}(\mathbf{x})$ on the **weighted training data** $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$.
   (b) *Update the weights* $\{w_i^{b+1}\}_{i=1}^n$ *from* $\{w_i^b\}_{i=1}^n$*:*
      i. Increase weights for all points misclassified by $\widehat{G}^{(b)}(\mathbf{x})$.
      ii. Decrease weights for all points correctly classified by $\widehat{G}^{(b)}(\mathbf{x})$.

AdaBoost pseudo-code

1. Assign weights $w_i^1 = 1/n$ to all data points.
2. For $b = 1$ to B
   (a) Train a weak classifier $\widehat{G}^{(b)}(\mathbf{x})$ on the **weighted training data** $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$.
   (b) *Update the weights* $\{w_i^{b+1}\}_{i=1}^n$ *from* $\{w_i^b\}_{i=1}^n$*:*
      i. Compute weighted classification error
      ii. Compute classifier "confidence"
      iii. Compute new weights

UPPSALA UNIVERSITET

AdaBoost pseudo-code

1. Assign weights $w_i^1 = 1/n$ to all data points.
2. For $b = 1$ to B
   (a) Train a weak classifier $\widehat{G}^{(b)}(\mathbf{x})$ on the **weighted training data** $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$.
   (b) *Update the weights* $\{w_i^{b+1}\}_{i=1}^n$ *from* $\{w_i^b\}_{i=1}^n$*:*
      i. Compute $E_{\text{train}}^b = \sum_{i=1}^n w_i^b \mathbb{I}\{y_i \neq \widehat{G}^{(b)}(\mathbf{x}_i)\}$
      ii. Compute classifier "confidence"
      iii. Compute new weights

UPPSALA UNIVERSITET

AdaBoost pseudo-code

1. Assign weights $w_i^1 = 1/n$ to all data points.
2. For $b = 1$ to B
   (a) Train a weak classifier $\widehat{G}^{(b)}(\mathbf{x})$ on the **weighted training data** $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$.
   (b) *Update the weights* $\{w_i^{b+1}\}_{i=1}^n$ *from* $\{w_i^b\}_{i=1}^n$*:*
      i. Compute $E_{\text{train}}^b = \sum_{i=1}^n w_i^b \mathbb{I}\{y_i \neq \widehat{G}^{(b)}(\mathbf{x}_i)\}$
      ii. Compute $\alpha^b = 0.5 \log((1 - E_{\text{train}}^b)/E_{\text{train}}^b)$.
      iii. Compute new weights

AdaBoost pseudo-code

1. Assign weights $w_i^1 = 1/n$ to all data points.
2. For $b = 1$ to B
   (a) Train a weak classifier $\widehat{G}^{(b)}(\mathbf{x})$ on the **weighted training data** $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$.
   (b) *Update the weights* $\{w_i^{b+1}\}_{i=1}^n$ *from* $\{w_i^b\}_{i=1}^n$*:*
      i. Compute $E_{\text{train}}^b = \sum_{i=1}^n w_i^b \mathbb{I}\{y_i \neq \widehat{G}^{(b)}(\mathbf{x}_i)\}$
      ii. Compute $\alpha^b = 0.5 \log((1 - E_{\text{train}}^b)/E_{\text{train}}^b)$.
      iii. Compute $w_i^{b+1} = w_i^b \exp(-\alpha^{(b)} y_i \widehat{G}^{(b)}(\mathbf{x}_i))$, $i = 1, \ldots, n$

AdaBoost pseudo-code

1. Assign weights $w_i^1 = 1/n$ to all data points.
2. For $b = 1$ to B
   (a) Train a weak classifier $\widehat{G}^{(b)}(\mathbf{x})$ on the **weighted training data** $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$.
   (b) *Update the weights* $\{w_i^{b+1}\}_{i=1}^n$ *from* $\{w_i^b\}_{i=1}^n$*:*
      i. Compute $E_{\text{train}}^b = \sum_{i=1}^n w_i^b \mathbb{I}\{y_i \neq \widehat{G}^{(b)}(\mathbf{x}_i)\}$
      ii. Compute $\alpha^b = 0.5 \log((1 - E_{\text{train}}^b)/E_{\text{train}}^b)$.
      iii. Compute $w_i^{b+1} = w_i^b \exp(-\alpha^{(b)} y_i \widehat{G}^{(b)}(\mathbf{x}_i))$, $i = 1, \ldots, n$
      iv. *Normalize.* Set $w_i^{b+1} \leftarrow w_i^{b+1} / \sum_{j=1}^n w_j^{b+1}$, for $i = 1, \ldots, n$.

UPPSALA UNIVERSITET

AdaBoost pseudo-code

1. Assign weights $w_i^1 = 1/n$ to all data points.
2. For $b = 1$ to B
   (a) Train a weak classifier $\widehat{G}^{(b)}(\mathbf{x})$ on the **weighted training data** $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$.
   (b) *Update the weights* $\{w_i^{b+1}\}_{i=1}^n$ *from* $\{w_i^b\}_{i=1}^n$*:*
      i. Compute $E_{\text{train}}^b = \sum_{i=1}^n w_i^b \mathbb{I}\{y_i \neq \widehat{G}^{(b)}(\mathbf{x}_i)\}$
      ii. Compute $\alpha^b = 0.5 \log((1 - E_{\text{train}}^b)/E_{\text{train}}^b)$.
      iii. Compute $w_i^{b+1} = w_i^b \exp(-\alpha^{(b)} y_i \widehat{G}^{(b)}(\mathbf{x}_i))$, $i = 1, \ldots, n$
      iv. *Normalize.* Set $w_i^{b+1} \leftarrow w_i^{b+1} / \sum_{j=1}^n w_j^{b+1}$, for $i = 1, \ldots, n$.
3. Output $\widehat{G}_{\text{boost}}^{(B)}(\mathbf{x}) = \operatorname{sign}\left(\sum_{b=1}^B \alpha^{(b)} \widehat{G}^{(b)}(\mathbf{x})\right)$.

AdaBoost pseudo-code

1. Assign weights $w_i^1 = 1/n$ to all data points.
2. For $b = 1$ to B
   (a) Train a weak classifier $\widehat{G}^{(b)}(\mathbf{x})$ on the **weighted training data** $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$.
   (b) *Update the weights* $\{w_i^{b+1}\}_{i=1}^n$ *from* $\{w_i^b\}_{i=1}^n$*:*
      i. Compute $E_{\text{train}}^b = \sum_{i=1}^n w_i^b \mathbb{I}\{y_i \neq \widehat{G}^{(b)}(\mathbf{x}_i)\}$
      ii. Compute $\alpha^b = 0.5 \log((1 - E_{\text{train}}^b)/E_{\text{train}}^b)$.
      iii. Compute $w_i^{b+1} = w_i^b \exp(-\alpha^{(b)} y_i \widehat{G}^{(b)}(\mathbf{x}_i))$, $i = 1, \dots, n$
      iv. *Normalize.* Set $w_i^{b+1} \leftarrow w_i^{b+1} / \sum_{j=1}^n w_j^{b+1}$, for $i = 1, \dots, n$.
3. Output $\widehat{G}_{\text{boost}}^{(B)}(\mathbf{x}) = \text{sign}\left(\sum_{b=1}^B \alpha^{(b)} \widehat{G}^{(b)}(\mathbf{x})\right)$.

Y. Freund and R. E. Schapire. **Experiments with a New Boosting Algorithm**. *Proceedings of the 13th International Conference on Machine Learning (ICML)*. Bari, Italy, 1996.

2003 Gödel Prize

Boosting vs. bagging

Bagging	**Boosting**
Learns base models in parallel	Learns base models sequentially
Uses bootstrapped datasets	Uses reweighted datasets
Does not overfit as B becomes large	Can overfit as B becomes large
Reduces variance but not bias (requires deep trees as base models)	Also reduces bias! (works well with shallow trees)

Boosting vs. bagging

Bagging	**Boosting**
Learns base models in parallel	Learns base models sequentially
Uses bootstrapped datasets	Uses reweighted datasets
Does not overfit as B becomes large	Can overfit as B becomes large
Reduces variance but not bias (requires deep trees as base models)	Also reduces bias! (works well with shallow trees)

N.B. Boosting does ***not*** require each base model to have low bias. Thus, a shallow classification tree (say, 4-8 terminal nodes) or even a tree with a single split (2 terminal nodes, a “stump”) is often sufficient.

A few concepts to summarize lecture 6

CART: Classification and regression trees. A class of nonparametric methods based on partitioning the input space into regions and fitting a simple model for each region.

Recursive binary splitting: A greedy method for partitioning the input space into “boxes” aligned with the coordinate axes.

Gini index and deviance: Commonly used error measures for constructing classification trees.

Ensemble methods: Umbrella term for methods that average or combine multiple models.

Bagging: Bootstrap aggregating. An ensemble method based on the statistical bootstrap.

Random forests: Bagging of trees, combined with random feature selection for further variance reduction (and computational gains).