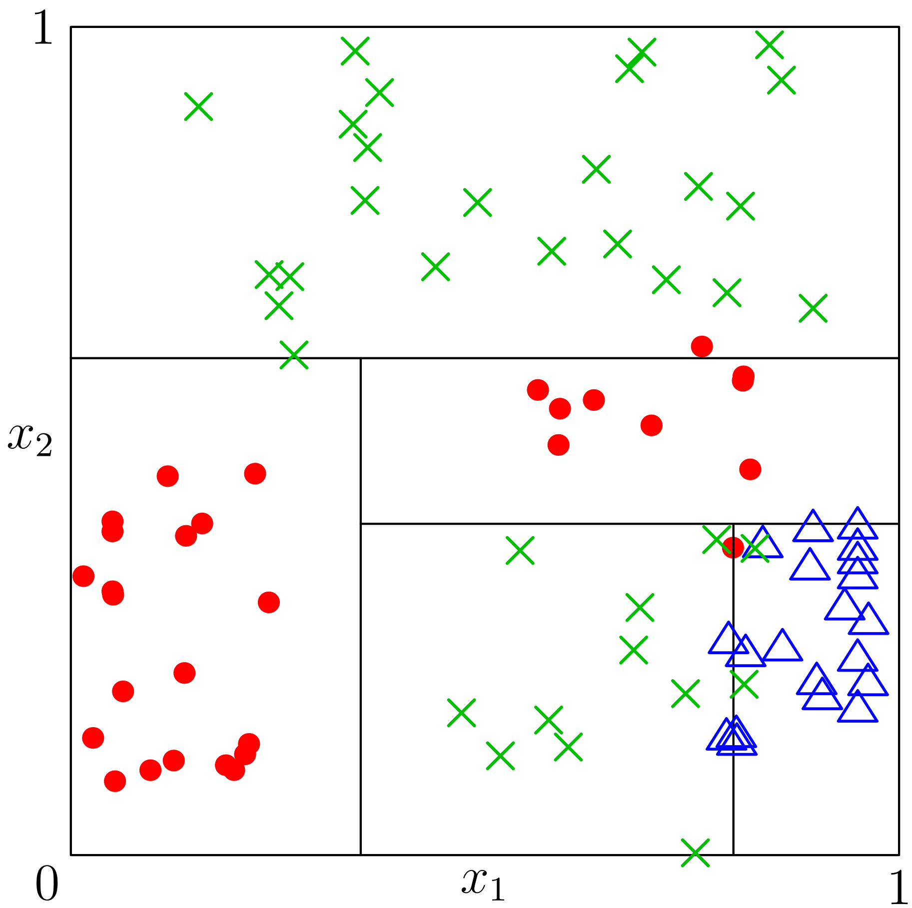
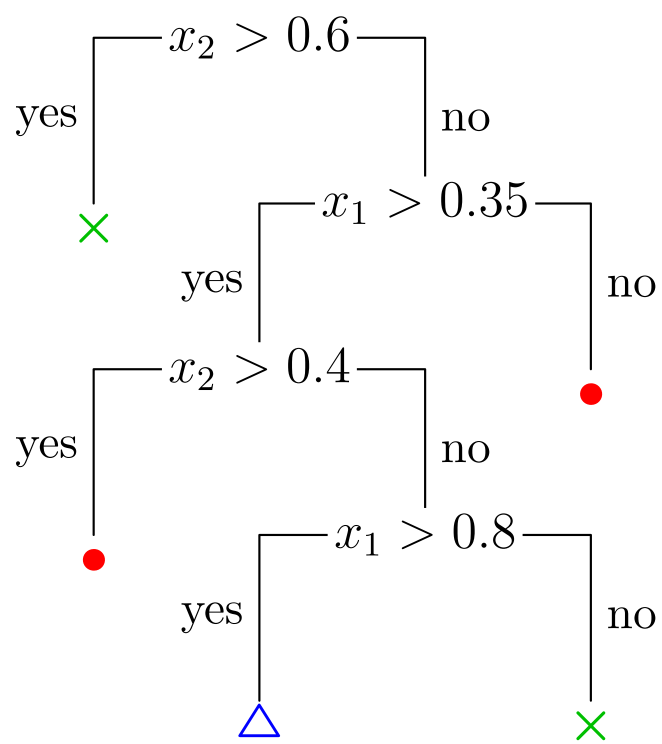
Advanced Machine Learning

Ensemble Methods

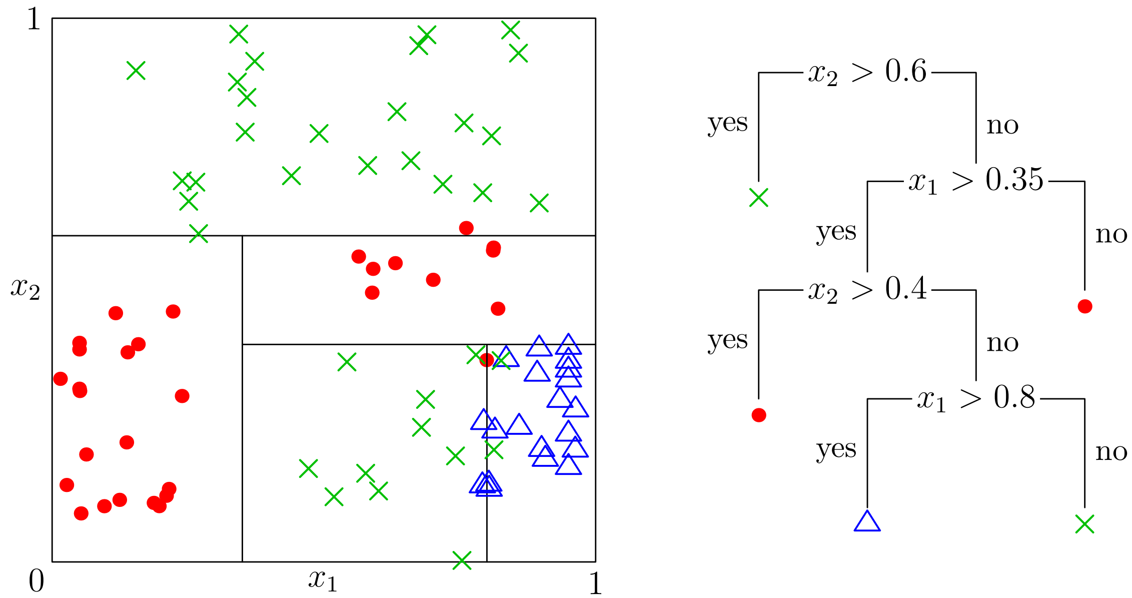




Decision Trees, Bagging, Boosting

Outline

1. **Decision Trees**
2. Bagging
3. Boosting



Removing Variance By Averaging

- We can reduce the variance and hence improve our generalisation error by averaging over different learning machines
- There are a number of different techniques for doing this that go by the name of **ensemble methods** or **ensemble learning**
- This trick can be used with many different learning machines, but is clearly most practical for machine that can be trained quickly

Removing Variance By Averaging

- We can reduce the variance and hence improve our generalisation error by averaging over different learning machines

- There are a number of different techniques for doing this that go by the name of **ensemble methods** or **ensemble learning**

- This trick can be used with many different learning machines, but is clearly most practical for machine that can be trained quickly

Removing Variance By Averaging

- We can reduce the variance and hence improve our generalisation error by averaging over different learning machines
- There are a number of different techniques for doing this that go by the name of **ensemble methods** or **ensemble learning**
- This trick can be used with many different learning machines, but is clearly most practical for machine that can be trained quickly

Removing Variance By Averaging

- We can reduce the variance and hence improve our generalisation error by averaging over different learning machines
- There are a number of different techniques for doing this that go by the name of **ensemble methods** or **ensemble learning**
- This trick can be used with many different learning machines, but is clearly most practical for machine that can be trained quickly
- (nevertheless, even for deep learning taking the average response of many machines is usually done to win competitions)

Ensembling of Decision Trees

- One set of algorithms where ensembling are common place are decision trees

- These are particularly good for handling messy data
  - ⋆ categorical data
  - ⋆ mixture of data types
  - ⋆ missing data
  - ⋆ large data sets
  - ⋆ multiclass

- In many competitions ensembled trees, particularly *random forests* and *gradient boosting* beats all other techniques

Ensembling of Decision Trees

- One set of algorithms where ensembling are common place are decision trees
- These are particularly good for handling messy data
  - ⋆ categorical data
  - ⋆ mixture of data types
  - ⋆ missing data
  - ⋆ large data sets
  - ⋆ multiclass
- In many competitions ensembled trees, particularly *random forests* and *gradient boosting* beats all other techniques

Ensembling of Decision Trees

- One set of algorithms where ensembling are common place are decision trees
- These are particularly good for handling messy data
  - ⋆ categorical data
  - ⋆ mixture of data types
  - ⋆ missing data
  - ⋆ large data sets
  - ⋆ multiclass
- In many competitions ensembled trees, particularly *random forests* and *gradient boosting* beats all other techniques

Ensembling of Decision Trees

- One set of algorithms where ensembling are common place are decision trees
- These are particularly good for handling messy data
  - ⋆ categorical data
  - ⋆ mixture of data types
  - ⋆ missing data
  - ⋆ large data sets
  - ⋆ multiclass
- In many competitions ensembled trees, particularly *random forests* and *gradient boosting* beats all other techniques

Ensembling of Decision Trees

- One set of algorithms where ensembling are common place are decision trees
- These are particularly good for handling messy data
  - ⋆ categorical data
  - ⋆ mixture of data types
  - ⋆ missing data
  - ⋆ large data sets
  - ⋆ multiclass
- In many competitions ensembled trees, particularly *random forests* and *gradient boosting* beats all other techniques

Ensembling of Decision Trees

- One set of algorithms where ensembling are common place are decision trees

- These are particularly good for handling messy data
  - ⋆ categorical data
  - ⋆ mixture of data types
  - ⋆ missing data
  - ⋆ large data sets
  - ⋆ multiclass

- In many competitions ensembled trees, particularly *random forests* and *gradient boosting* beats all other techniques

Ensembling of Decision Trees

- One set of algorithms where ensembling are common place are decision trees
- These are particularly good for handling messy data
  - ⋆ categorical data
  - ⋆ mixture of data types
  - ⋆ missing data
  - ⋆ large data sets
  - ⋆ multiclass
- In many competitions ensembled trees, particularly *random forests* and *gradient boosting* beats all other techniques

Decision Trees

- Decision trees builds a binary tree to partition the data, $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) | i = 1, \ldots, m\}$, into the leaves of the tree

- Each decision rule depends on a single feature

- At each step the rule is chosen that maximise the "$purity$" of the leaf nodes

- Decisions can be made on numerical values or categories

Decision Trees

- Decision trees builds a binary tree to partition the data, $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) | i = 1, \ldots, m\}$, into the leaves of the tree
- Each decision rule depends on a single feature
- At each step the rule is chosen that maximise the "$purity$" of the leaf nodes
- Decisions can be made on numerical values or categories

Decision Trees

- Decision trees builds a binary tree to partition the data, $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) | i = 1, \ldots, m\}$, into the leaves of the tree
- Each decision rule depends on a single feature
- At each step the rule is chosen that maximise the "$purity$" of the leaf nodes
- Decisions can be made on numerical values or categories

Decision Trees

- Decision trees builds a binary tree to partition the data, $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) | i = 1, \ldots, m\}$, into the leaves of the tree

- Each decision rule depends on a single feature

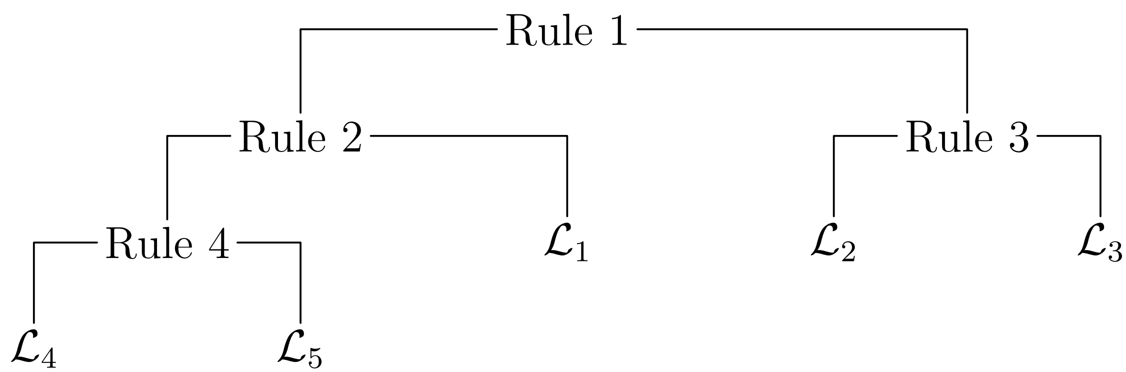
- At each step the rule is chosen that maximise the "$purity$" of the leaf nodes

- Decisions can be made on numerical values or categories

Partitioning

- Consider a classification problems with examples $(\boldsymbol{x}, y)$ belonging to some classes $y \in \mathcal{C}$

- The data is partitioned by the tree into leaves



- The proportion of data points in leaf $\mathcal{L}$ belonging to class c is

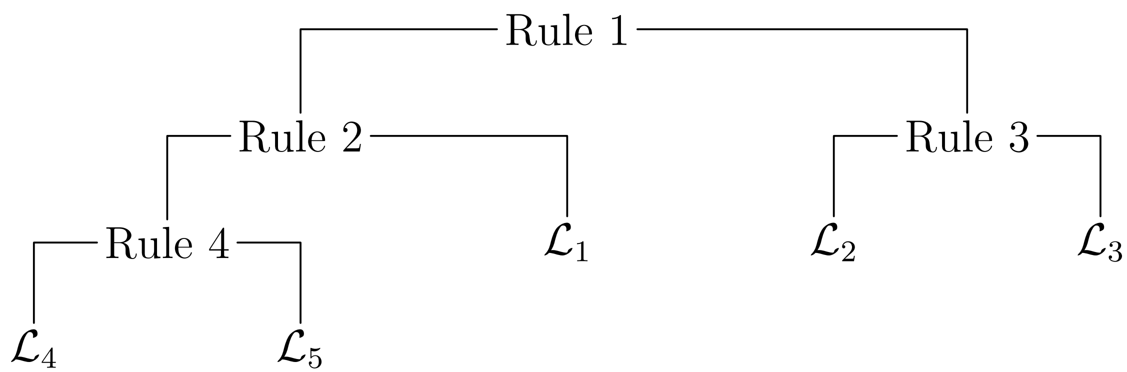
$$p_c(\mathcal{L}) = \frac{1}{|\mathcal{L}|} \sum_{(\boldsymbol{x},y)\in\mathcal{L}} [\![y = c]\!]$$

where $[\![y = c]\!] = 1$ if $y = c$ and 0 otherwise

Partitioning

- Consider a classification problems with examples $(\boldsymbol{x}, y)$ belonging to some classes $y \in \mathcal{C}$

- The data is partitioned by the tree into leaves



- The proportion of data points in leaf $\mathcal{L}$ belonging to class c is

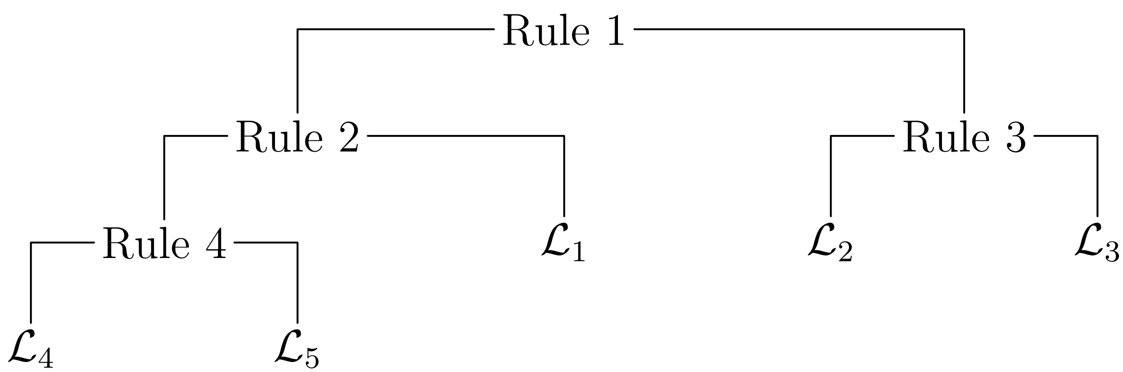
$$p_c(\mathcal{L}) = \frac{1}{|\mathcal{L}|} \sum_{(\boldsymbol{x}, y) \in \mathcal{L}} [\![y = c]\!]$$

where $[\![y = c]\!] = 1$ if $y = c$ and 0 otherwise

Partitioning

- Consider a classification problems with examples $(\boldsymbol{x}, y)$ belonging to some classes $y \in \mathcal{C}$

- The data is partitioned by the tree into leaves



- The proportion of data points in leaf $\mathcal{L}$ belonging to class c is

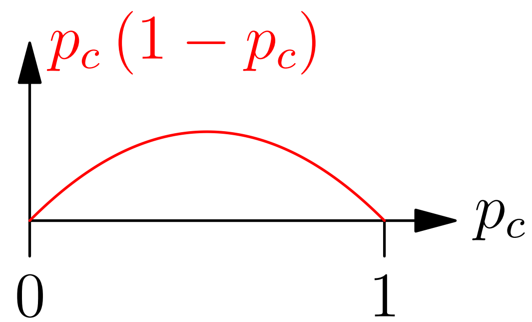
$$p_c(\mathcal{L}) = \frac{1}{|\mathcal{L}|} \sum_{(\boldsymbol{x},y)\in\mathcal{L}} [\![y = c]\!]$$

where $[\![y = c]\!] = 1$ if $y = c$ and 0 otherwise

Leaf Purity

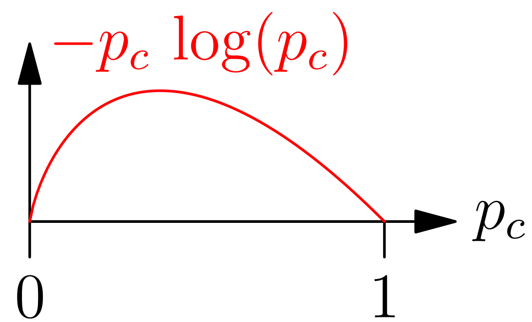
- Two different purity measures, $Q_m(\mathcal{L})$, for a leaf node $\mathcal{L}$ are commonly used
  - **Gini index**

$$Q_m^g(\mathcal{L}) = \sum_{c \in \mathcal{C}} p_c(\mathcal{L})\ (1 - p_c(\mathcal{L}))$$



  - **Cross-entropy**

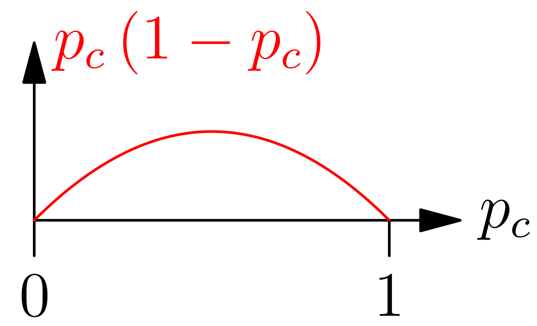
$$Q_m^e(\mathcal{L}) = -\sum_{c \in \mathcal{C}} p_c(\mathcal{L})\ \log(p_c(\mathcal{L}))$$



Leaf Purity

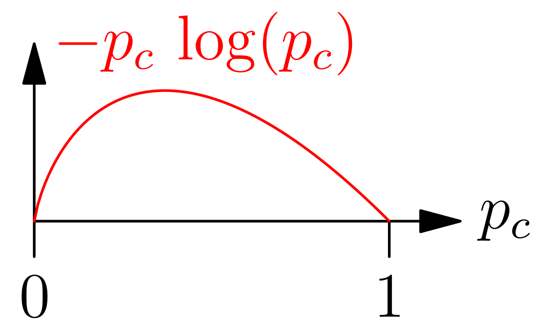
- Two different purity measures, $Q_m(\mathcal{L})$, for a leaf node $\mathcal{L}$ are commonly used
  - ⋆ **Gini index**

$$Q_m^g(\mathcal{L}) = \sum_{c\in\mathcal{C}} p_c(\mathcal{L})\ (1 - p_c(\mathcal{L}))$$



  - ⋆ **Cross-entropy**

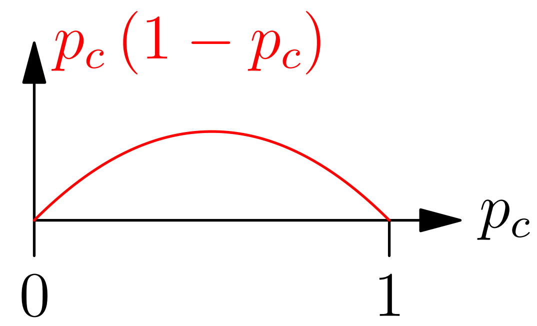
$$Q_m^e(\mathcal{L}) = -\sum_{c\in\mathcal{C}} p_c(\mathcal{L})\ \log(p_c(\mathcal{L}))$$



Leaf Purity

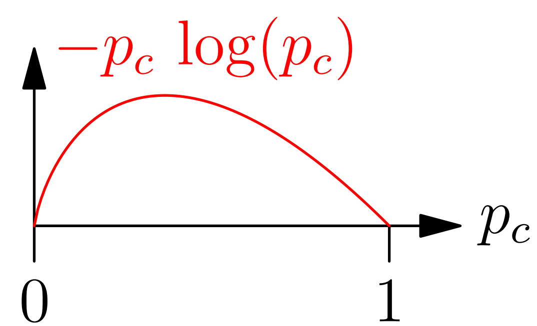
- Two different purity measures, $Q_m(\mathcal{L})$, for a leaf node $\mathcal{L}$ are commonly used
  - ⋆ **Gini index**

$$Q^g_m(\mathcal{L}) = \sum_{c \in \mathcal{C}} p_c(\mathcal{L}) \, (1 - p_c(\mathcal{L}))$$

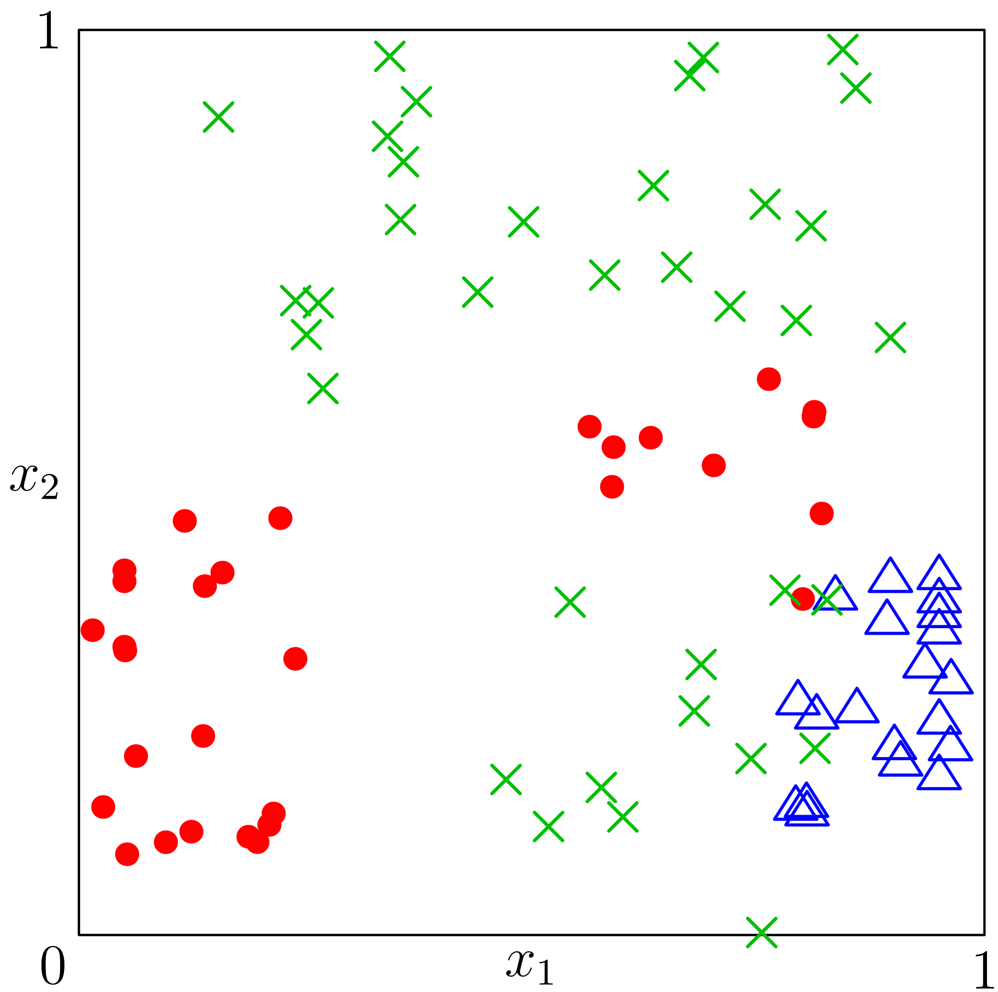


  - ⋆ **Cross-entropy**

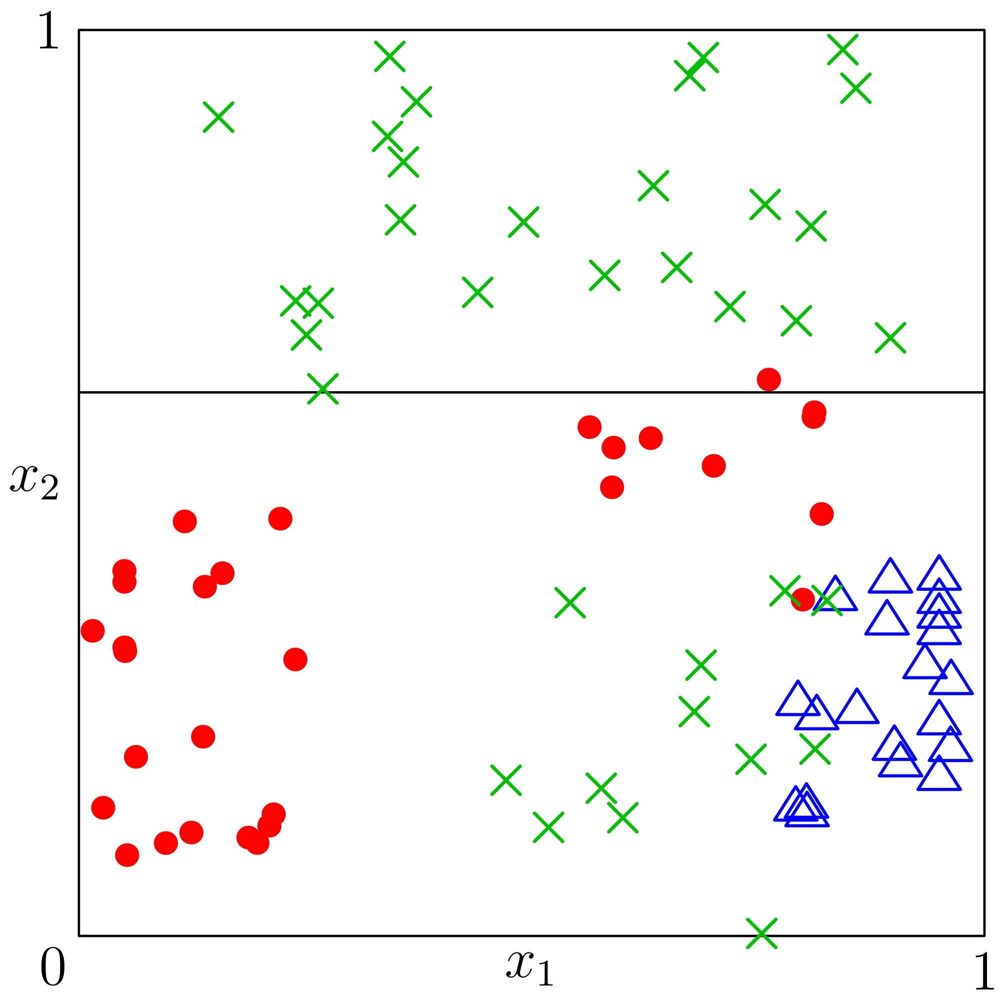
$$Q^e_m(\mathcal{L}) = -\sum_{c \in \mathcal{C}} p_c(\mathcal{L}) \, \log(p_c(\mathcal{L}))$$

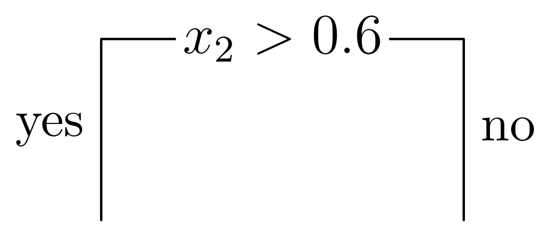


Building Decision Trees

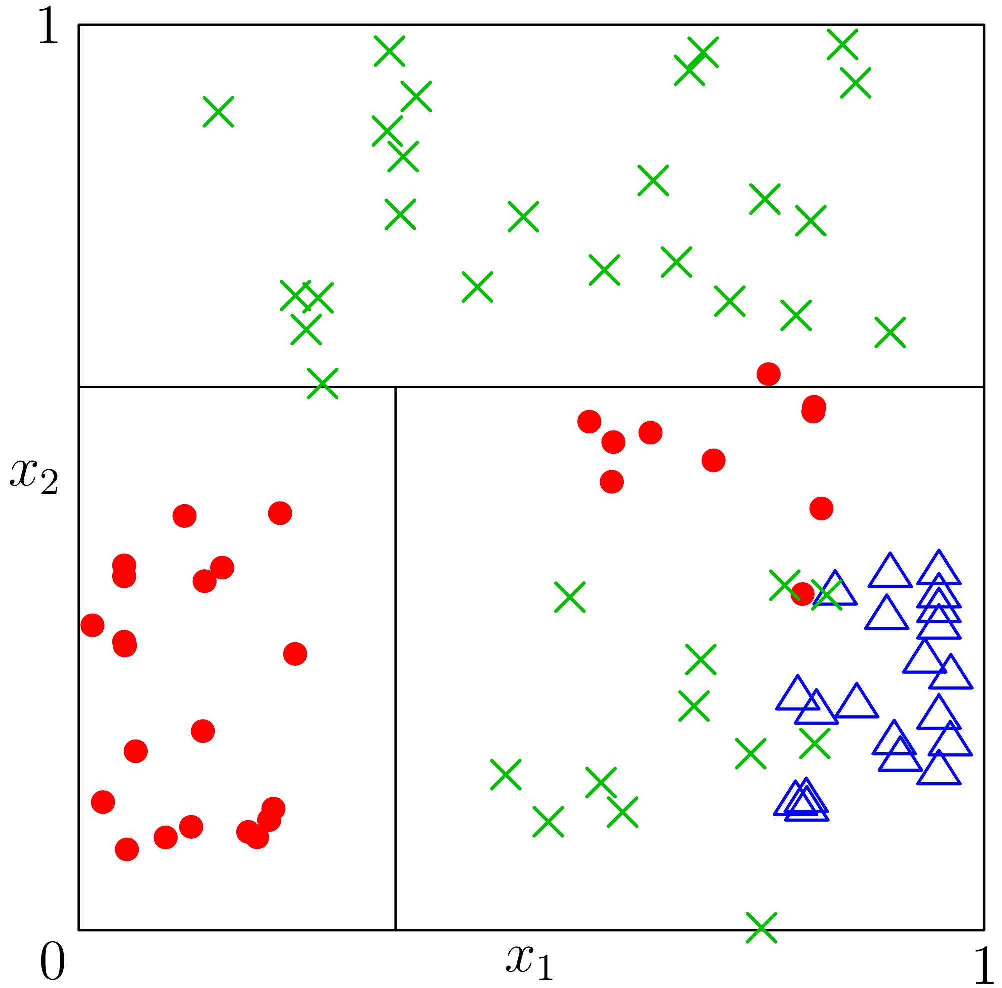


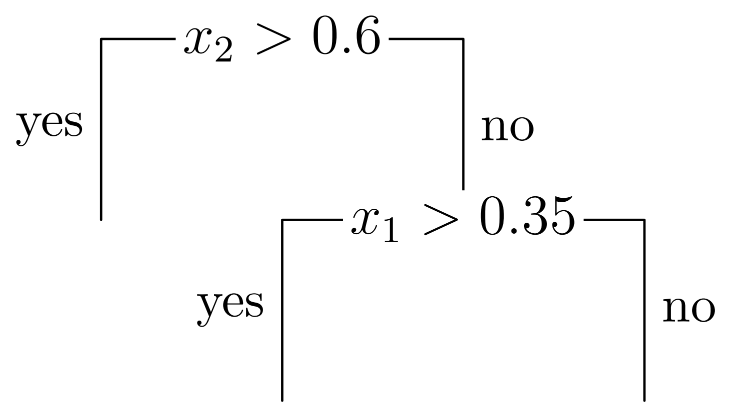
Building Decision Trees



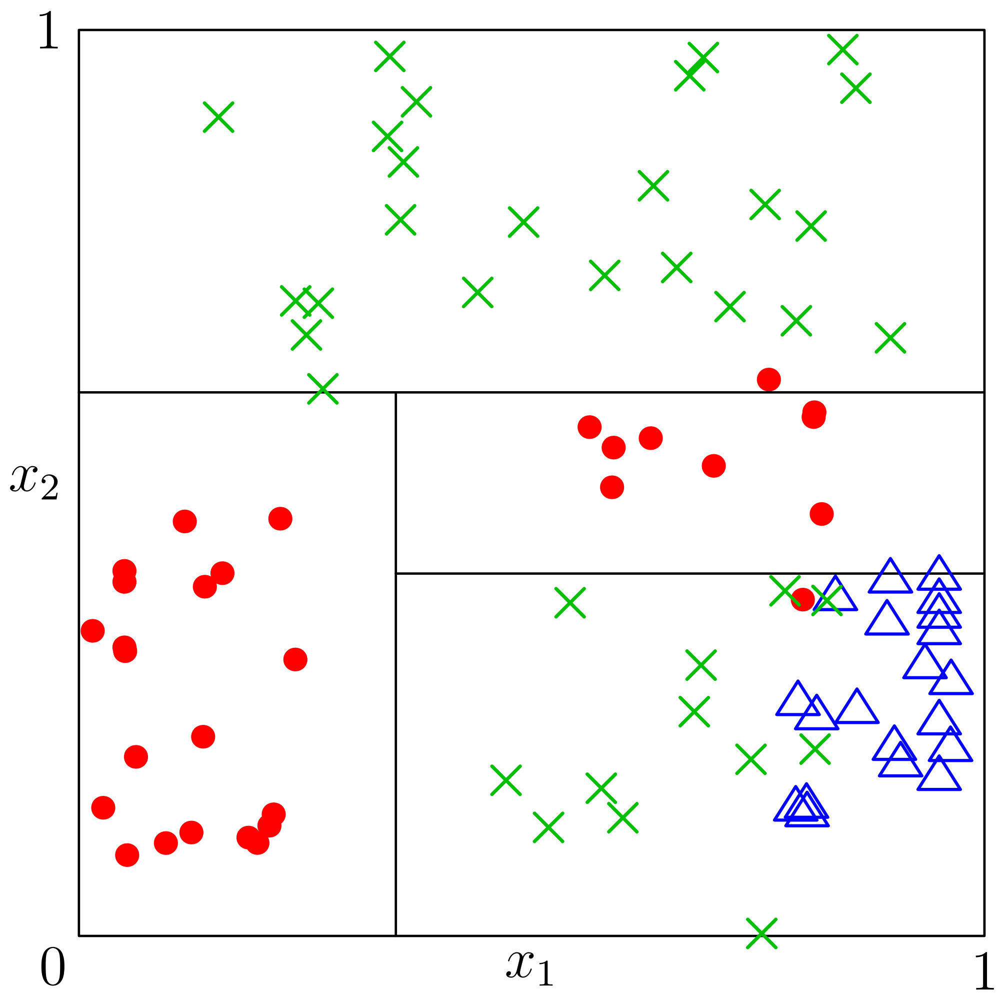


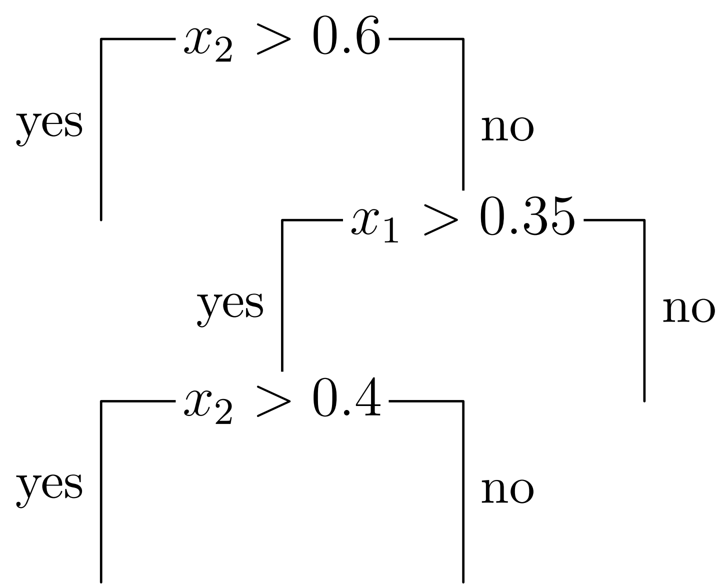
Building Decision Trees



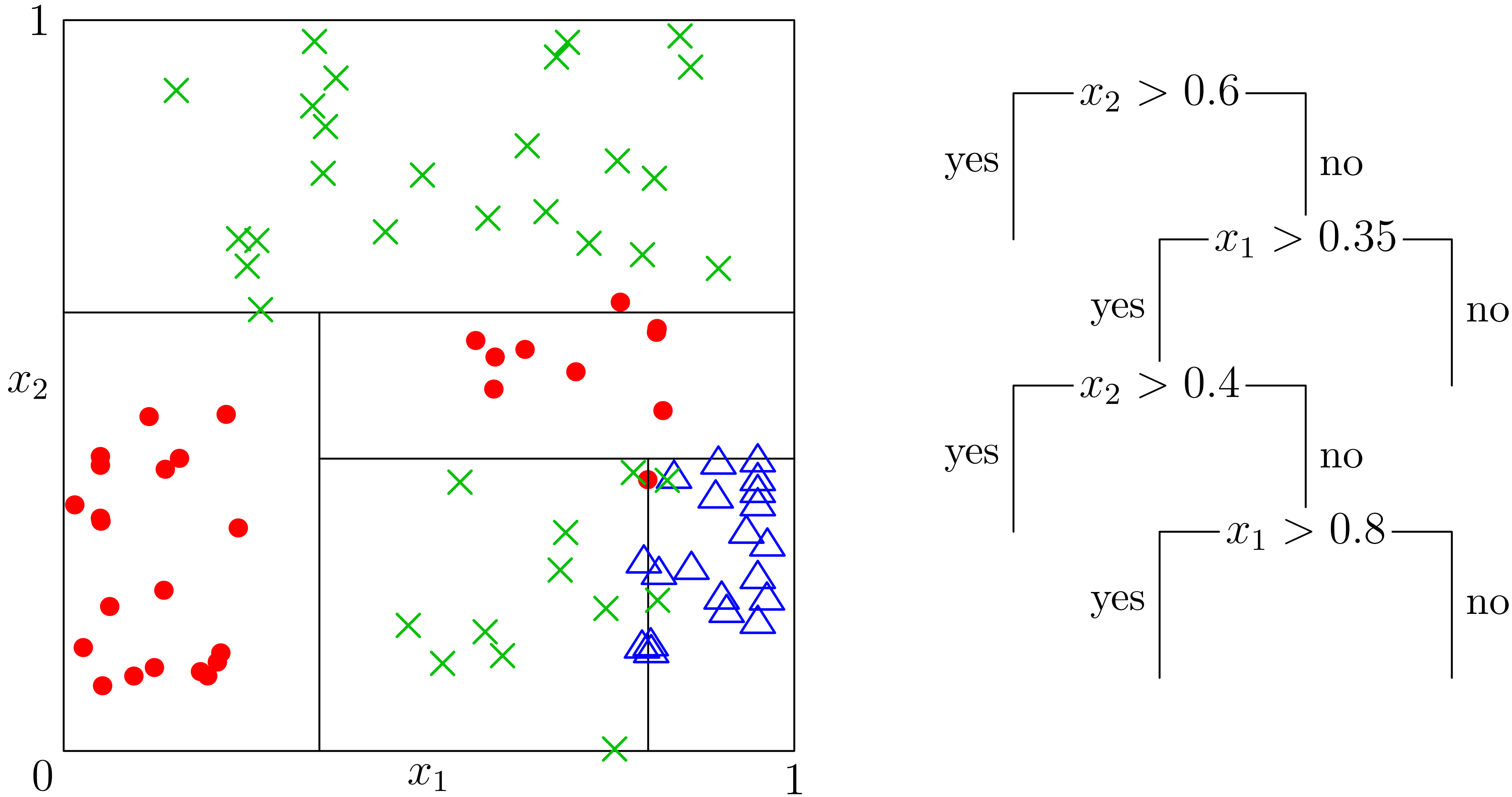


Building Decision Trees

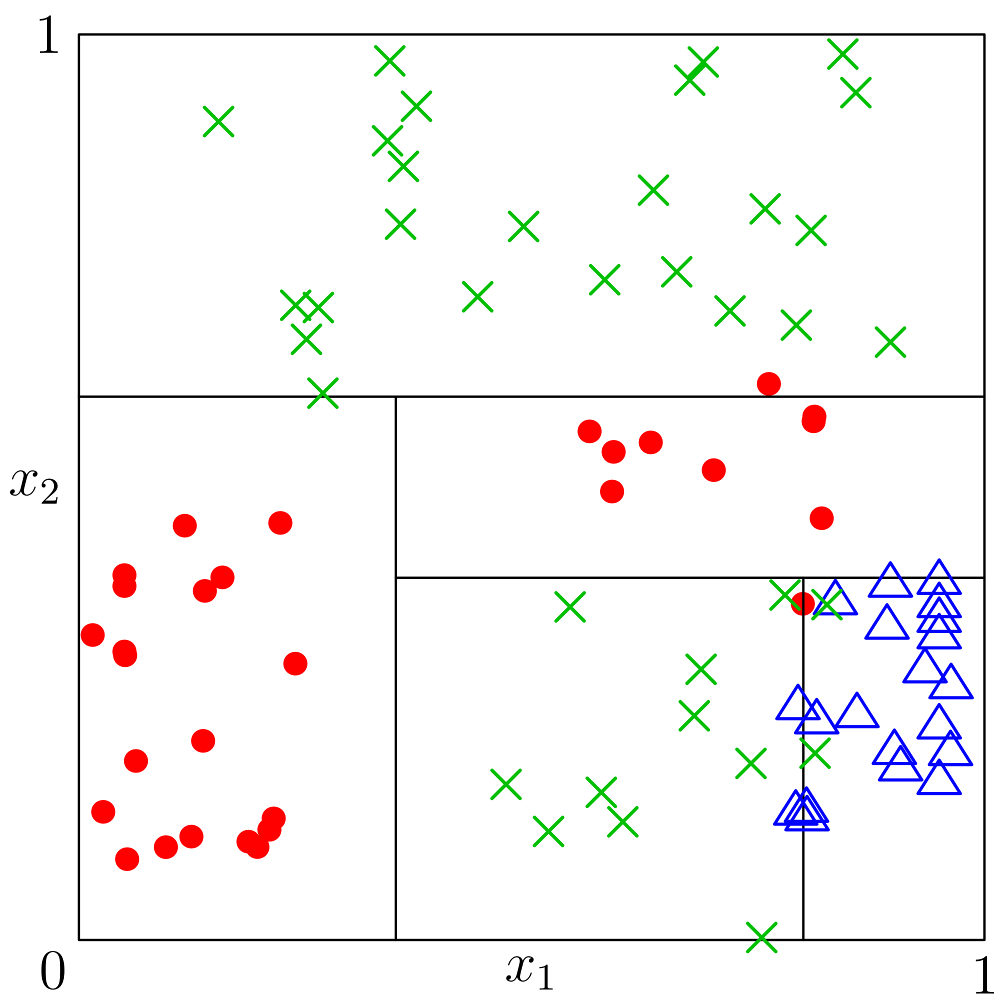


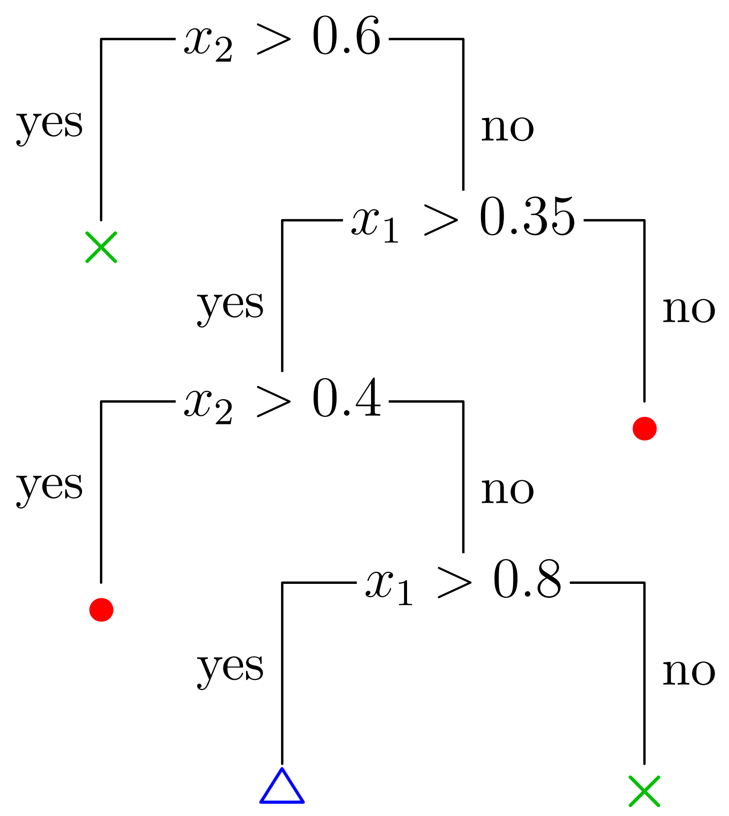


Building Decision Trees



Building Decision Trees





Observations

- Decision trees are very useful for exploring new data sets—the tree shows what features are most important

- Decision trees can also be used for regression problems
  - ⋆ Approximate function by a series of steps
  - ⋆ Reduce variance between data points assigned to leaf nodes

- CART is a classic implementation that builds **C**lassification **A**nd **R**egression **T**rees

- Decision trees depend strongly on the early decisions and so vary a lot for slightly different data sets

Observations

- Decision trees are very useful for exploring new data sets—the tree shows what features are most important
- Decision trees can also be used for regression problems
  - ⋆ Approximate function by a series of steps
  - ⋆ Reduce variance between data points assigned to leaf nodes
- CART is a classic implementation that builds **C**lassification **A**nd **R**egression **T**rees
- Decision trees depend strongly on the early decisions and so vary a lot for slightly different data sets

Observations

- Decision trees are very useful for exploring new data sets—the tree shows what features are most important
- Decision trees can also be used for regression problems
  - ⋆ Approximate function by a series of steps
  - ⋆ Reduce variance between data points assigned to leaf nodes
- CART is a classic implementation that builds **C**lassification **A**nd **R**egression **T**rees
- Decision trees depend strongly on the early decisions and so vary a lot for slightly different data sets

Observations

- Decision trees are very useful for exploring new data sets—the tree shows what features are most important
- Decision trees can also be used for regression problems
  - ⋆ Approximate function by a series of steps
  - ⋆ Reduce variance between data points assigned to leaf nodes
- CART is a classic implementation that builds **C**lassification **A**nd **R**egression **T**rees
- Decision trees depend strongly on the early decisions and so vary a lot for slightly different data sets

Observations

- Decision trees are very useful for exploring new data sets—the tree shows what features are most important
- Decision trees can also be used for regression problems
  - ⋆ Approximate function by a series of steps
  - ⋆ Reduce variance between data points assigned to leaf nodes
- CART is a classic implementation that builds **C**lassification **A**nd **R**egression **T**rees
- Decision trees depend strongly on the early decisions and so vary a lot for slightly different data sets

Observations

- Decision trees are very useful for exploring new data sets—the tree shows what features are most important
- Decision trees can also be used for regression problems
  - ⋆ Approximate function by a series of steps
  - ⋆ Reduce variance between data points assigned to leaf nodes
- CART is a classic implementation that builds **C**lassification **A**nd **R**egression **T**rees
- Decision trees depend strongly on the early decisions and so vary a lot for slightly different data sets

Observations

- Decision trees are very useful for exploring new data sets—the tree shows what features are most important
- Decision trees can also be used for regression problems
  - ⋆ Approximate function by a series of steps
  - ⋆ Reduce variance between data points assigned to leaf nodes
- CART is a classic implementation that builds **C**lassification **A**nd **R**egression **T**rees
- Decision trees depend strongly on the early decisions and so vary a lot for slightly different data sets

Observations

- Decision trees are very useful for exploring new data sets—the tree shows what features are most important

- Decision trees can also be used for regression problems
  - ⋆ Approximate function by a series of steps
  - ⋆ Reduce variance between data points assigned to leaf nodes

- CART is a classic implementation that builds **C**lassification **A**nd **R**egression **T**rees

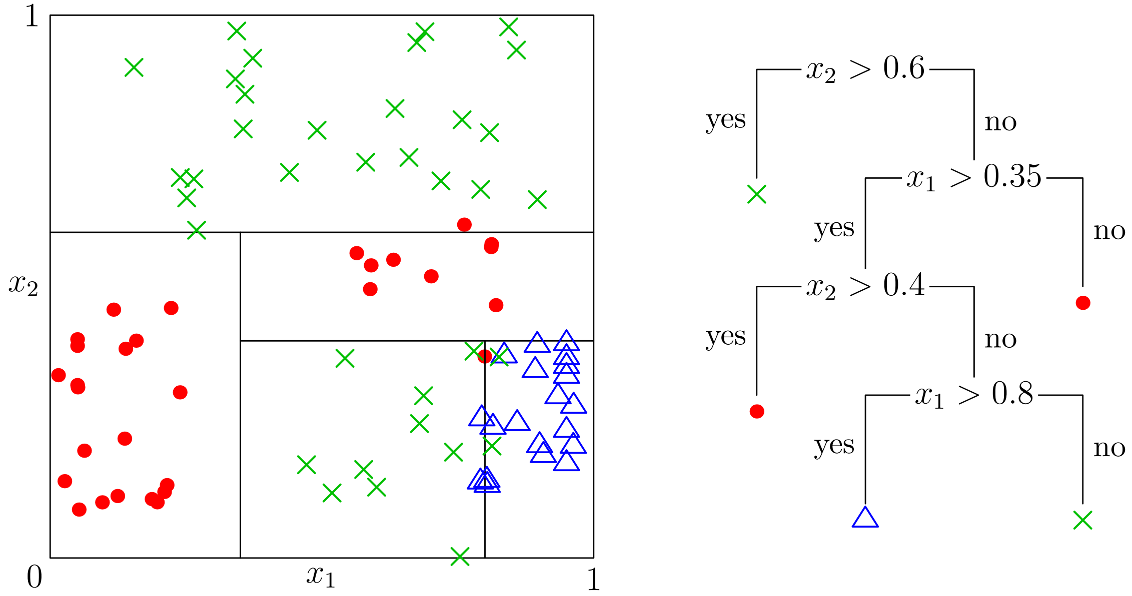
- Decision trees depend strongly on the early decisions and so vary a lot for slightly different data sets—high variance

Outline

1. Decision Trees

2. **Bagging**

3. Boosting



Error In The Means

- By taking the mean over many samples we can reduce out variance and thus improve our generalisation performance

- To get a feel for this consider estimating the mean of a random variable X, from a number of samples ($n = 5$ in the example below)

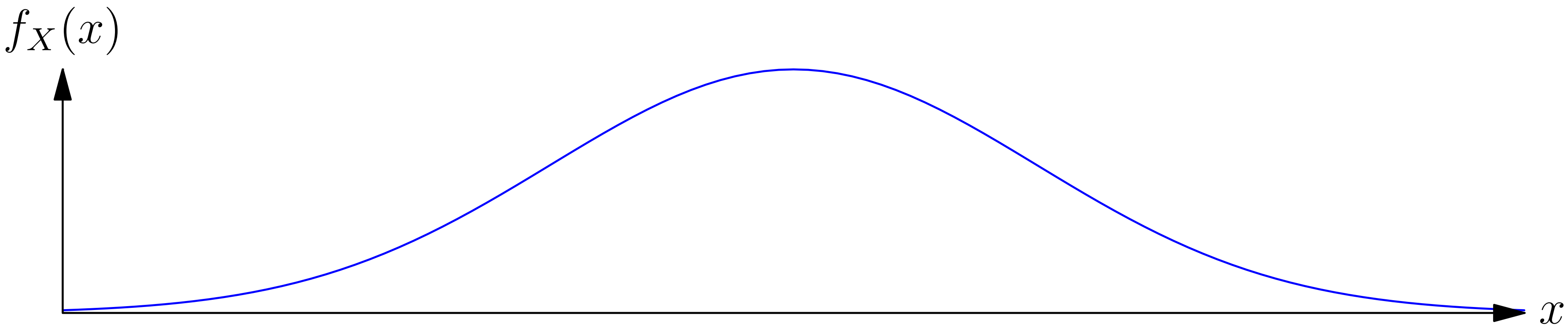
Error In The Means

- By taking the mean over many samples we can reduce out variance and thus improve our generalisation performance

- To get a feel for this consider estimating the mean of a random variable X, from a number of samples ($n = 5$ in the example below)

Error In The Means

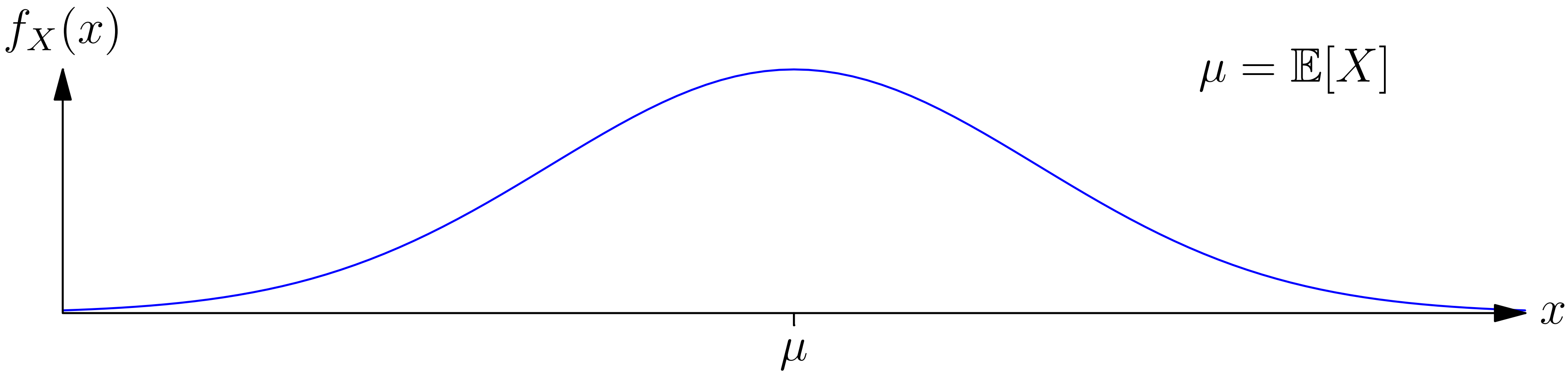
- By taking the mean over many samples we can reduce out variance and thus improve our generalisation performance
- To get a feel for this consider estimating the mean of a random variable X, from a number of samples ($n = 5$ in the example below)



Error In The Means

- By taking the mean over many samples we can reduce out variance and thus improve our generalisation performance

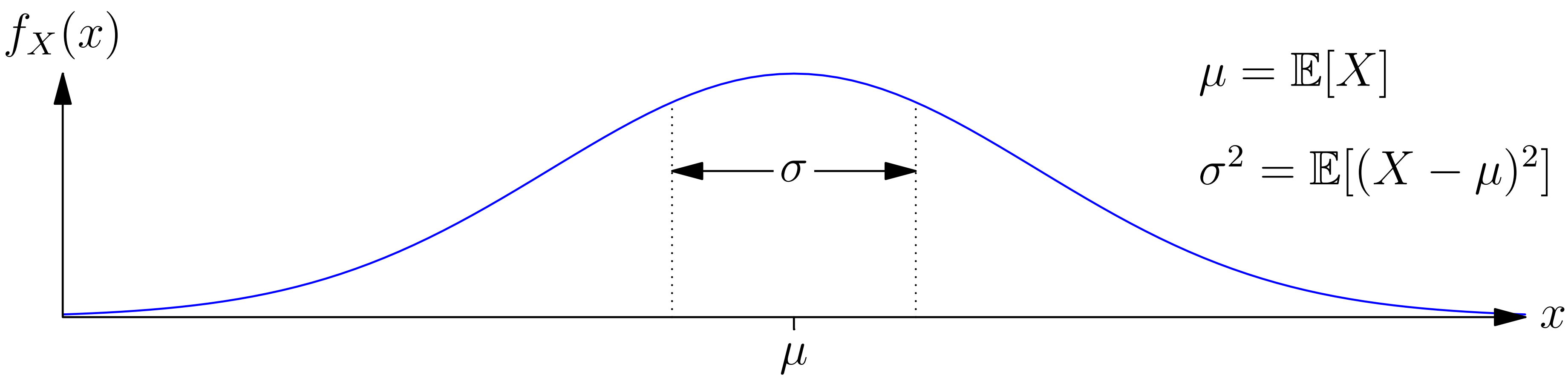
- To get a feel for this consider estimating the mean of a random variable X, from a number of samples ($n = 5$ in the example below)



Error In The Means

- By taking the mean over many samples we can reduce out variance and thus improve our generalisation performance

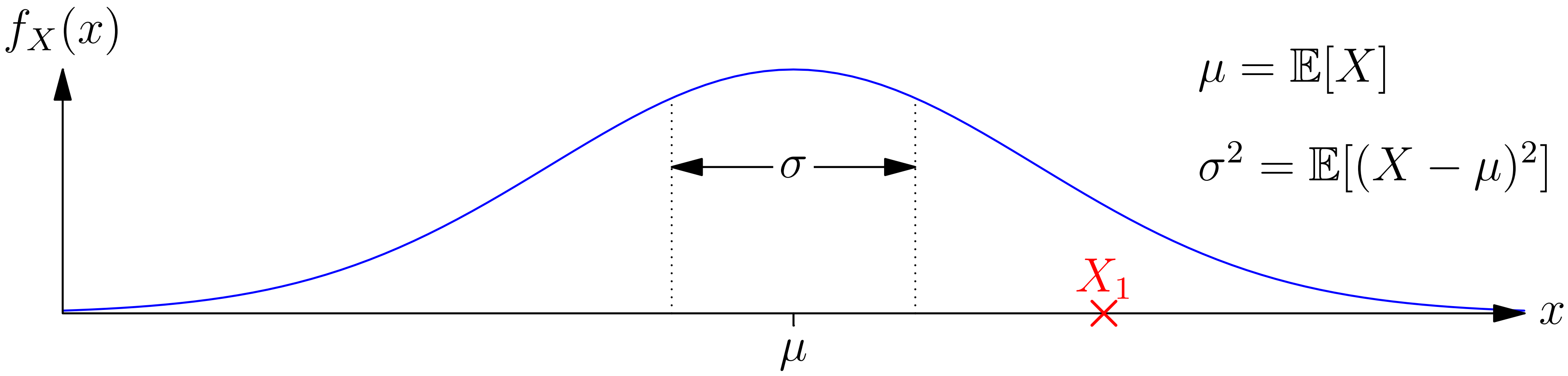
- To get a feel for this consider estimating the mean of a random variable X, from a number of samples ($n = 5$ in the example below)



Error In The Means

- By taking the mean over many samples we can reduce out variance and thus improve our generalisation performance

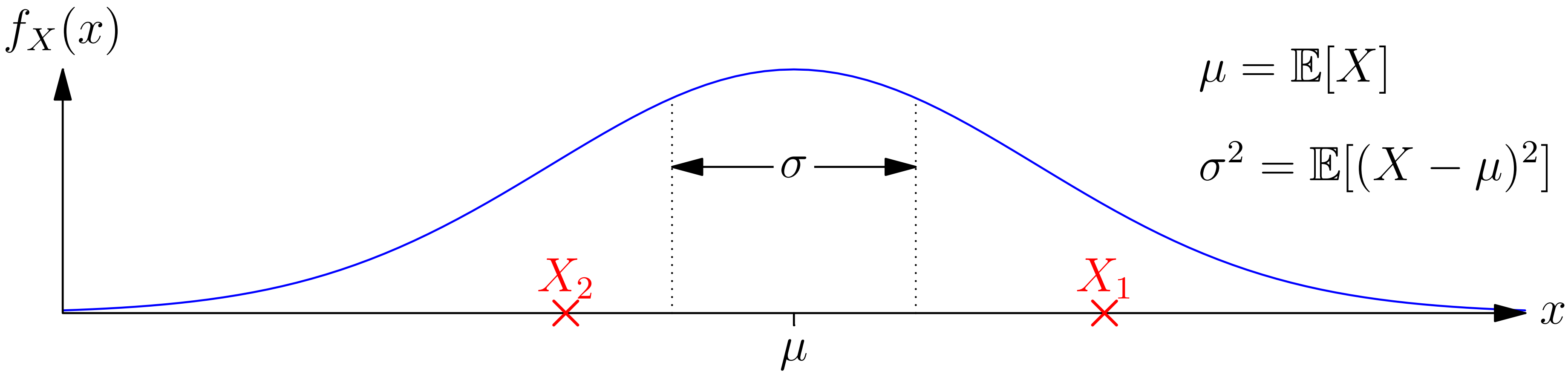
- To get a feel for this consider estimating the mean of a random variable X, from a number of samples ($n = 5$ in the example below)



Error In The Means

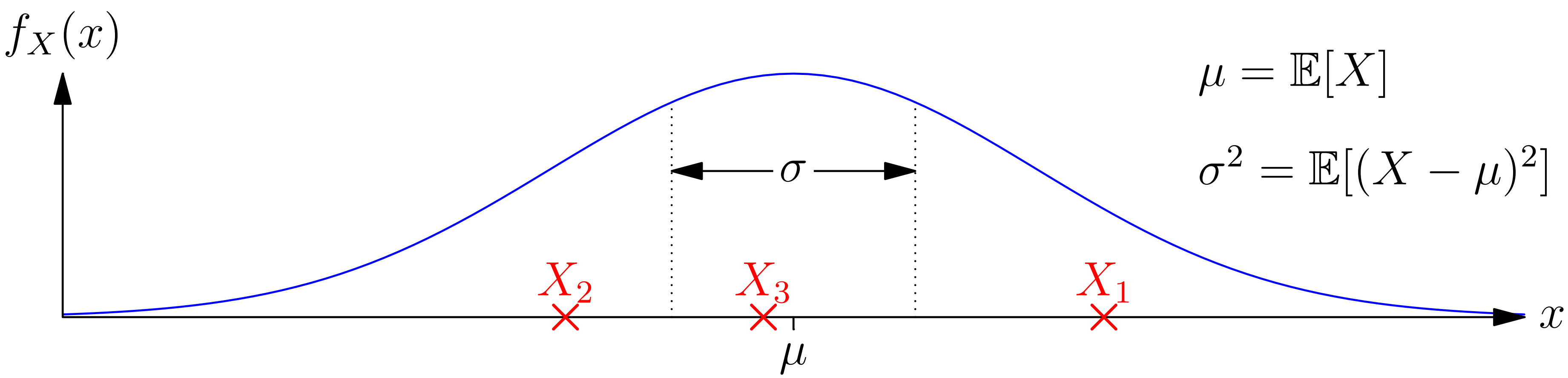
- By taking the mean over many samples we can reduce out variance and thus improve our generalisation performance

- To get a feel for this consider estimating the mean of a random variable X, from a number of samples ($n = 5$ in the example below)



Error In The Means

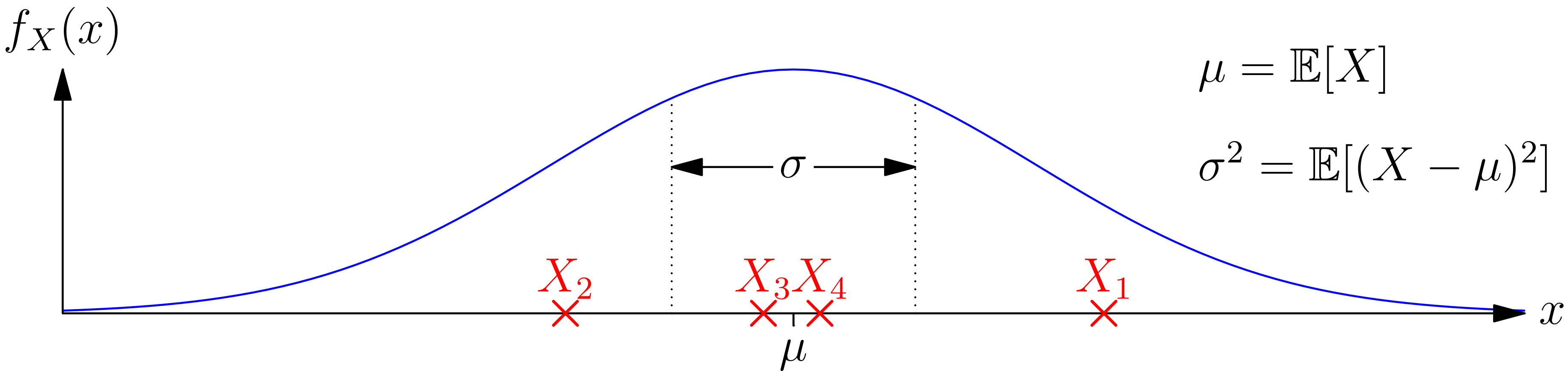
- By taking the mean over many samples we can reduce out variance and thus improve our generalisation performance
- To get a feel for this consider estimating the mean of a random variable X, from a number of samples ($n = 5$ in the example below)



Error In The Means

- By taking the mean over many samples we can reduce out variance and thus improve our generalisation performance

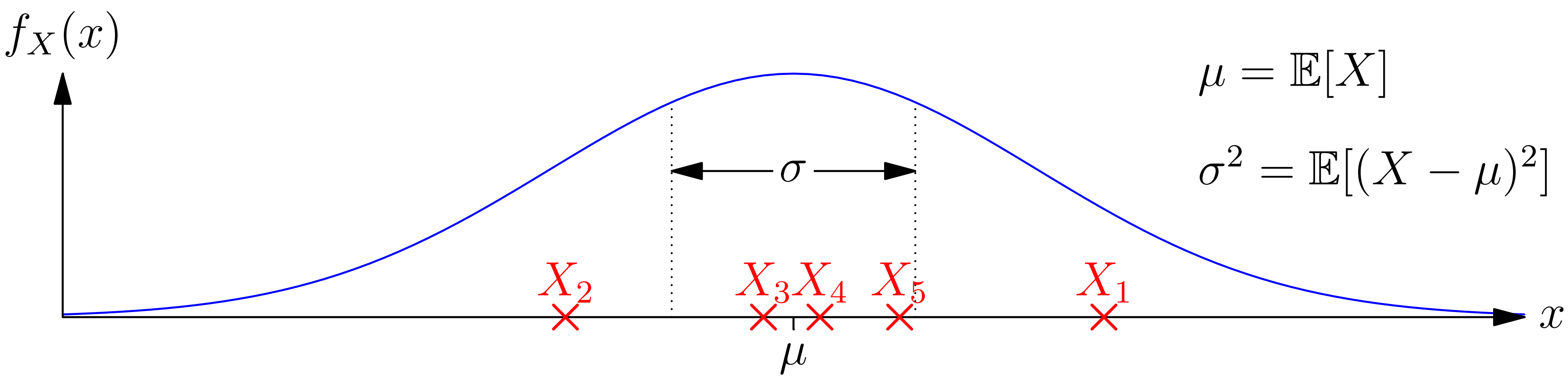
- To get a feel for this consider estimating the mean of a random variable X, from a number of samples ($n = 5$ in the example below)



Error In The Means

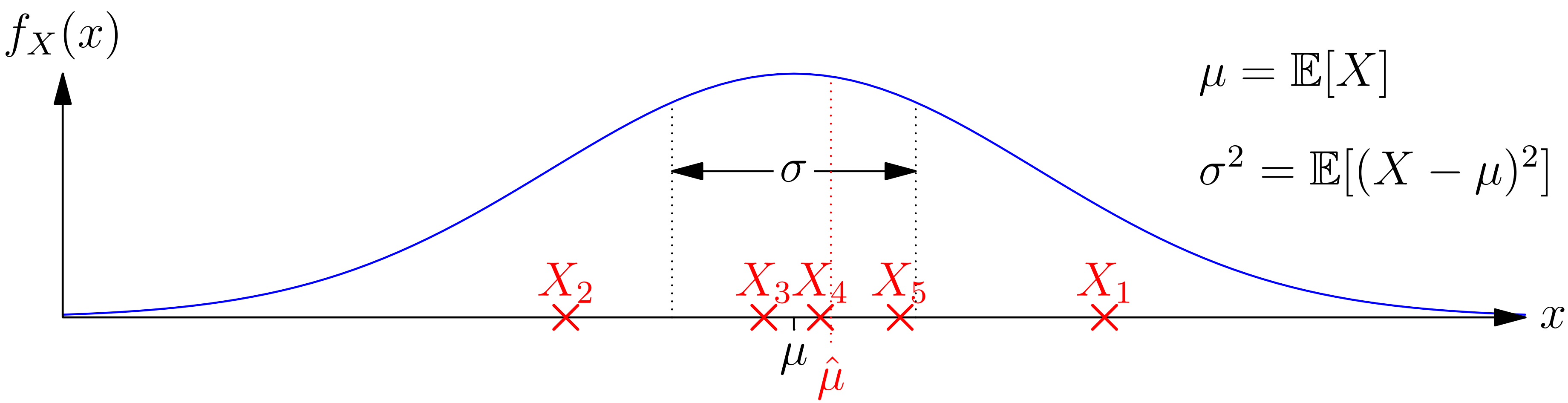
- By taking the mean over many samples we can reduce out variance and thus improve our generalisation performance

- To get a feel for this consider estimating the mean of a random variable X, from a number of samples ($n = 5$ in the example below)



Error In The Means

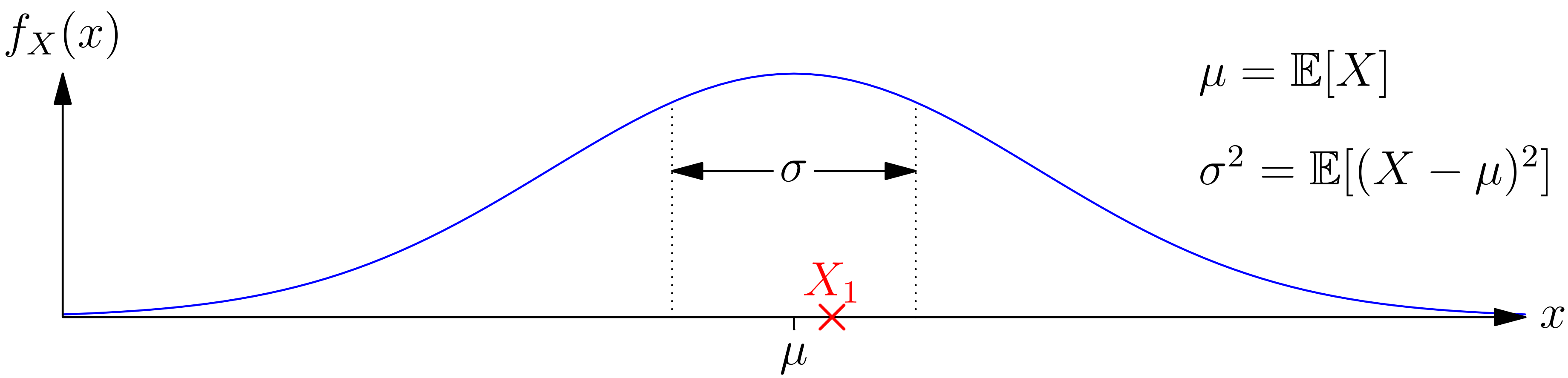
- By taking the mean over many samples we can reduce out variance and thus improve our generalisation performance
- To get a feel for this consider estimating the mean of a random variable X, from a number of samples ($n = 5$ in the example below)



Error In The Means

- By taking the mean over many samples we can reduce out variance and thus improve our generalisation performance

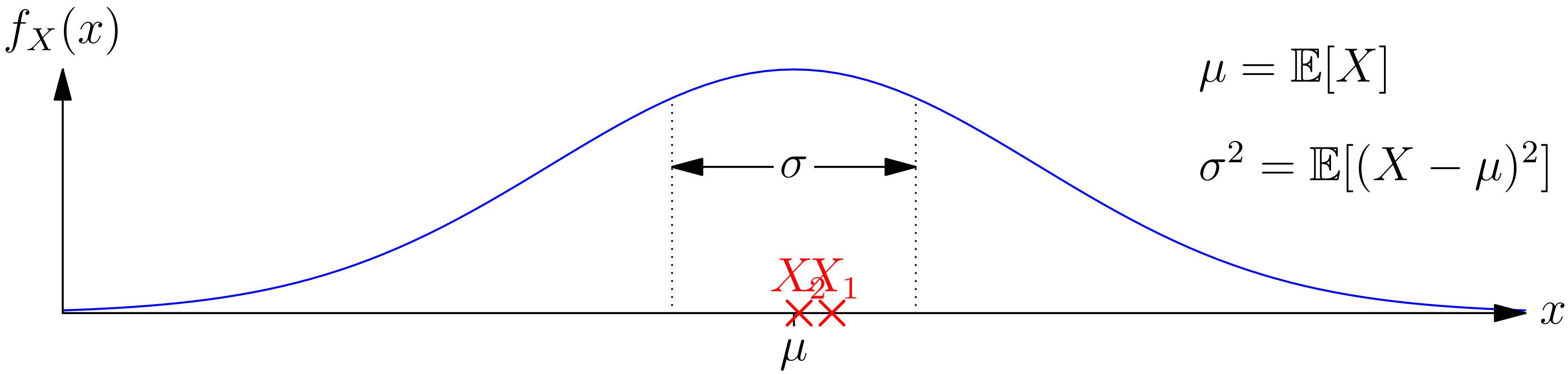
- To get a feel for this consider estimating the mean of a random variable X, from a number of samples ($n = 5$ in the example below)



Error In The Means

- By taking the mean over many samples we can reduce out variance and thus improve our generalisation performance

- To get a feel for this consider estimating the mean of a random variable X, from a number of samples ($n = 5$ in the example below)

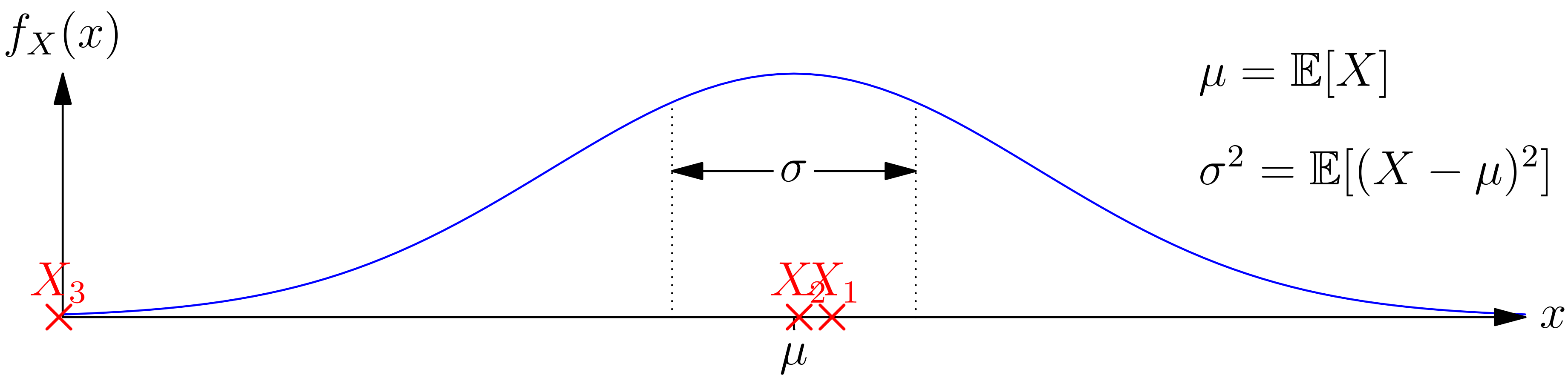


$\mu = \mathbb{E}[X]$

$\sigma^2 = \mathbb{E}[(X - \mu)^2]$

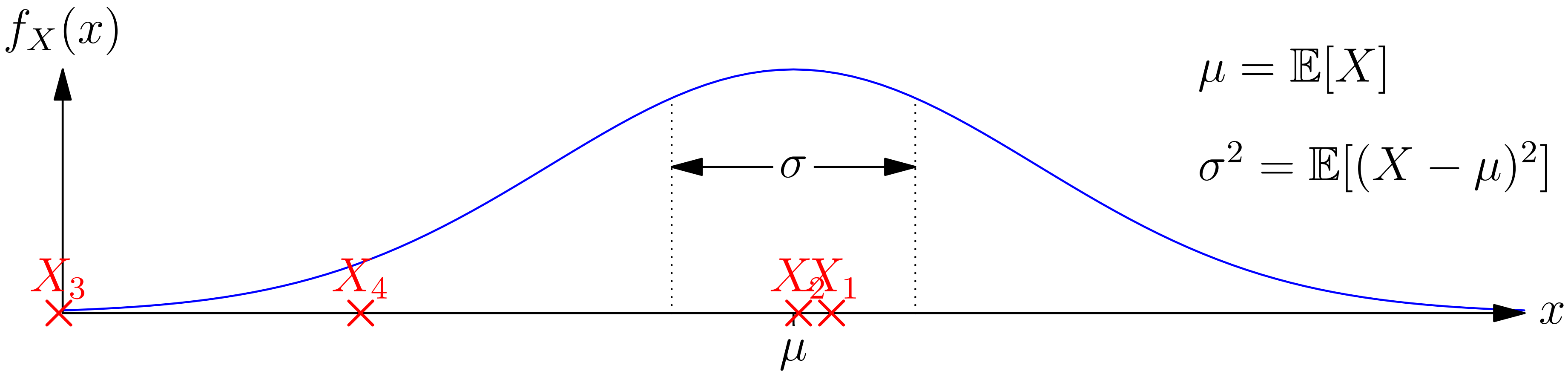
Error In The Means

- By taking the mean over many samples we can reduce out variance and thus improve our generalisation performance
- To get a feel for this consider estimating the mean of a random variable X, from a number of samples ($n = 5$ in the example below)



Error In The Means

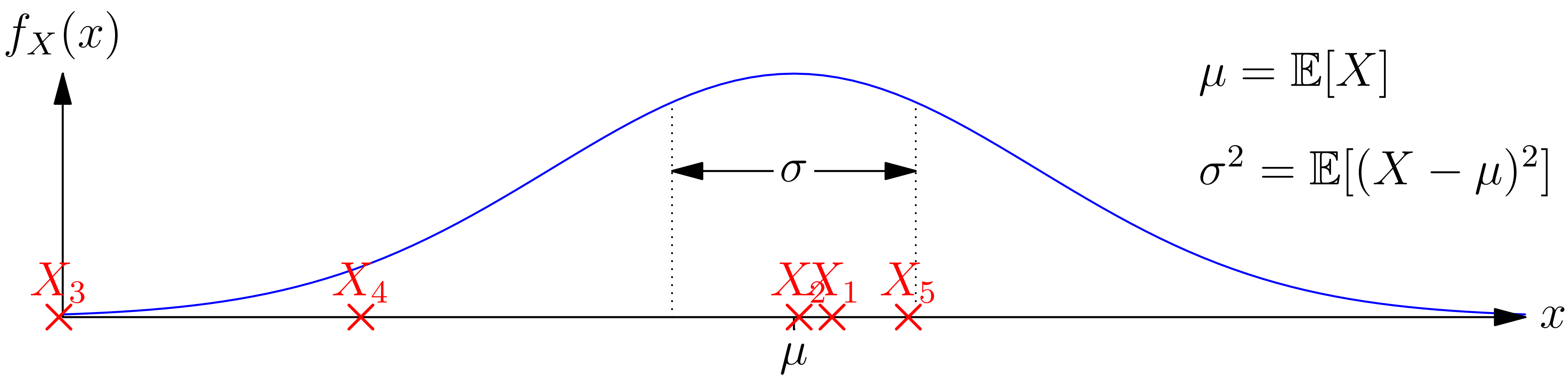
- By taking the mean over many samples we can reduce out variance and thus improve our generalisation performance
- To get a feel for this consider estimating the mean of a random variable X, from a number of samples ($n = 5$ in the example below)



Error In The Means

- By taking the mean over many samples we can reduce out variance and thus improve our generalisation performance

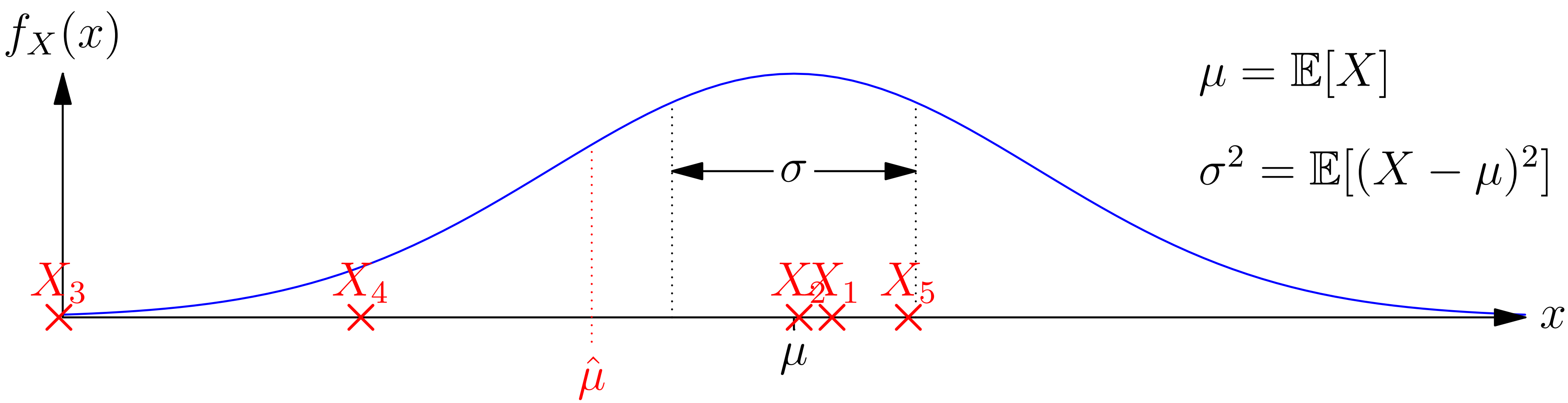
- To get a feel for this consider estimating the mean of a random variable X, from a number of samples ($n = 5$ in the example below)



Error In The Means

- By taking the mean over many samples we can reduce out variance and thus improve our generalisation performance

- To get a feel for this consider estimating the mean of a random variable X, from a number of samples ($n = 5$ in the example below)



Mean and Variance

- The expected value of the mean, $\hat{\mu}_n$ of n random **independent** variables, X_i, is the expected value $\mu = \mathbb{E}[X_i]$

$$\mathbb{E}[\hat{\mu}_n] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[X_i] = \frac{1}{n}\sum_{i=1}^{n}\mu = \mu$$

- The variance is $\mathbb{E}\left[(\hat{\mu}_n - \mu)^2\right]$ or equivalently

$$\frac{1}{n^2}\mathbb{E}\left[\left(\sum_{i=1}^{n}(X_i - \mu)\right)^2\right] = \frac{1}{n^2}\mathbb{E}\left[\sum_{i=1}^{n}(X_i - \mu)^2 + \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n}(X_i - \mu)\,(X_j - \mu)\right]$$

$$= \frac{1}{n^2}\sum_{i=1}^{n}\left(\mathbb{E}\left[(X_i - \mu)^2\right] + \sum_{\substack{j=1\\ j\neq i}}^{n}\mathbb{E}[X_i - \mu]\,\mathbb{E}[X_j - \mu]\right)$$

$$= \frac{1}{n^2}\sum_{i=1}^{n}\sigma^2 = \frac{1}{n}\sigma^2$$

Mean and Variance

- The expected value of the mean, $\hat{\mu}_n$ of n random **independent** variables, X_i, is the expected value $\mu = \mathbb{E}[X_i]$

$$\mathbb{E}[\hat{\mu}_n] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[X_i] = \frac{1}{n}\sum_{i=1}^{n}\mu = \mu$$

- The variance is $\mathbb{E}\left[(\hat{\mu}_n - \mu)^2\right]$ or equivalently

$$\frac{1}{n^2}\mathbb{E}\left[\left(\sum_{i=1}^{n}(X_i-\mu)\right)^2\right] = \frac{1}{n^2}\mathbb{E}\left[\sum_{i=1}^{n}(X_i-\mu)^2 + \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n}(X_i-\mu)\,(X_j-\mu)\right]$$

$$= \frac{1}{n^2}\sum_{i=1}^{n}\left(\mathbb{E}\left[(X_i-\mu)^2\right] + \sum_{\substack{j=1\\ j\neq i}}^{n}\mathbb{E}[X_i-\mu]\;\mathbb{E}[X_j-\mu]\right)$$

$$= \frac{1}{n^2}\sum_{i=1}^{n}\sigma^2 = \frac{1}{n}\sigma^2$$

Mean and Variance

- The expected value of the mean, $\hat{\mu}_n$ of n random **independent** variables, X_i, is the expected value $\mu = \mathbb{E}[X_i]$

$$\mathbb{E}[\hat{\mu}_n] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[X_i] = \frac{1}{n}\sum_{i=1}^{n}\mu = \mu$$

- The variance is $\mathbb{E}\left[(\hat{\mu}_n - \mu)^2\right]$ or equivalently

$$\frac{1}{n^2}\mathbb{E}\left[\left(\sum_{i=1}^{n}(X_i-\mu)\right)^2\right] = \frac{1}{n^2}\mathbb{E}\left[\sum_{i=1}^{n}(X_i-\mu)^2 + \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n}(X_i-\mu)\,(X_j-\mu)\right]$$

$$= \frac{1}{n^2}\sum_{i=1}^{n}\left(\mathbb{E}\left[(X_i-\mu)^2\right] + \sum_{\substack{j=1\\ j\neq i}}^{n}\mathbb{E}[X_i-\mu]\,\mathbb{E}[X_j-\mu]\right)$$

$$= \frac{1}{n^2}\sum_{i=1}^{n}\sigma^2 = \frac{1}{n}\sigma^2$$

Mean and Variance

- The expected value of the mean, $\hat{\mu}_n$ of n random **independent** variables, X_i, is the expected value $\mu = \mathbb{E}[X_i]$

$$\mathbb{E}[\hat{\mu}_n] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[X_i] = \frac{1}{n}\sum_{i=1}^{n}\mu = \mu$$

- The variance is $\mathbb{E}\left[(\hat{\mu}_n - \mu)^2\right]$ or equivalently

$$\frac{1}{n^2}\mathbb{E}\left[\left(\sum_{i=1}^{n}(X_i-\mu)\right)^2\right] = \frac{1}{n^2}\mathbb{E}\left[\sum_{i=1}^{n}(X_i-\mu)^2 + \sum_{i=1}^{n}\sum_{\substack{j=1\\j\neq i}}^{n}(X_i-\mu)\,(X_j-\mu)\right]$$

$$= \frac{1}{n^2}\sum_{i=1}^{n}\left(\mathbb{E}\left[(X_i-\mu)^2\right] + \sum_{\substack{j=1\\j\neq i}}^{n}\mathbb{E}[X_i-\mu]\;\mathbb{E}[X_j-\mu]\right)$$

$$= \frac{1}{n^2}\sum_{i=1}^{n}\sigma^2 = \frac{1}{n}\sigma^2$$

Mean and Variance

- The expected value of the mean, $\hat{\mu}_n$ of n random **independent** variables, X_i, is the expected value $\mu = \mathbb{E}[X_i]$

$$\mathbb{E}[\hat{\mu}_n] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[X_i] = \frac{1}{n}\sum_{i=1}^{n}\mu = \mu$$

- The variance is $\mathbb{E}\left[(\hat{\mu}_n - \mu)^2\right]$ or equivalently

$$\frac{1}{n^2}\mathbb{E}\left[\left(\sum_{i=1}^{n}(X_i-\mu)\right)^2\right] = \frac{1}{n^2}\mathbb{E}\left[\sum_{i=1}^{n}(X_i-\mu)^2 + \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n}(X_i-\mu)\,(X_j-\mu)\right]$$

$$= \frac{1}{n^2}\sum_{i=1}^{n}\left(\mathbb{E}\left[(X_i-\mu)^2\right] + \sum_{\substack{j=1\\ j\neq i}}^{n}\mathbb{E}[X_i-\mu]\;\mathbb{E}[X_j-\mu]\right)$$

$$= \frac{1}{n^2}\sum_{i=1}^{n}\sigma^2 = \frac{1}{n}\sigma^2$$

Mean and Variance

- The expected value of the mean, $\hat{\mu}_n$ of n random **independent** variables, X_i, is the expected value $\mu = \mathbb{E}[X_i]$

$$\mathbb{E}[\hat{\mu}_n] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[X_i] = \frac{1}{n}\sum_{i=1}^{n}\mu = \mu$$

- The variance is $\mathbb{E}\left[(\hat{\mu}_n - \mu)^2\right]$ or equivalently

$$\begin{aligned}
\frac{1}{n^2}\mathbb{E}\left[\left(\sum_{i=1}^{n}(X_i - \mu)\right)^2\right] &= \frac{1}{n^2}\mathbb{E}\left[\sum_{i=1}^{n}(X_i - \mu)^2 + \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n}(X_i - \mu)\,(X_j - \mu)\right] \\
&= \frac{1}{n^2}\sum_{i=1}^{n}\left(\mathbb{E}\left[(X_i - \mu)^2\right] + \sum_{\substack{j=1\\ j\neq i}}^{n}\mathbb{E}[X_i - \mu]\;\mathbb{E}[X_j - \mu]\right) \\
&= \frac{1}{n^2}\sum_{i=1}^{n}\sigma^2 = \frac{1}{n}\sigma^2
\end{aligned}$$

Bootstrap Aggregation (Bagging)

- To reduce the variance in a learning machine (such as a decision tree) we can average over many machines
- To average many machines they must learn something different
- We only have one data set, but we can resample from the data set to make them look a bit different—this is known a **bootstrapping**

Bootstrap Aggregation (Bagging)

- To reduce the variance in a learning machine (such as a decision tree) we can average over many machines
- To average many machines they must learn something different
- We only have one data set, but we can resample from the data set to make them look a bit different—this is known a **bootstrapping**

Bootstrap Aggregation (Bagging)

- To reduce the variance in a learning machine (such as a decision tree) we can average over many machines
- To average many machines they must learn something different
- We only have one data set, but we can resample from the data set to make them look a bit different—this is known a **bootstrapping**

Bootstrap Aggregation (Bagging)

- To reduce the variance in a learning machine (such as a decision tree) we can average over many machines
- To average many machines they must learn something different
- We only have one data set, but we can resample from the data set to make them look a bit different—this is known a **bootstrapping**

Bootstrap Aggregation (Bagging)

- To reduce the variance in a learning machine (such as a decision tree) we can average over many machines
- To average many machines they must learn something different
- We only have one data set, but we can resample from the data set to make them look a bit different—this is known a **bootstrapping**

$(\boldsymbol{x}_1, y_1)$ $(\boldsymbol{x}_2, y_2)$ $(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_4, y_4)$ $(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_6, y_6)$

Bootstrap Aggregation (Bagging)

- To reduce the variance in a learning machine (such as a decision tree) we can average over many machines
- To average many machines they must learn something different
- We only have one data set, but we can resample from the data set to make them look a bit different—this is known a **bootstrapping**

$(\boldsymbol{x}_5,\, y_5)$

$(\boldsymbol{x}_1,\, y_1)$ $(\boldsymbol{x}_2,\, y_2)$ $(\boldsymbol{x}_3,\, y_3)$ $(\boldsymbol{x}_4,\, y_4)$ $(\boldsymbol{x}_5,\, y_5)$ $(\boldsymbol{x}_6,\, y_6)$

Bootstrap Aggregation (Bagging)

- To reduce the variance in a learning machine (such as a decision tree) we can average over many machines
- To average many machines they must learn something different
- We only have one data set, but we can resample from the data set to make them look a bit different—this is known a **bootstrapping**

$(\boldsymbol{x}_5,\, y_5)\;\; (\boldsymbol{x}_4,\, y_4)$

$(\boldsymbol{x}_1,\, y_1)\;\; (\boldsymbol{x}_2,\, y_2)\;\; (\boldsymbol{x}_3,\, y_3)\;\; (\boldsymbol{x}_4,\, y_4)\;\; (\boldsymbol{x}_5,\, y_5)\;\; (\boldsymbol{x}_6,\, y_6)$

Bootstrap Aggregation (Bagging)

- To reduce the variance in a learning machine (such as a decision tree) we can average over many machines

- To average many machines they must learn something different

- We only have one data set, but we can resample from the data set to make them look a bit different—this is known a **bootstrapping**

$(\boldsymbol{x}_5,\, y_5)\ \ (\boldsymbol{x}_4,\, y_4)\ \ (\boldsymbol{x}_5,\, y_5)$

$(\boldsymbol{x}_1,\, y_1)\ \ (\boldsymbol{x}_2,\, y_2)\ \ (\boldsymbol{x}_3,\, y_3)\ \ (\boldsymbol{x}_4,\, y_4)\ \ (\boldsymbol{x}_5,\, y_5)\ \ (\boldsymbol{x}_6,\, y_6)$

Bootstrap Aggregation (Bagging)

- To reduce the variance in a learning machine (such as a decision tree) we can average over many machines
- To average many machines they must learn something different
- We only have one data set, but we can resample from the data set to make them look a bit different—this is known a **bootstrapping**

$(\boldsymbol{x}_5,\, y_5)\ \ (\boldsymbol{x}_4,\, y_4)\ \ (\boldsymbol{x}_5,\, y_5)\ \ (\boldsymbol{x}_6,\, y_6)$

$(\boldsymbol{x}_1,\, y_1)\ \ (\boldsymbol{x}_2,\, y_2)\ \ (\boldsymbol{x}_3,\, y_3)\ \ (\boldsymbol{x}_4,\, y_4)\ \ (\boldsymbol{x}_5,\, y_5)\ \ (\boldsymbol{x}_6,\, y_6)$

Bootstrap Aggregation (Bagging)

- To reduce the variance in a learning machine (such as a decision tree) we can average over many machines
- To average many machines they must learn something different
- We only have one data set, but we can resample from the data set to make them look a bit different—this is known a **bootstrapping**

$(\boldsymbol{x}_5,\, y_5)\ \ (\boldsymbol{x}_4,\, y_4)\ \ (\boldsymbol{x}_5,\, y_5)\ \ (\boldsymbol{x}_6,\, y_6)\ \ (\boldsymbol{x}_2,\, y_2)$

$(\boldsymbol{x}_1,\, y_1)\ \ (\boldsymbol{x}_2,\, y_2)\ \ (\boldsymbol{x}_3,\, y_3)\ \ (\boldsymbol{x}_4,\, y_4)\ \ (\boldsymbol{x}_5,\, y_5)\ \ (\boldsymbol{x}_6,\, y_6)$

Bootstrap Aggregation (Bagging)

- To reduce the variance in a learning machine (such as a decision tree) we can average over many machines
- To average many machines they must learn something different
- We only have one data set, but we can resample from the data set to make them look a bit different—this is known a **bootstrapping**

$(\boldsymbol{x}_5,\, y_5)\ (\boldsymbol{x}_4,\, y_4)\ (\boldsymbol{x}_5,\, y_5)\ (\boldsymbol{x}_6,\, y_6)\ (\boldsymbol{x}_2,\, y_2)\ (\boldsymbol{x}_1,\, y_1)$

$(\boldsymbol{x}_1,\, y_1)\ (\boldsymbol{x}_2,\, y_2)\ (\boldsymbol{x}_3,\, y_3)\ (\boldsymbol{x}_4,\, y_4)\ (\boldsymbol{x}_5,\, y_5)\ (\boldsymbol{x}_6,\, y_6)$

Bootstrap Aggregation (Bagging)

- To reduce the variance in a learning machine (such as a decision tree) we can average over many machines

- To average many machines they must learn something different

- We only have one data set, but we can resample from the data set to make them look a bit different—this is known a **bootstrapping**

$(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_4, y_4)$ $(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_6, y_6)$ $(\boldsymbol{x}_2, y_2)$ $(\boldsymbol{x}_1, y_1)$

$(\boldsymbol{x}_1, y_1)$ $(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_2, y_2)$ $(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_2, y_2)$

$(\boldsymbol{x}_1, y_1)$ $(\boldsymbol{x}_2, y_2)$ $(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_4, y_4)$ $(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_6, y_6)$

Bootstrap Aggregation (Bagging)

- To reduce the variance in a learning machine (such as a decision tree) we can average over many machines
- To average many machines they must learn something different
- We only have one data set, but we can resample from the data set to make them look a bit different—this is known a **bootstrapping**

$(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_4, y_4)$ $(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_6, y_6)$ $(\boldsymbol{x}_2, y_2)$ $(\boldsymbol{x}_1, y_1)$

$(\boldsymbol{x}_1, y_1)$ $(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_2, y_2)$ $(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_2, y_2)$

$(\boldsymbol{x}_1, y_1)$ $(\boldsymbol{x}_2, y_2)$ $(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_4, y_4)$ $(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_6, y_6)$ $(\boldsymbol{x}_6, y_6)$ $(\boldsymbol{x}_1, y_1)$ $(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_6, y_6)$ $(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_4, y_4)$

Bootstrap Aggregation (Bagging)

- To reduce the variance in a learning machine (such as a decision tree) we can average over many machines
- To average many machines they must learn something different
- We only have one data set, but we can resample from the data set to make them look a bit different—this is known a **bootstrapping**

$(\boldsymbol{x}_1, y_1)$ $(\boldsymbol{x}_2, y_2)$ $(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_4, y_4)$ $(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_6, y_6)$

$(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_4, y_4)$ $(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_6, y_6)$ $(\boldsymbol{x}_2, y_2)$ $(\boldsymbol{x}_1, y_1)$

$(\boldsymbol{x}_1, y_1)$ $(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_2, y_2)$ $(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_2, y_2)$

$(\boldsymbol{x}_6, y_6)$ $(\boldsymbol{x}_1, y_1)$ $(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_6, y_6)$ $(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_4, y_4)$

$(\boldsymbol{x}_4, y_4)$ $(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_1, y_1)$ $(\boldsymbol{x}_6, y_6)$ $(\boldsymbol{x}_4, y_4)$ $(\boldsymbol{x}_6, y_6)$

Bootstrap Aggregation (Bagging)

- To reduce the variance in a learning machine (such as a decision tree) we can average over many machines
- To average many machines they must learn something different
- We only have one data set, but we can resample from the data set to make them look a bit different—this is known a **bootstrapping**

$(\boldsymbol{x}_1, y_1)$ $(\boldsymbol{x}_2, y_2)$ $(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_4, y_4)$ $(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_6, y_6)$

$(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_4, y_4)$ $(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_6, y_6)$ $(\boldsymbol{x}_2, y_2)$ $(\boldsymbol{x}_1, y_1)$

$(\boldsymbol{x}_1, y_1)$ $(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_2, y_2)$ $(\boldsymbol{x}_5, y_5)$ $(\boldsymbol{x}_2, y_2)$

$(\boldsymbol{x}_6, y_6)$ $(\boldsymbol{x}_1, y_1)$ $(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_6, y_6)$ $(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_4, y_4)$

$(\boldsymbol{x}_4, y_4)$ $(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_1, y_1)$ $(\boldsymbol{x}_6, y_6)$ $(\boldsymbol{x}_4, y_4)$ $(\boldsymbol{x}_6, y_6)$

$(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_2, y_2)$ $(\boldsymbol{x}_3, y_3)$ $(\boldsymbol{x}_4, y_4)$ $(\boldsymbol{x}_6, y_6)$ $(\boldsymbol{x}_2, y_2)$

Performance of Bagging

- For classification we get our different machines to vote
- For regression we can average the prediction of different machines
- Bagging improves the performance of decision trees
- However, we can usually do better using Boosting
- This is because our decision trees are correlated

Performance of Bagging

- For classification we get our different machines to vote
- For regression we can average the prediction of different machines
- Bagging improves the performance of decision trees
- However, we can usually do better using Boosting
- This is because our decision trees are correlated

Performance of Bagging

- For classification we get our different machines to vote
- For regression we can average the prediction of different machines
- Bagging improves the performance of decision trees
- However, we can usually do better using Boosting
- This is because our decision trees are correlated

Performance of Bagging

- For classification we get our different machines to vote
- For regression we can average the prediction of different machines
- Bagging improves the performance of decision trees
- However, we can usually do better using Boosting
- This is because our decision trees are correlated

Performance of Bagging

- For classification we get our different machines to vote
- For regression we can average the prediction of different machines
- Bagging improves the performance of decision trees
- However, we can usually do better using Boosting
- This is because our decision trees are correlated

Variance of Positive Correlated Variables

- If we calculate the variance of the mean of positively correlated variables with correlation ρ we find

$$\frac{1}{n^2}\,\mathbb{E}\left[\left(\sum_{i=1}^{n} X_i - \mu\right)^2\right] = \rho\,\sigma^2 + \frac{1-\rho}{n}\sigma^2$$

$(\rho = \mathbb{E}[(X_i - \mu)\,(X_j - \mu)]\,/\sigma^2)$

- As $n \to \infty$ the second term vanishes, but we are left with the first term

- If we want to do well we need our learning machines to be unbiased and decorrelated

Variance of Positive Correlated Variables

- If we calculate the variance of the mean of positively correlated variables with correlation ρ we find

$$\frac{1}{n^2}\mathbb{E}\left[\left(\sum_{i=1}^{n} X_i - \mu\right)^2\right] = \rho\,\sigma^2 + \frac{1-\rho}{n}\sigma^2$$

$(\rho = \mathbb{E}[(X_i - \mu)\,(X_j - \mu)]\,/\sigma^2)$

- As $n \to \infty$ the second term vanishes, but we are left with the first term

- If we want to do well we need our learning machines to be unbiased and decorrelated

Variance of Positive Correlated Variables

- If we calculate the variance of the mean of positively correlated variables with correlation ρ we find

$$\frac{1}{n^2}\mathbb{E}\left[\left(\sum_{i=1}^{n} X_i - \mu\right)^2\right] = \rho\,\sigma^2 + \frac{1-\rho}{n}\sigma^2$$

$(\rho = \mathbb{E}[(X_i - \mu)\,(X_j - \mu)]\,/\sigma^2)$

- As $n \to \infty$ the second term vanishes, but we are left with the first term

- If we want to do well we need our learning machines to be unbiased and decorrelated

Random Forest

- In random forests we average much less correlated trees
- We do this for each tree we choose a subset of $p' \ll p$ of the features on which to split the tree
- Typically p' can range from 1 to $\sqrt{p}$
- The trees aren't that good, but are very decorrelated
- By averaging over a huge number of trees (order of 1000) we typically get good results
- Random Forest won (wins?) many competitions

Random Forest

- In random forests we average much less correlated trees
- We do this for each tree we choose a subset of $p' \ll p$ of the features on which to split the tree
- Typically p' can range from 1 to $\sqrt{p}$
- The trees aren't that good, but are very decorrelated
- By averaging over a huge number of trees (order of 1000) we typically get good results
- Random Forest won (wins?) many competitions

Random Forest

- In random forests we average much less correlated trees
- We do this for each tree we choose a subset of $p' \ll p$ of the features on which to split the tree
- Typically p' can range from 1 to $\sqrt{p}$
- The trees aren't that good, but are very decorrelated
- By averaging over a huge number of trees (order of 1000) we typically get good results
- Random Forest won (wins?) many competitions

Random Forest

- In random forests we average much less correlated trees
- We do this for each tree we choose a subset of $p' \ll p$ of the features on which to split the tree
- Typically p' can range from 1 to $\sqrt{p}$
- The trees aren't that good, but are very decorrelated
- By averaging over a huge number of trees (order of 1000) we typically get good results
- Random Forest won (wins?) many competitions

Random Forest

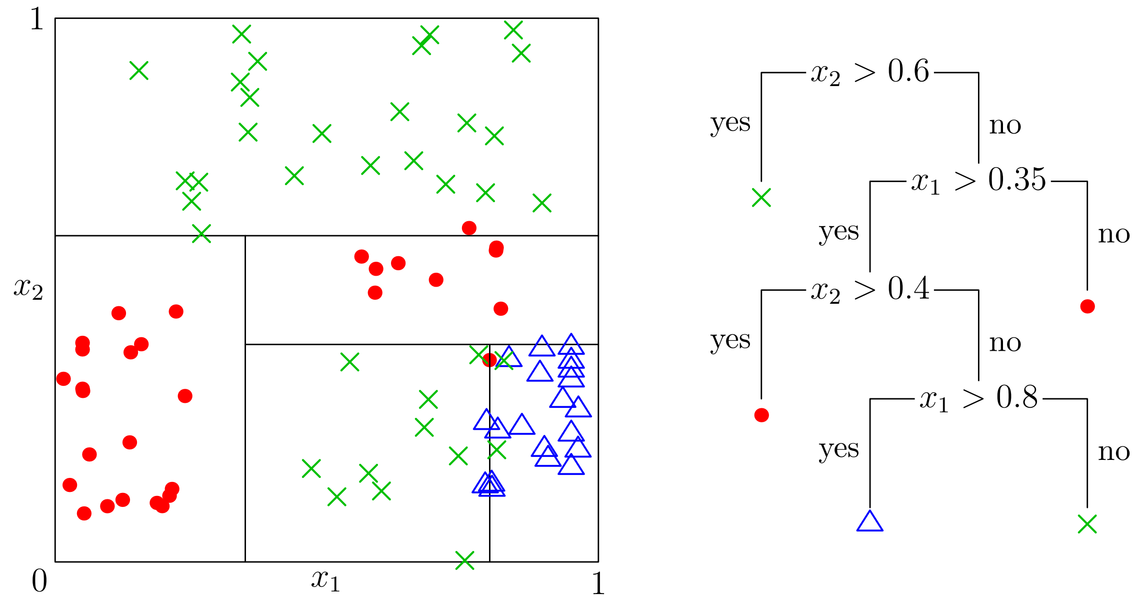
- In random forests we average much less correlated trees
- We do this for each tree we choose a subset of $p' \ll p$ of the features on which to split the tree
- Typically p' can range from 1 to $\sqrt{p}$
- The trees aren't that good, but are very decorrelated
- By averaging over a huge number of trees (order of 1000) we typically get good results
- Random Forest won (wins?) many competitions

Random Forest

- In random forests we average much less correlated trees
- We do this for each tree we choose a subset of $p' \ll p$ of the features on which to split the tree
- Typically p' can range from 1 to $\sqrt{p}$
- The trees aren't that good, but are very decorrelated
- By averaging over a huge number of trees (order of 1000) we typically get good results
- Random Forest won (wins?) many competitions

Outline

1. Decision Trees
2. Bagging
3. **Boosting**



Boosting

- In boosting we make a **strong learner** by using a weighted sum of **weak learners**

$$C_n(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \, \hat{h}_i(\boldsymbol{x})$$

- Weak learners, $\hat{h}_i(\boldsymbol{x})$, are learning machine that do a little better than chance

- The trick is to choose the weights, α_i

- Because the weak learners do little better than chance we (miraculously) **don't** overfit

Boosting

- In boosting we make a **strong learner** by using a weighted sum of **weak learners**

$$C_n(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \, \hat{h}_i(\boldsymbol{x})$$

- Weak learners, $\hat{h}_i(\boldsymbol{x})$, are learning machine that do a little better than chance
- The trick is to choose the weights, α_i
- Because the weak learners do little better than chance we (miraculously) **don't** overfit

Boosting

- In boosting we make a **strong learner** by using a weighted sum of **weak learners**

$$C_n(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \, \hat{h}_i(\boldsymbol{x})$$

- Weak learners, $\hat{h}_i(\boldsymbol{x})$, are learning machine that do a little better than chance
- The trick is to choose the weights, α_i
- Because the weak learners do little better than chance we (miraculously) **don't** overfit

Boosting

- In boosting we make a **strong learner** by using a weighted sum of **weak learners**

$$C_n(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \, \hat{h}_i(\boldsymbol{x})$$

- Weak learners, $\hat{h}_i(\boldsymbol{x})$, are learning machine that do a little better than chance
- The trick is to choose the weights, α_i
- Because the weak learners do little better than chance we (miraculously) **don't** overfit

Boosting

- In boosting we make a **strong learner** by using a weighted sum of **weak learners**

$$C_n(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \, \hat{h}_i(\boldsymbol{x})$$

- Weak learners, $\hat{h}_i(\boldsymbol{x})$, are learning machine that do a little better than chance

- The trick is to choose the weights, α_i

- Because the weak learners do little better than chance we (miraculously) **don't** overfit that much

Shallow Trees

- One of the most effective type of weak learner are very shallow trees
- Sometimes we just use one variable (the stump)
- There are different algorithms for choosing the weights
  - ⋆ adaboost
  - ⋆ gradient boosting

Shallow Trees

- One of the most effective type of weak learner are very shallow trees
- Sometimes we just use one variable (the stump)
- There are different algorithms for choosing the weights
  - ⋆ adaboost
  - ⋆ gradient boosting

Shallow Trees

- One of the most effective type of weak learner are very shallow trees
- Sometimes we just use one variable (the stump)
- There are different algorithms for choosing the weights
  - ⋆ adaboost
  - ⋆ gradient boosting

Shallow Trees

- One of the most effective type of weak learner are very shallow trees
- Sometimes we just use one variable (the stump)
- There are different algorithms for choosing the weights
  - ⋆ adaboost—a classic algorithm for binary problems
  - ⋆ gradient boosting

Shallow Trees

- One of the most effective type of weak learner are very shallow trees
- Sometimes we just use one variable (the stump)
- There are different algorithms for choosing the weights
  - ⋆ adaboost—a classic algorithm for binary problems
  - ⋆ gradient boosting

Shallow Trees

- One of the most effective type of weak learner are very shallow trees
- Sometimes we just use one variable (the stump)
- There are different algorithms for choosing the weights
  - ⋆ adaboost—a classic algorithm for binary problems
  - ⋆ gradient boosting—used for regression, trains a classifier on the residual errors

Boosting a Binary Classifier

- Suppose we have a binary classification task with data $\mathcal{D} = \{(\boldsymbol{x}^\mu, y^\mu) | \mu = 1, 2, \ldots, m\}$ with $y^\mu \in \{-1, 1\}$
- Our i^{th} weak learner provides a prediction $\hat{h}_i(\boldsymbol{x}^\mu) \in \{-1, 1\}$
- We ask, can we find a linear combination

$$C_n(\boldsymbol{x}) = \alpha_1 \hat{h}_1(\boldsymbol{x}) + \alpha_2 \hat{h}_2(\boldsymbol{x}) + \cdots + \alpha_n \hat{h}_n(\boldsymbol{x})$$

- So that $\operatorname{sgn}\big(C_n(\boldsymbol{x})\big)$ is a strong learner?

Boosting a Binary Classifier

- Suppose we have a binary classification task with data $\mathcal{D} = \{(\boldsymbol{x}^\mu, y^\mu) | \mu = 1, 2, \ldots, m\}$ with $y^\mu \in \{-1, 1\}$

- Our i^{th} weak learner provides a prediction $\hat{h}_i(\boldsymbol{x}^\mu) \in \{-1, 1\}$

- We ask, can we find a linear combination

$$C_n(\boldsymbol{x}) = \alpha_1 \hat{h}_1(\boldsymbol{x}) + \alpha_2 \hat{h}_2(\boldsymbol{x}) + \cdots + \alpha_n \hat{h}_n(\boldsymbol{x})$$

- So that $\operatorname{sgn}\big(C_n(\boldsymbol{x})\big)$ is a strong learner?

Boosting a Binary Classifier

- Suppose we have a binary classification task with data $\mathcal{D} = \{(\boldsymbol{x}^\mu, y^\mu) | \mu = 1, 2, \ldots, m\}$ with $y^\mu \in \{-1, 1\}$
- Our i^{th} weak learner provides a prediction $\hat{h}_i(\boldsymbol{x}^\mu) \in \{-1, 1\}$
- We ask, can we find a linear combination

$$C_n(\boldsymbol{x}) = \alpha_1 \hat{h}_1(\boldsymbol{x}) + \alpha_2 \hat{h}_2(\boldsymbol{x}) + \cdots + \alpha_n \hat{h}_n(\boldsymbol{x})$$

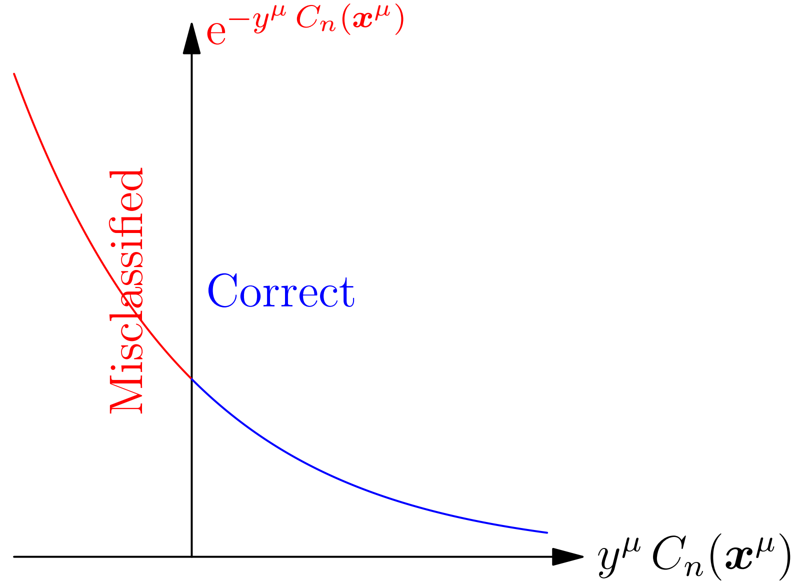
- So that $\operatorname{sgn}\big(C_n(\boldsymbol{x})\big)$ is a strong learner?

AdaBoost

- AdaBoost is a classic solution to this problem

- It assigns an "loss function"

$$L_n = \sum_{\mu=1}^{m} e^{-y^\mu C_n(\boldsymbol{x}^\mu)}$$

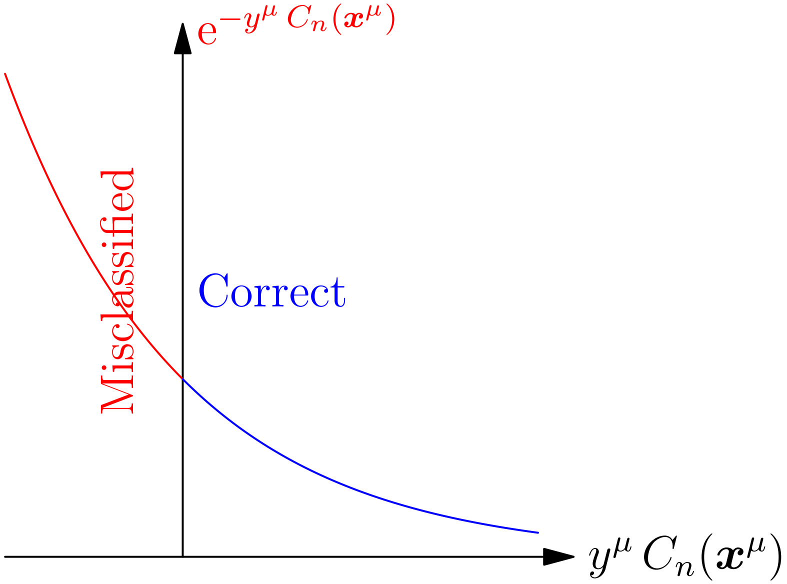


- This punishes examples where there is an errors more than correct classifications

AdaBoost

- AdaBoost is a classic solution to this problem
- It assigns an "loss function"

$$L_n = \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu C_n(\boldsymbol{x}^\mu)}$$

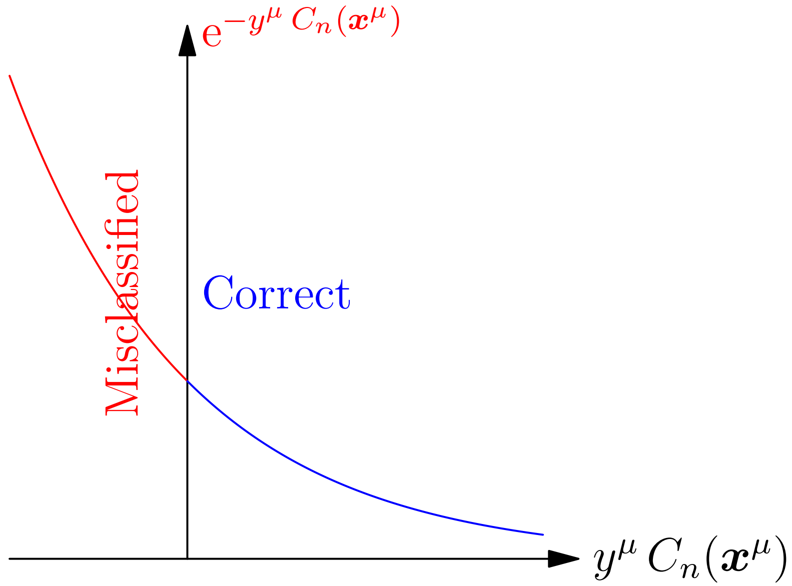


- This punishes examples where there is an errors more than correct classifications

AdaBoost

- AdaBoost is a classic solution to this problem
- It assigns an "loss function"

$$L_n = \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu C_n(\boldsymbol{x}^\mu)}$$



- This punishes examples where there is an errors more than correct classifications

Iterative Learning

- We build up a strong learner iteratively (greedily)

$$C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \alpha_n \, \hat{h}_n(\boldsymbol{x})$$

- Defining $w_1^\mu = 1$ and $w_n^\mu = \mathrm{e}^{-y^\mu C_{n-1}(\boldsymbol{x}^\mu)}$ then

$$\begin{aligned}
L_n(\alpha_n) &= \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu C_n(\boldsymbol{x}^\mu)} = \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu (C_{n-1}(\boldsymbol{x}^\mu) + \alpha_n \, \hat{h}_n(\boldsymbol{x}^\mu))} \\
&= \sum_{\mu=1}^{m} w_n^\mu \, \mathrm{e}^{-\alpha_n \, y^\mu \, \hat{h}_n(\boldsymbol{x}^\mu)} = \sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu \, \mathrm{e}^{\alpha_n} + \sum_{\mu: y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu \, \mathrm{e}^{-\alpha_n} \\
&= \mathrm{e}^{-\alpha_n} \sum_{\mu=1}^{m} w_n^\mu + (\mathrm{e}^{\alpha_n} - \mathrm{e}^{-\alpha_n}) \sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu
\end{aligned}$$

Iterative Learning

- We build up a strong learner iteratively (greedily)

$$C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \alpha_n \, \hat{h}_n(\boldsymbol{x})$$

- Defining $w_1^\mu = 1$ and $w_n^\mu = \mathrm{e}^{-y^\mu C_{n-1}(\boldsymbol{x}^\mu)}$ then

$$L_n(\alpha_n) = \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu C_n(\boldsymbol{x}^\mu)} = \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu (C_{n-1}(\boldsymbol{x}^\mu) + \alpha_n \, \hat{h}_n(\boldsymbol{x}^\mu))}$$

$$= \sum_{\mu=1}^{m} w_n^\mu \, \mathrm{e}^{-\alpha_n \, y^\mu \, \hat{h}_n(\boldsymbol{x}^\mu)} = \sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu \, \mathrm{e}^{\alpha_n} + \sum_{\mu: y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu \, \mathrm{e}^{-\alpha_n}$$

$$= \mathrm{e}^{-\alpha_n} \sum_{\mu=1}^{m} w_n^\mu + (\mathrm{e}^{\alpha_n} - \mathrm{e}^{-\alpha_n}) \sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$$

Iterative Learning

- We build up a strong learner iteratively (greedily)

$$C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \alpha_n \, \hat{h}_n(\boldsymbol{x})$$

- Defining $w_1^\mu = 1$ and $w_n^\mu = \mathrm{e}^{-y^\mu C_{n-1}(\boldsymbol{x}^\mu)}$ then

$$\begin{aligned}
L_n(\alpha_n) &= \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu C_n(\boldsymbol{x}^\mu)} = \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu (C_{n-1}(\boldsymbol{x}^\mu) + \alpha_n \, \hat{h}_n(\boldsymbol{x}^\mu))} \\
&= \sum_{\mu=1}^{m} w_n^\mu \, \mathrm{e}^{-\alpha_n \, y^\mu \, \hat{h}_n(\boldsymbol{x}^\mu)} = \sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu \, \mathrm{e}^{\alpha_n} + \sum_{\mu: y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu \, \mathrm{e}^{-\alpha_n} \\
&= \mathrm{e}^{-\alpha_n} \sum_{\mu=1}^{m} w_n^\mu + \left(\mathrm{e}^{\alpha_n} - \mathrm{e}^{-\alpha_n}\right) \sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu
\end{aligned}$$

Iterative Learning

- We build up a strong learner iteratively (greedily)

$$C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \alpha_n \, \hat{h}_n(\boldsymbol{x})$$

- Defining $w_1^\mu = 1$ and $w_n^\mu = \mathrm{e}^{-y^\mu C_{n-1}(\boldsymbol{x}^\mu)}$ then

$$L_n(\alpha_n) = \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu C_n(\boldsymbol{x}^\mu)} = \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu (C_{n-1}(\boldsymbol{x}^\mu) + \alpha_n \, \hat{h}_n(\boldsymbol{x}^\mu))}$$

$$= \sum_{\mu=1}^{m} w_n^\mu \, \mathrm{e}^{-\alpha_n \, y^\mu \, \hat{h}_n(\boldsymbol{x}^\mu)} = \sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu \, \mathrm{e}^{\alpha_n} + \sum_{\mu : y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu \, \mathrm{e}^{-\alpha_n}$$

$$= \mathrm{e}^{-\alpha_n} \sum_{\mu=1}^{m} w_n^\mu + \left(\mathrm{e}^{\alpha_n} - \mathrm{e}^{-\alpha_n}\right) \sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$$

Iterative Learning

- We build up a strong learner iteratively (greedily)

$$C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \alpha_n \, \hat{h}_n(\boldsymbol{x})$$

- Defining $w_1^\mu = 1$ and $w_n^\mu = \mathrm{e}^{-y^\mu C_{n-1}(\boldsymbol{x}^\mu)}$ then

$$\begin{aligned}
L_n(\alpha_n) &= \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu C_n(\boldsymbol{x}^\mu)} = \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu (C_{n-1}(\boldsymbol{x}^\mu) + \alpha_n \, \hat{h}_n(\boldsymbol{x}^\mu))} \\
&= \sum_{\mu=1}^{m} w_n^\mu \, \mathrm{e}^{-\alpha_n \, y^\mu \, \hat{h}_n(\boldsymbol{x}^\mu)} = \sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu \, \mathrm{e}^{\alpha_n} + \sum_{\mu : y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu \, \mathrm{e}^{-\alpha_n} \\
&= \mathrm{e}^{-\alpha_n} \sum_{\mu=1}^{m} w_n^\mu + (\mathrm{e}^{\alpha_n} - \mathrm{e}^{-\alpha_n}) \sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu
\end{aligned}$$

Iterative Learning

- We build up a strong learner iteratively (greedily)

$$C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \alpha_n \, \hat{h}_n(\boldsymbol{x})$$

- Defining $w_1^\mu = 1$ and $w_n^\mu = \mathrm{e}^{-y^\mu C_{n-1}(\boldsymbol{x}^\mu)}$ then

$$L_n(\alpha_n) = \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu C_n(\boldsymbol{x}^\mu)} = \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu (C_{n-1}(\boldsymbol{x}^\mu) + \alpha_n \, \hat{h}_n(\boldsymbol{x}^\mu))}$$

$$= \sum_{\mu=1}^{m} w_n^\mu \, \mathrm{e}^{-\alpha_n \, y^\mu \, \hat{h}_n(\boldsymbol{x}^\mu)} = \sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu \, \mathrm{e}^{\alpha_n} + \sum_{\mu : y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu \, \mathrm{e}^{-\alpha_n}$$

$$= \mathrm{e}^{-\alpha_n} \sum_{\mu=1}^{m} w_n^\mu + (\mathrm{e}^{\alpha_n} - \mathrm{e}^{-\alpha_n}) \sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$$

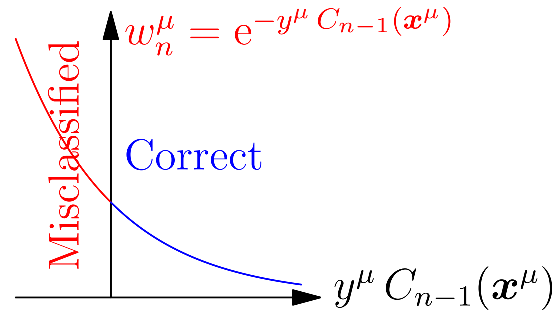
Choosing a Weak Classifier

- To minimise the error

$$L_n(\alpha_n) = \mathrm{e}^{-\alpha_n} \sum_{\mu=1}^{m} w_n^\mu + \left(\mathrm{e}^{\alpha_n} - \mathrm{e}^{-\alpha_n}\right) \sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$$

- We choose the weak learner with the lowest value of

$$\sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu = \sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} \mathrm{e}^{-y^\mu C_{n-1}(\boldsymbol{x}^\mu)}$$



- That is, it misclassifies only where the other learners classify well

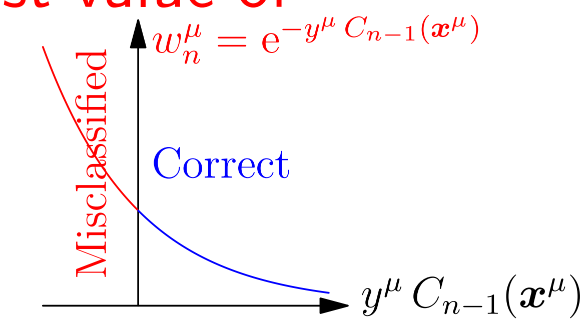
Choosing a Weak Classifier

- To minimise the error

$$L_n(\alpha_n) = \mathrm{e}^{-\alpha_n} \sum_{\mu=1}^{m} w_n^\mu + \left(\mathrm{e}^{\alpha_n} - \mathrm{e}^{-\alpha_n}\right) \sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$$

- We choose the weak learner with the lowest value of

$$\sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu = \sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} \mathrm{e}^{-y^\mu C_{n-1}(\boldsymbol{x}^\mu)}$$



- That is, it misclassifies only where the other learners classify well

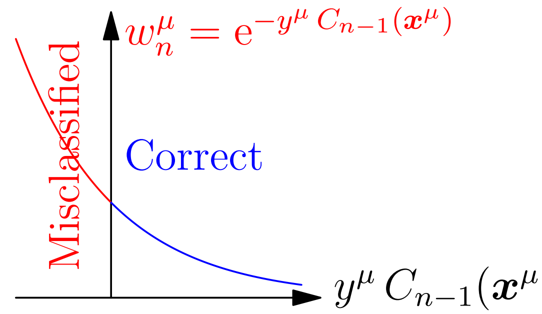
Choosing a Weak Classifier

- To minimise the error

$$L_n(\alpha_n) = \mathrm{e}^{-\alpha_n} \sum_{\mu=1}^{m} w_n^\mu + \left(\mathrm{e}^{\alpha_n} - \mathrm{e}^{-\alpha_n}\right) \sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$$

- We choose the weak learner with the lowest value of

$$\sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu = \sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} \mathrm{e}^{-y^\mu C_{n-1}(\boldsymbol{x}^\mu)}$$



- That is, it misclassifies only where the other learners classify well

Choosing Weights

- We now choose the weight α_n to minimise the error $L_n(\alpha_n)$

$$\frac{\partial\, L_n(\alpha_n)}{\partial\, \alpha_n} = \mathrm{e}^{\alpha_n} \sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu \; - \mathrm{e}^{-\alpha_n} \sum_{\mu : y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu = 0$$

- That is

$$\mathrm{e}^{2\,\alpha_n} = \frac{\displaystyle\sum_{\mu : y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\displaystyle\sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu} \qquad \text{or} \qquad \alpha_n = \frac{1}{2} \log\left(\frac{\displaystyle\sum_{\mu : y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\displaystyle\sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu} \right)$$

Choosing Weights

- We now choose the weight α_n to minimise the error $L_n(\alpha_n)$

$$\frac{\partial L_n(\alpha_n)}{\partial \alpha_n} = \mathrm{e}^{\alpha_n} \sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu - \mathrm{e}^{-\alpha_n} \sum_{\mu : y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu = 0$$

- That is

$$\mathrm{e}^{2\,\alpha_n} = \frac{\displaystyle\sum_{\mu : y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\displaystyle\sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu} \qquad \text{or} \qquad \alpha_n = \frac{1}{2} \log\left(\frac{\displaystyle\sum_{\mu : y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\displaystyle\sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu} \right)$$

Choosing Weights

- We now choose the weight α_n to minimise the error $L_n(\alpha_n)$

$$\frac{\partial\, L_n(\alpha_n)}{\partial\, \alpha_n} = \mathrm{e}^{\alpha_n} \sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu \; - \mathrm{e}^{-\alpha_n} \sum_{\mu: y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu = 0$$

- That is

$$\mathrm{e}^{2\,\alpha_n} = \frac{\displaystyle\sum_{\mu: y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\displaystyle\sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu} \qquad \text{or} \qquad \alpha_n = \frac{1}{2} \log\left(\frac{\displaystyle\sum_{\mu: y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\displaystyle\sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu} \right)$$

Algorithm

1. Start with a set of weak learners $\mathcal{W}$
2. Associate a weight, w_n^μ, with every data point $(\boldsymbol{x}^\mu, y^\mu)$, $\mu = 1, 2, \ldots, m$
3. Initially $w_0^\mu = 1$
4. Choose the weak learning, $\hat{h}_n(\boldsymbol{x}) \in \mathcal{W}$, that minimises $\displaystyle\sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$
5. Update predictor $C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \alpha_n\, \hat{h}_n(\boldsymbol{x})$ where
$$\alpha_n = \frac{1}{2}\log\left(\frac{\sum_{\mu: y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}\right)$$
6. Update $w_{n+1}^\mu = w_n^\mu\, \mathrm{e}^{-y^\mu\, \alpha_n\, \hat{h}_n(\boldsymbol{x}^\mu)}$
7. Go to 4

Algorithm

1. Start with a set of weak learners $\mathcal{W}$
2. Associate a weight, w_n^μ, with every data point $(\boldsymbol{x}^\mu, y^\mu)$, $\mu = 1, 2, \ldots, m$
3. Initially $w_0^\mu = 1$
4. Choose the weak learning, $\hat{h}_n(\boldsymbol{x}) \in \mathcal{W}$, that minimises $\sum\limits_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$
5. Update predictor $C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \alpha_n \, \hat{h}_n(\boldsymbol{x})$ where
$$\alpha_n = \tfrac{1}{2} \log \left(\frac{\sum\limits_{\mu: y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\sum\limits_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu} \right)$$
6. Update $w_{n+1}^\mu = w_n^\mu \, \mathrm{e}^{-y^\mu \, \alpha_n \, \hat{h}_n(\boldsymbol{x}^\mu)}$
7. Go to 4

Algorithm

1. Start with a set of weak learners $\mathcal{W}$
2. Associate a weight, w_n^μ, with every data point $(\boldsymbol{x}^\mu, y^\mu)$, $\mu = 1, 2, \ldots, m$
3. Initially $w_0^\mu = 1$
4. Choose the weak learning, $\hat{h}_n(\boldsymbol{x}) \in \mathcal{W}$, that minimises $\sum\limits_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$
5. Update predictor $C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \alpha_n \, \hat{h}_n(\boldsymbol{x})$ where
$$\alpha_n = \frac{1}{2} \log\left(\frac{\sum\limits_{\mu: y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\sum\limits_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu} \right)$$
6. Update $w_{n+1}^\mu = w_n^\mu \, \mathrm{e}^{-y^\mu \, \alpha_n \, \hat{h}_n(\boldsymbol{x}^\mu)}$
7. Go to 4

Algorithm

1. Start with a set of weak learners $\mathcal{W}$
2. Associate a weight, w_n^μ, with every data point $(\boldsymbol{x}^\mu, y^\mu)$, $\mu = 1, 2, \ldots, m$
3. Initially $w_0^\mu = 1$ (large weight, w_n^μ, means $(\boldsymbol{x}^\mu, y^\mu)$ is poorly classified)
4. Choose the weak learning, $\hat{h}_n(\boldsymbol{x}) \in \mathcal{W}$, that minimises $\displaystyle\sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$
5. Update predictor $C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \alpha_n \, \hat{h}_n(\boldsymbol{x})$ where
$$\alpha_n = \frac{1}{2} \log \left(\frac{\sum_{\mu : y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu} \right)$$
6. Update $w_{n+1}^\mu = w_n^\mu \, \mathrm{e}^{-y^\mu \, \alpha_n \, \hat{h}_n(\boldsymbol{x}^\mu)}$
7. Go to 4

Algorithm

1. Start with a set of weak learners $\mathcal{W}$

2. Associate a weight, w_n^μ, with every data point $(\boldsymbol{x}^\mu, y^\mu)$, $\mu = 1, 2, \ldots, m$

3. Initially $w_0^\mu = 1$ (large weight, w_n^μ, means $(\boldsymbol{x}^\mu, y^\mu)$ is poorly classified)

4. Choose the weak learning, $\hat{h}_n(\boldsymbol{x}) \in \mathcal{W}$, that minimises $\sum\limits_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$

5. Update predictor $C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \alpha_n \, \hat{h}_n(\boldsymbol{x})$ where
$$\alpha_n = \frac{1}{2} \log \left(\frac{\sum\limits_{\mu : y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\sum\limits_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu} \right)$$

6. Update $w_{n+1}^\mu = w_n^\mu \, \mathrm{e}^{-y^\mu \, \alpha_n \, \hat{h}_n(\boldsymbol{x}^\mu)}$

7. Go to 4

Algorithm

1. Start with a set of weak learners $\mathcal{W}$

2. Associate a weight, w_n^μ, with every data point $(\boldsymbol{x}^\mu, y^\mu)$, $\mu = 1, 2, \ldots, m$

3. Initially $w_0^\mu = 1$ (large weight, w_n^μ, means $(\boldsymbol{x}^\mu, y^\mu)$ is poorly classified)

4. Choose the weak learning, $\hat{h}_n(\boldsymbol{x}) \in \mathcal{W}$, that minimises $\displaystyle\sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$

5. Update predictor $C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \alpha_n \, \hat{h}_n(\boldsymbol{x})$ where
$$\alpha_n = \frac{1}{2} \log\left(\frac{\sum_{\mu : y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu} \right)$$

6. Update $w_{n+1}^\mu = w_n^\mu \, \mathrm{e}^{-y^\mu \, \alpha_n \, \hat{h}_n(\boldsymbol{x}^\mu)}$

7. Go to 4

Algorithm

1. Start with a set of weak learners $\mathcal{W}$
2. Associate a weight, w_n^μ, with every data point $(\boldsymbol{x}^\mu, y^\mu)$, $\mu = 1, 2, \ldots, m$
3. Initially $w_0^\mu = 1$ (large weight, w_n^μ, means $(\boldsymbol{x}^\mu, y^\mu)$ is poorly classified)
4. Choose the weak learning, $\hat{h}_n(\boldsymbol{x}) \in \mathcal{W}$, that minimises $\displaystyle\sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$
5. Update predictor $C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \alpha_n \, \hat{h}_n(\boldsymbol{x})$ where
$$\alpha_n = \frac{1}{2} \log \left(\frac{\sum_{\mu : y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu} \right)$$
6. Update $w_{n+1}^\mu = w_n^\mu \, \mathrm{e}^{-y^\mu \, \alpha_n \, \hat{h}_n(\boldsymbol{x}^\mu)}$
7. Go to 4

Algorithm

1. Start with a set of weak learners $\mathcal{W}$
2. Associate a weight, w_n^μ, with every data point $(\boldsymbol{x}^\mu, y^\mu)$, $\mu = 1, 2, \ldots, m$
3. Initially $w_0^\mu = 1$ (large weight, w_n^μ, means $(\boldsymbol{x}^\mu, y^\mu)$ is poorly classified)
4. Choose the weak learning, $\hat{h}_n(\boldsymbol{x}) \in \mathcal{W}$, that minimises $\displaystyle\sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$
5. Update predictor $C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \alpha_n \, \hat{h}_n(\boldsymbol{x})$ where
$$\alpha_n = \frac{1}{2} \log \left(\frac{\displaystyle\sum_{\mu : y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\displaystyle\sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu} \right)$$
6. Update $w_{n+1}^\mu = w_n^\mu \, \mathrm{e}^{-y^\mu \, \alpha_n \, \hat{h}_n(\boldsymbol{x}^\mu)}$
7. Go to 4

Performance

- Adaboost works well with weak learners, usually out-performing bagging
- It doesn't work well with strong learners (tends to over-fit)
- It is limited to binary classification (there are generalisation, but they are difficult to get to work)
- It has fallen from fashion
- In contrast **gradient boosting** used for regression is very popular

Performance

- Adaboost works well with weak learners, usually out-performing bagging
- It doesn't work well with strong learners (tends to over-fit)
- It is limited to binary classification (there are generalisation, but they are difficult to get to work)
- It has fallen from fashion
- In contrast **gradient boosting** used for regression is very popular

Performance

- Adaboost works well with weak learners, usually out-performing bagging
- It doesn't work well with strong learners (tends to over-fit)
- It is limited to binary classification (there are generalisation, but they are difficult to get to work)
- It has fallen from fashion
- In contrast **gradient boosting** used for regression is very popular

Performance

- Adaboost works well with weak learners, usually out-performing bagging
- It doesn't work well with strong learners (tends to over-fit)
- It is limited to binary classification (there are generalisation, but they are difficult to get to work)
- It has fallen from fashion
- In contrast **gradient boosting** used for regression is very popular

Performance

- Adaboost works well with weak learners, usually out-performing bagging
- It doesn't work well with strong learners (tends to over-fit)
- It is limited to binary classification (there are generalisation, but they are difficult to get to work)
- It has fallen from fashion
- In contrast **gradient boosting** used for regression is very popular

Gradient Boosting

- In gradient boosting we again build a strong learner as a linear combination of weak learners

$$C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \hat{h}_n(\boldsymbol{x})$$

- At each step $\hat{h}_n(\boldsymbol{x})$ is trained to predict the residual error, $\Delta_{n-1} = y - C_{n-1}(\boldsymbol{x})$, (i.e. the target minus the current prediction)

- (This difference looks a bit like a gradient hence the rather confusing name)

- Gradient boosting also mainly used with regression decision trees

Gradient Boosting

- In gradient boosting we again build a strong learner as a linear combination of weak learners

$$C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \hat{h}_n(\boldsymbol{x})$$

- At each step $\hat{h}_n(\boldsymbol{x})$ is trained to predict the residual error, $\Delta_{n-1} = y - C_{n-1}(\boldsymbol{x})$, (i.e. the target minus the current prediction)

- (This difference looks a bit like a gradient hence the rather confusing name)

- Gradient boosting also mainly used with regression decision trees

Gradient Boosting

- In gradient boosting we again build a strong learner as a linear combination of weak learners

$$C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \hat{h}_n(\boldsymbol{x})$$

- At each step $\hat{h}_n(\boldsymbol{x})$ is trained to predict the residual error, $\Delta_{n-1} = y - C_{n-1}(\boldsymbol{x})$, (i.e. the target minus the current prediction)
- (This difference looks a bit like a gradient hence the rather confusing name)
- Gradient boosting also mainly used with regression decision trees

Gradient Boosting

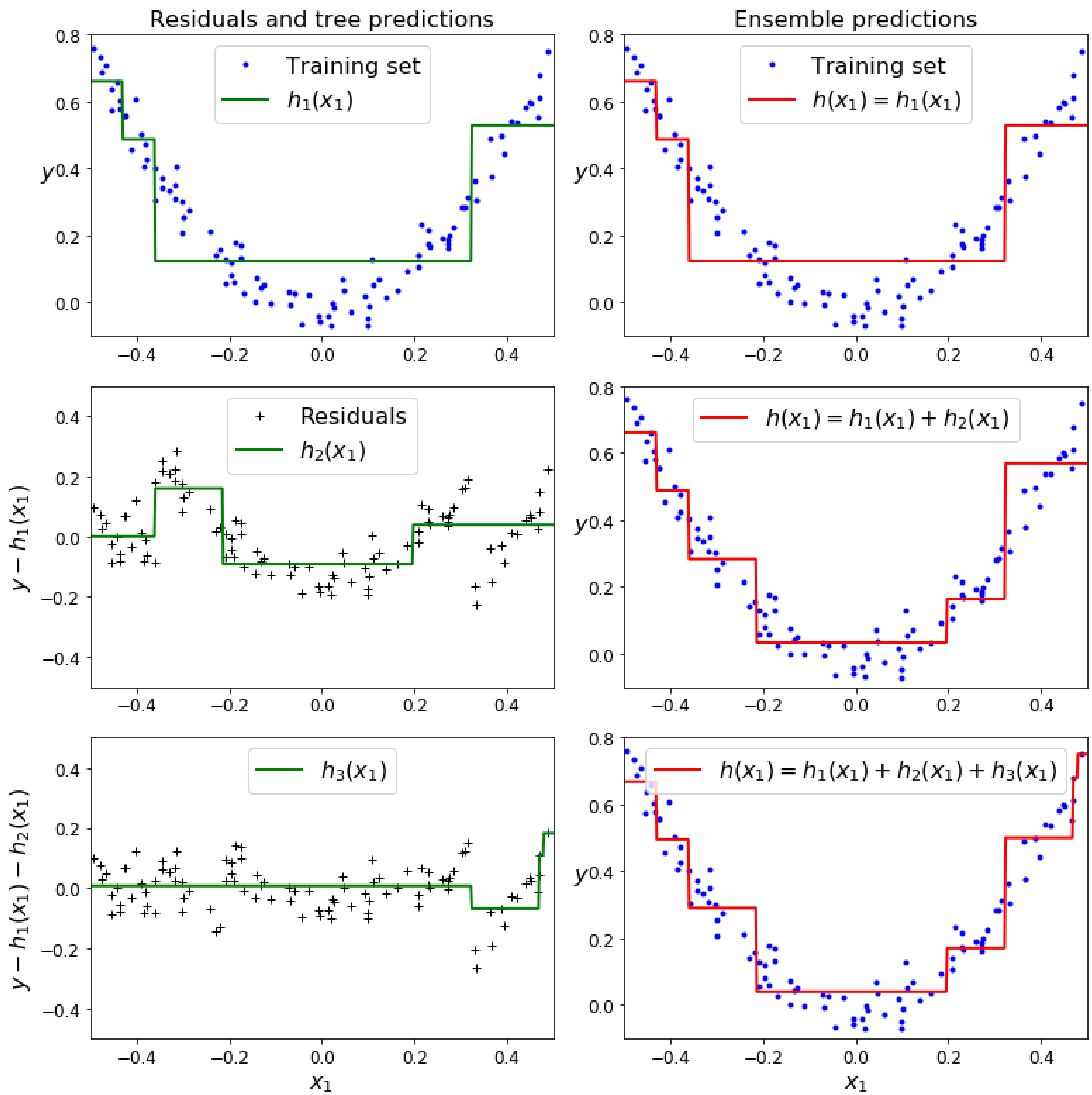
- In gradient boosting we again build a strong learner as a linear combination of weak learners

$$C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \hat{h}_n(\boldsymbol{x})$$

- At each step $\hat{h}_n(\boldsymbol{x})$ is trained to predict the residual error, $\Delta_{n-1} = y - C_{n-1}(\boldsymbol{x})$, (i.e. the target minus the current prediction)

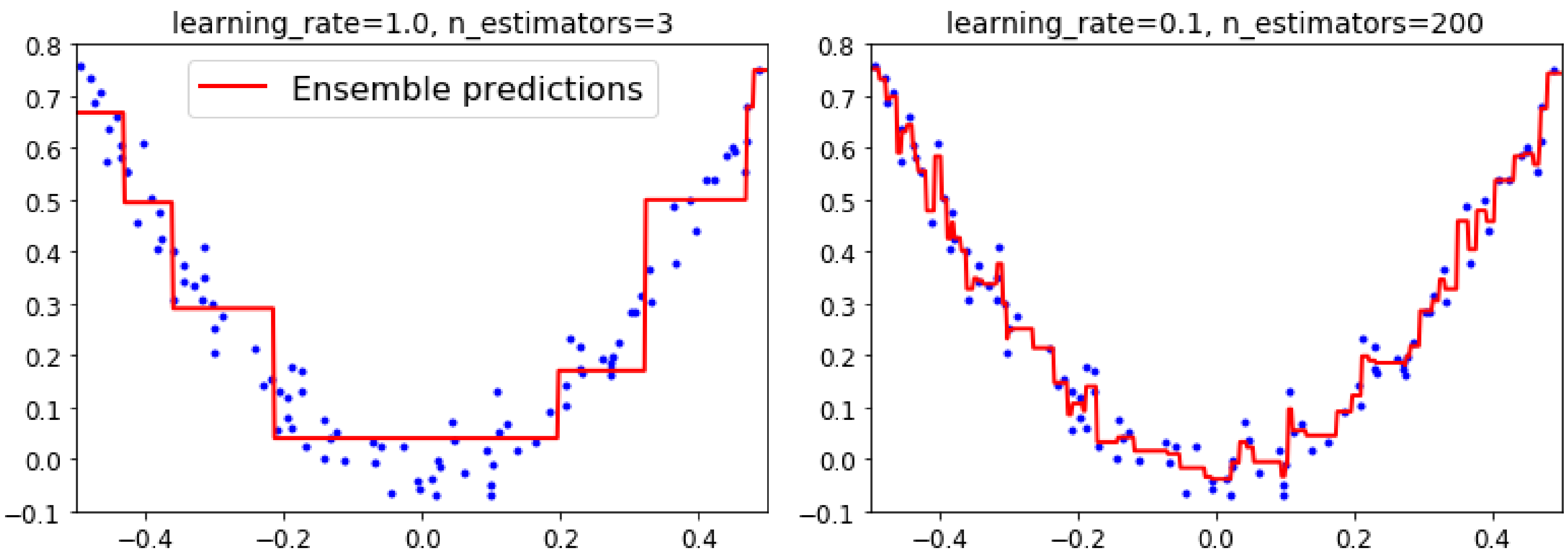
- (This difference looks a bit like a gradient hence the rather confusing name)

- Gradient boosting also mainly used with regression decision trees


Residuals and tree predictions
Training set
$h_1(x_1)$
Ensemble predictions
Training set
$h(x_1) = h_1(x_1)$
Residuals
$h_2(x_1)$
$h(x_1) = h_1(x_1) + h_2(x_1)$
$h_3(x_1)$
$h(x_1) = h_1(x_1) + h_2(x_1) + h_3(x_1)$
y
$y - h_1(x_1)$
$y - h_1(x_1) - h_2(x_1)$
x_1

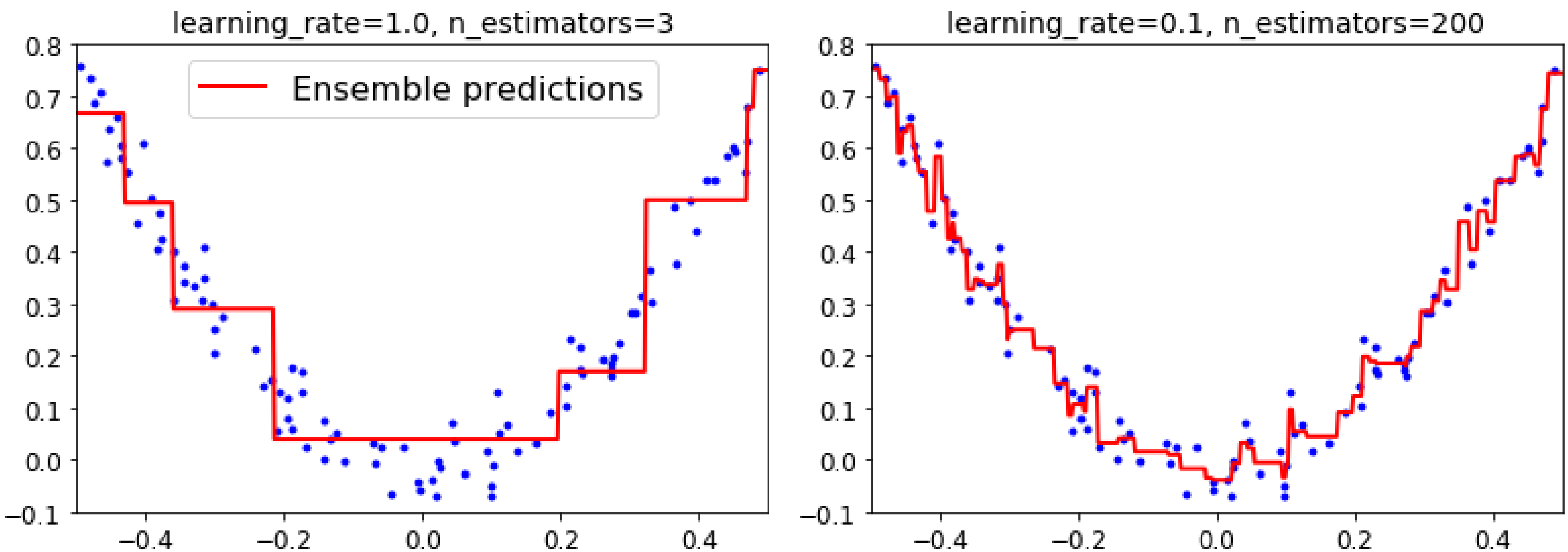
Keep On Going

- We can keep on going



Keep On Going

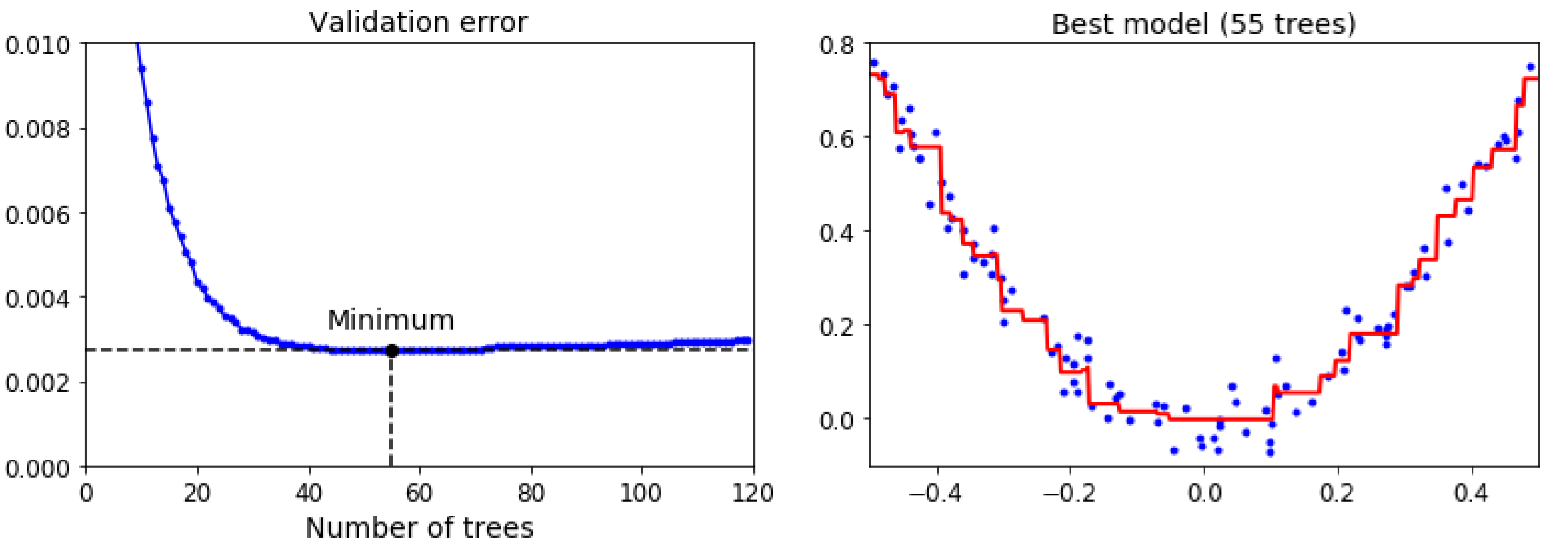
- We can keep on going



- But we will over-fit eventually

Early Stopping

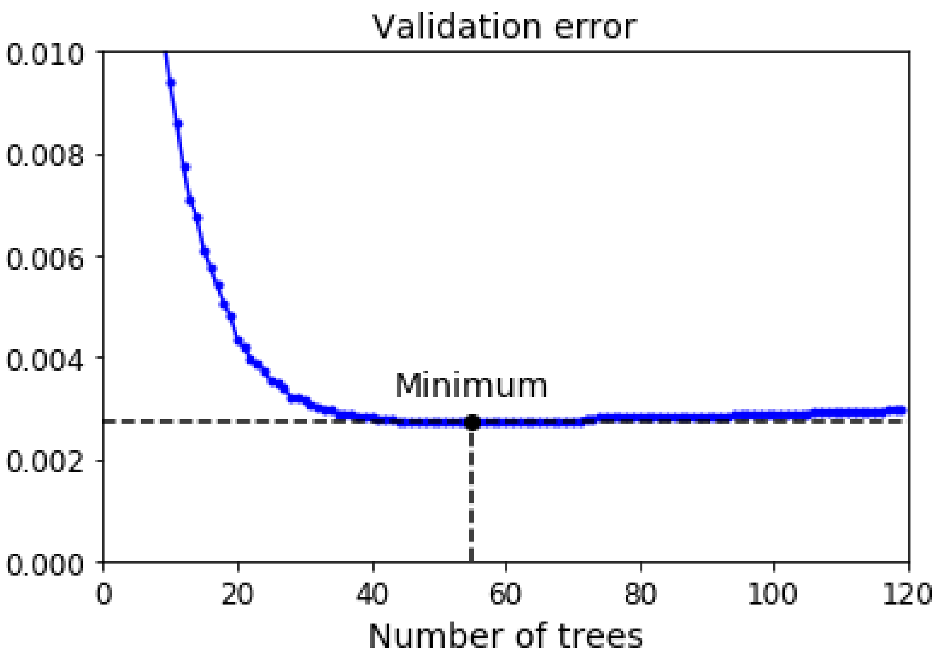
- Like many algorithms we often get better results by early stopping

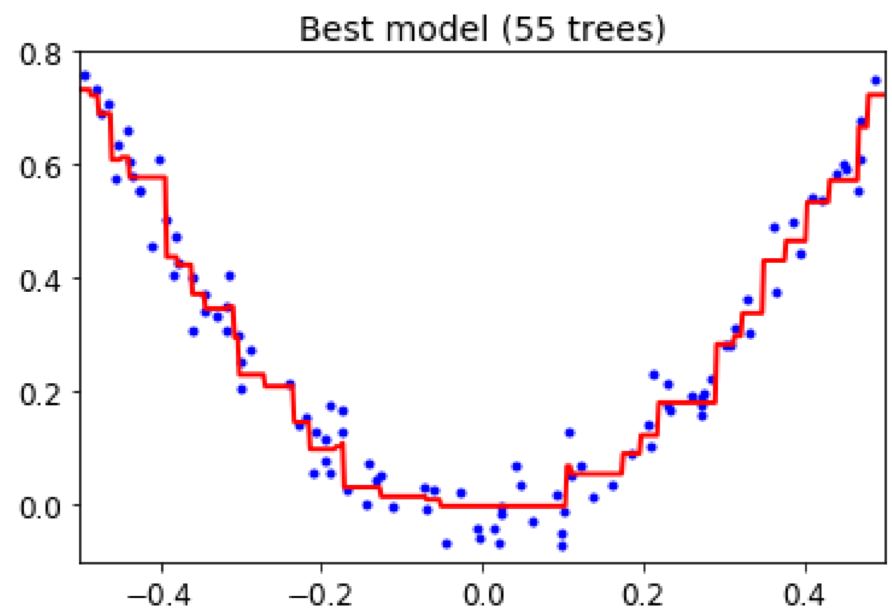


- Use cross-validation against a validation set to decide when to stop

Early Stopping

- Like many algorithms we often get better results by early stopping





- Use cross-validation against a validation set to decide when to stop

XGBoost

- XGBoost is an implementation of gradient boosting that won the Higg's Boson challenge and regularly wins Kaggle competitions
- XGBoost stands for eXtreme Gradient Boosting
- It was much faster than most gradient boosting algorithms and scales to billions of training data points—although GBM is often better
- It uses a cleverly chosen regularisation term to favour simple trees
- Finds a clever way to approximately minimise error plus regulariser very fast

XGBoost

- XGBoost is an implementation of gradient boosting that won the Higg's Boson challenge and regularly wins Kaggle competitions
- XGBoost stands for eXtreme Gradient Boosting
- It was much faster than most gradient boosting algorithms and scales to billions of training data points—although GBM is often better
- It uses a cleverly chosen regularisation term to favour simple trees
- Finds a clever way to approximately minimise error plus regulariser very fast

XGBoost

- XGBoost is an implementation of gradient boosting that won the Higg's Boson challenge and regularly wins Kaggle competitions
- XGBoost stands for eXtreme Gradient Boosting
- It was much faster than most gradient boosting algorithms and scales to billions of training data points—although GBM is often better
- It uses a cleverly chosen regularisation term to favour simple trees
- Finds a clever way to approximately minimise error plus regulariser very fast

XGBoost

- XGBoost is an implementation of gradient boosting that won the Higg's Boson challenge and regularly wins Kaggle competitions
- XGBoost stands for eXtreme Gradient Boosting
- It was much faster than most gradient boosting algorithms and scales to billions of training data points—although GBM is often better
- It uses a cleverly chosen regularisation term to favour simple trees
- Finds a clever way to approximately minimise error plus regulariser very fast

XGBoost

- XGBoost is an implementation of gradient boosting that won the Higg's Boson challenge and regularly wins Kaggle competitions
- XGBoost stands for eXtreme Gradient Boosting
- It was much faster than most gradient boosting algorithms and scales to billions of training data points—although GBM is often better
- It uses a cleverly chosen regularisation term to favour simple trees
- Finds a clever way to approximately minimise error plus regulariser very fast

XGBoost

- XGBoost is an implementation of gradient boosting that won the Higg's Boson challenge and regularly wins Kaggle competitions
- XGBoost stands for eXtreme Gradient Boosting
- It was much faster than most gradient boosting algorithms and scales to billions of training data points—although GBM is often better
- It uses a cleverly chosen regularisation term to favour simple trees
- Finds a clever way to approximately minimise error plus regulariser very fast
- Rather a bodge of optimisation hacks

Conclusion

- Ensemble methods have proved themselves to be very powerful
- Tend to work best with very simple models (true of random forest and boosting)
- XGBoost or GBM are currently the best methods for tabular data (particular for large training sets)
- For images, signal and speech deep learning can give very significant advantage
- Probabilistic models can be better if you have a good model

Conclusion

- Ensemble methods have proved themselves to be very powerful
- Tend to work best with very simple models (true of random forest and boosting)
- XGBoost or GBM are currently the best methods for tabular data (particular for large training sets)
- For images, signal and speech deep learning can give very significant advantage
- Probabilistic models can be better if you have a good model

Conclusion

- Ensemble methods have proved themselves to be very powerful
- Tend to work best with very simple models (true of random forest and boosting)—seems to reduce over-fitting
- XGBoost or GBM are currently the best methods for tabular data (particular for large training sets)
- For images, signal and speech deep learning can give very significant advantage
- Probabilistic models can be better if you have a good model

Conclusion

- Ensemble methods have proved themselves to be very powerful
- Tend to work best with very simple models (true of random forest and boosting)—seems to reduce over-fitting
- XGBoost or GBM are currently the best methods for tabular data (particular for large training sets)
- For images, signal and speech deep learning can give very significant advantage
- Probabilistic models can be better if you have a good model

Conclusion

- Ensemble methods have proved themselves to be very powerful
- Tend to work best with very simple models (true of random forest and boosting)—seems to reduce over-fitting
- XGBoost or GBM are currently the best methods for tabular data (particular for large training sets)—probably
- For images, signal and speech deep learning can give very significant advantage
- Probabilistic models can be better if you have a good model

Conclusion

- Ensemble methods have proved themselves to be very powerful
- Tend to work best with very simple models (true of random forest and boosting)—seems to reduce over-fitting
- XGBoost or GBM are currently the best methods for tabular data (particular for large training sets)—probably
- For images, signal and speech deep learning can give very significant advantage
- Probabilistic models can be better if you have a good model

Conclusion

- Ensemble methods have proved themselves to be very powerful
- Tend to work best with very simple models (true of random forest and boosting)—seems to reduce over-fitting
- XGBoost or GBM are currently the best methods for tabular data (particular for large training sets)—probably
- For images, signal and speech deep learning can give very significant advantage
- Probabilistic models can be better if you have a good model