Decision Trees and Random Forests

Lecture 7 - DAMLF | ML1

human + machine INTELLIGENCE

Frankfurt School

Parametric vs Nonparametric Models

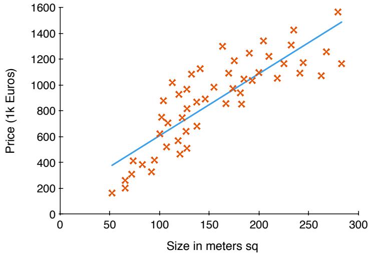
Parametric Models

Have a **fixed** number of parameters.

- + Simple
- + Fast
- – Makes strong assumptions about data
- – Prone to underfit*

  **except high-capacity variants that depend on regularization*

Parametric vs Nonparametric Models



Linear Regression: $h(x; \boldsymbol{\theta}) = \boldsymbol{\theta}^T x$

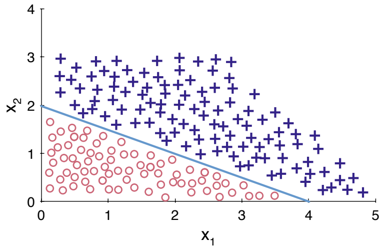
Parametric Models

Have a **fixed** number of parameters.

+ Simple
+ Fast
— Makes strong assumptions about data
— Prone to underfit*

  **except high-capacity variants that depend on regularization*

Parametric vs Nonparametric Models



Logistic Regression: $h(x; \boldsymbol{\theta}) = g\left(\boldsymbol{\theta}^T x\right)$

Parametric Models

Have a **fixed** number of parameters.

+ Simple
+ Fast
– Makes strong assumptions about data
– Prone to underfit*

  **except high-capacity variants that depend on regularization*

Parametric vs Nonparametric Models

Parametric Models

Have a **fixed** number of parameters.

+ Simple
+ Fast
– Makes strong assumptions about data
– Prone to underfit*

**except high-capacity variants that depend on regularization*

Nonparametric Models

The number of parameters **grows** with the amount of training data.

+ Makes few if any assumptions about data
+ Can fit a larger class of functional forms
– Slow
– Prone to overfit

Parametric vs Nonparametric Models

Parametric Models

Have a **fixed** number of parameters.

+ Simple
+ Fast
– Makes strong assumptions about data
– Prone to underfit*

**except high-capacity variants that depend on regularization*

Two Nonparametric Models

- **Support Vector Machines**
  kernel method
- **Decision Trees**
  adaptive basis function model

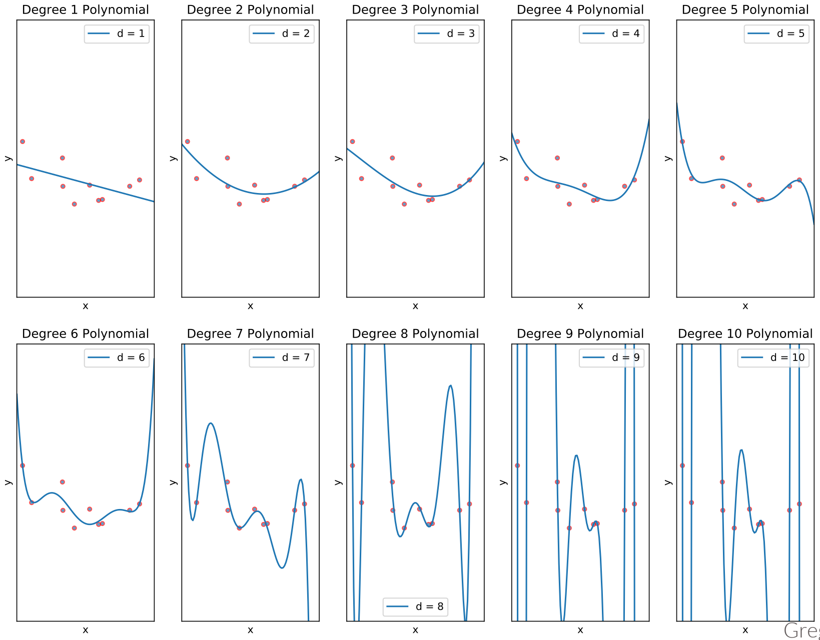
Nonparametric Models

The number of parameters **grows** with the amount of training data.

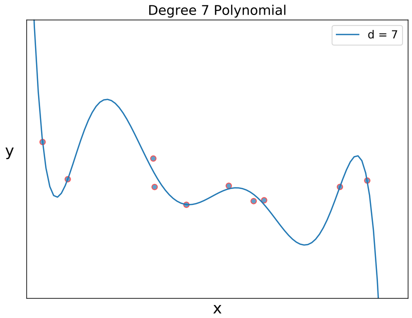
+ Makes few if any assumptions about data
+ Can fit a larger class of functional forms
– Slow
– Prone to overfit

Kernels and Adaptive Basis Functions

Review: Polynomial Regression



Review: Polynomial Regression



Polynomial Regression

The degree 7 polynomial

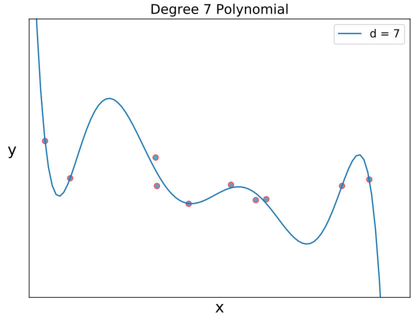
$$h(x;\boldsymbol{\theta}) = \theta_0 + \theta_1 x, +\theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5 + \theta_6 x^6 + \theta_7 x^7$$

may be rewritten as

$$\begin{aligned} h(x;\boldsymbol{\theta}) &= \boldsymbol{\theta}^T \mathbf{z} \\ &= \theta_0 + \theta_1 z_1, \theta_2 z_2 + \theta_3 z_3 + \theta_4 z + \theta_5 z_5 + \theta_6 z_6 + \theta_7 z_7 \end{aligned}$$

where $\mathbf{z} = [1, x, x^2, x^3, x^4, x^5, x^6, x^7]$.

Review: Polynomial Regression



POLYNOMIAL REGRESSION

The degree 7 polynomial

$$h(x;\boldsymbol{\theta}) = \theta_0 + \theta_1 x, + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5 + \theta_6 x^6 + \theta_7 x^7$$

may be rewritten as

$$\begin{aligned} h(x;\boldsymbol{\theta}) &= \boldsymbol{\theta}^T \mathbf{z} \\ &= \theta_0 + \theta_1 z_1, \theta_2 z_2 + \theta_3 z_3 + \theta_4 z + \theta_5 z_5 + \theta_6 z_6 + \theta_7 z_7 \end{aligned}$$

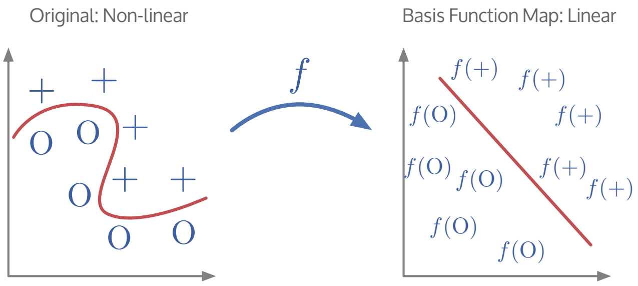
where $\mathbf{z} = [1, x, x^2, x^3, x^4, x^5, x^6, x^7]$.

Suppose $f_j(x) = \mathbf{z}$ is a polynomial **basis function** of x whose range is $z_j = x^j$. Then, we may rewrite $h(x;\boldsymbol{\theta})$ again as:

$$h(x;\boldsymbol{\theta}) = \boldsymbol{\theta}^T f_j(x) \qquad \text{for } j = 1, \ldots, 7$$

Review: Polynomial Logistic Regression

Non-linear Decision Boundaries



Review: Polynomial Logistic Regression

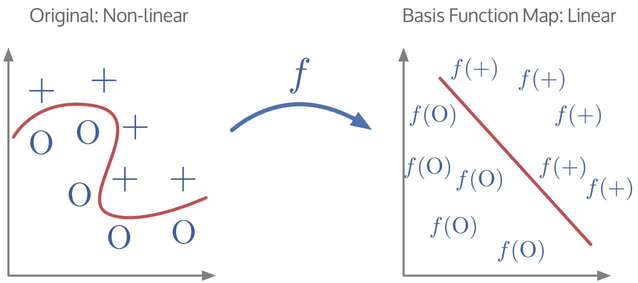
Problem: High Dimensionality.

Introducing non-linearity by explicitly engineering and computing a vector of new features for each input is impractical for multi-dimensional input.

Problem: Choosing *f*.

In addition to the ***curse of dimensionality****, you have to engineer the basis function, f.*

NON-LINEAR DECISION BOUNDARIES



Kernel Methods

Kernel Function

A kernel method has the form

$$h(x) = \boldsymbol{\theta}^T \phi(x)$$

where $\phi(x)$ is a list of **kernel functions** κ:

$$\phi(x) = [\kappa(x, \mu_1), \kappa(x, \mu_2), \ldots, \kappa(x, \mu_m)]$$

Kernel Function

A Measure of Similarity

$$\phi(x) = [\kappa(x, \mu_1), \kappa(x, \mu_2), \ldots, \kappa(x, \mu_m)]$$

The **kernel function** $\kappa(x, \mu_k)$ is a real-valued function that is typically:

(**non-negative**): $\kappa(x, \mu_k) \geqslant 0$

(**symmetric**): $\kappa(x, \mu_k) = \kappa(\mu_k, x)$

Kernel Function

OK: High Dimensionality.
Problem: Choosing κ

A Measure of Similarity

$$\phi(x) = [\kappa(x, \mu_1), \kappa(x, \mu_2), \ldots, \kappa(x, \mu_m)]$$

The **kernel function** $\kappa(x, \mu_k)$ is a real-valued function that is typically:

(**non-negative**): $\kappa(x, \mu_k) \geqslant 0$

(**symmetric**): $\kappa(x, \mu_k) = \kappa(\mu_k, x)$

Kernel functions are interpreted as **measures of similarity** between training example x and sample μ_k, where μ_k is either all m training examples or some (proper) subset of training examples.

Kernel Function

Some Kernel Functions

- **Radial Basis (RBF)**
  Gaussian kernel
  ARD kernel
- **Cosine similarity**
  to compare documents
- **Mercer kernels**
  aka, positive definite kernels
- **Linear kernels**
  for linearly separable features
- **Matern kernel**
  Gaussian process regression
- **String kernels**
  num. of strings in common
- **Pyramid match kernels**
  Mercer ker. for bag-of-words representation of images
- **Probability product**
  probability generative model
- **Fisher kernels**
  cf. string kernels for Markov chains

A Measure of Similarity

$$\phi(x) = [\kappa(x, \mu_1), \kappa(x, \mu_2), \ldots, \kappa(x, \mu_m)]$$

The **kernel function** $\kappa(x, \mu_k)$ is a real-valued function that is typically:

(**non-negative**): $\kappa(x, \mu_k) \geqslant 0$

(**symmetric**): $\kappa(x, \mu_k) = \kappa(\mu_k, x)$

Kernel functions are interpreted as **measures of similarity** between training example x and sample μ_k, where μ_k is either all m training examples or some (proper) subset of training examples.

Kernel Function

Some Kernel Functions

- **Radial Basis (RBF)**
  Gaussian kernel
  ARD kernel
- **Cosine similarity**
  to compare documents
- **Mercer kernels**
  aka, positive definite kernels
- **Linear kernels**
  for linearly separable features
- **Matern kernel**
  Gaussian process regression
- **String kernels**
  num. of strings in common
- **Pyramid match kernels**
  Mercer ker. for bag-of-words representation of images
- **Probability product**
  probability generative model
- **Fisher kernels**
  cf. string kernels for Markov chains

Template Matching

Kernel methods perform a type of **template matching**.

They compare the input x to **saved prototypes** μ_k.

Kernel Function

Some Kernel Functions

- **Radial Basis (RBF)**
  Gaussian kernel
  ARD kernel
- **Cosine similarity**
  to compare documents
- **Mercer kernels**
  aka, positive definite kernels
- **Linear kernels**
  for linearly separable features
- **Matern kernel**
  Gaussian process regression
- **String kernels**
  num. of strings in common
- **Pyramid match kernels**
  Mercer ker. for bag-of-words representation of images
- **Probability product**
  probability generative model
- **Fisher kernels**
  cf. string kernels for Markov chains

Template Matching

One way to **learn** the parameters of a kernel is to try the **ARD kernel**:

$$\kappa(\mathbf{x}, \mathbf{x}') = \theta_0 \exp\left(-\frac{1}{2}\sum_{j=1}^{n}\theta_j(x_j - x_j')^2\right)$$

where n is the total number of features, to estimate θ_j.

Kernel Methods

Require good kernels

Kernel methods depend on having **good base kernels** and are **computationally expensive**.

Kernel Methods

Require good kernels

Kernel methods depend on having **good base kernels** and are **computationally expensive**.

These limitations of kernel methods motivate other techniques that learn useful features from data.

Adaptive Basis Function Models

Get rid of kernels

Instead, **learn** useful **features** $\phi(\boldsymbol{x})$ directly from the input data.

Adaptive Basis Function Models

Get rid of kernels

Instead, **learn** useful **features** $\phi(\boldsymbol{x})$ directly from the input data.

Adaptive Basis Function models are of the form

$$h(\boldsymbol{x}) = \theta_0 + \sum_{k=1}^{K} \theta_k \phi_k(\boldsymbol{x})$$

where $\phi_k(\boldsymbol{x})$ is the k-th basis function learned from data.

Adaptive Basis Function Models (ABMs)

ABMs are not linear in parameters

$$h(\boldsymbol{x}) = \theta_0 + \sum_{k=1}^{K} \theta_k \phi_k(\boldsymbol{x})$$

Each basis function $\phi_k = \phi(\boldsymbol{x}, \boldsymbol{\omega}_k)$ is **parametric**.

Adaptive Basis Function Models (ABMs)

ABMs are not linear in parameters

$$h(\boldsymbol{x}) = \theta_0 + \sum_{k=1}^{K} \theta_k \phi_k(\boldsymbol{x})$$

Each basis function $\phi_k = \phi(\boldsymbol{x}, \boldsymbol{\omega}_k)$ is **parametric**.

Vector $\boldsymbol{\omega}_k$ parameterizes the basis function ϕ_k, and the full parameter set then is:

$$\boldsymbol{\theta} = [\theta_0, \theta_1, \ldots, \theta_n, \{\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_n\}]$$

Adaptive Basis Function Models (ABMs)

ABMs are not linear in parameters

$$h(\boldsymbol{x}) = \theta_0 + \sum_{k=1}^{K} \theta_k \phi_k(\boldsymbol{x})$$

Each basis function $\phi_k = \phi(\boldsymbol{x}, \boldsymbol{\omega}_k)$ is **parametric**.

Vector $\boldsymbol{\omega}_k$ parameterizes the basis function ϕ_k, and the full parameter set then is:

$$\boldsymbol{\theta} = [\theta_0, \theta_1, \ldots, \theta_n, \{\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_n\}]$$

But, the resulting model is **not linear** in $\boldsymbol{\theta}$.

Thus, ABMs are **nonparametric models**.

Adaptive Basis Function Models (ABMs)

Some Adaptive Basis Models

- **Classification and Regression Trees (CART)**
  aka, decision trees; random forests
- **Generalized Additive Models (GAM)**
  multivariate adaptive regression splines (MARS)
- **Feed Forward Neural Networks**
  aka, multi-layer perceptrons
- **Ensemble Learning**

ABMs are not linear in parameters

$$h(\boldsymbol{x}) = \theta_0 + \sum_{k=1}^{K} \theta_k \phi_k(\boldsymbol{x})$$

Each basis function $\phi_k = \phi(\boldsymbol{x}, \boldsymbol{\omega}_k)$ is **parametric**.

Vector $\boldsymbol{\omega}_k$ parameterizes the basis function ϕ_k, and the full parameter set then is:

$$\boldsymbol{\theta} = [\theta_0, \theta_1, \ldots, \theta_n, \{\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_n\}]$$

But, the resulting model is **not linear** in $\boldsymbol{\theta}$.

Thus, ABMs are **nonparametric models**.

Classification and Regression Trees

Decision Trees

CART Models

Classification and Regression Trees (CART) are adaptive basis function models.

CART models **recursively partition** the input, with each cell of the partition covering a region of the input.

Then, a **local model** is constructed for each of the resulting regions.

Decision Trees

Cart Models

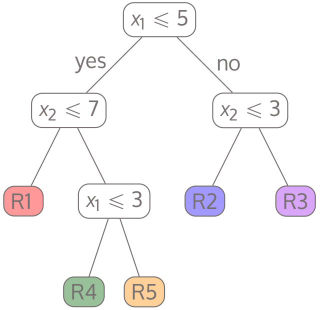
A CART model can be represented by a **tree**, where leaves correspond to the partitioned regions of the input space.

Decision Trees

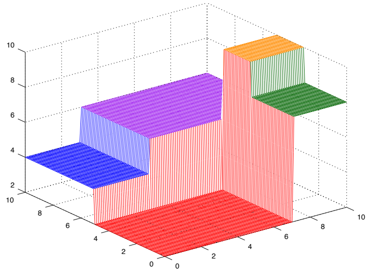
Cart Models

A CART model can be represented by a **tree**, where leaves correspond to the partitioned regions of the input space.

Example: two inputs, x_1 and x_2.



Decision Trees

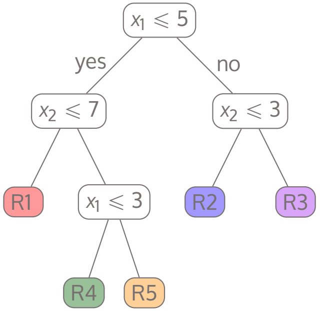


Based on Murphy (2012, Figure 16.1)

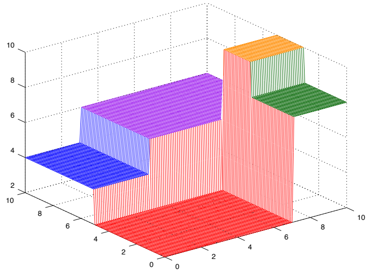
CART Models

A CART model can be represented by a **tree**, where leaves correspond to the partitioned regions of the input space.

Example: two inputs, x_1 and x_2.



Decision Trees



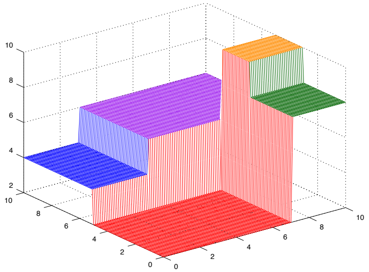
Based on Murphy (2012, Figure 16.1)

CART Models

In this example, the **2d space** is partitioned into 5 regions: R_1, R_2, R_3, R_4, R_5.

Partitions splits are **parallel to the axes** x_1, x_2.

Decision Trees



Based on Murphy (2012, Figure 16.1)

Mean response in each region

Define $\mathbf{1}_R(x) = 1$ if $x \in R$ and 0 otherwise.

Then, the equation

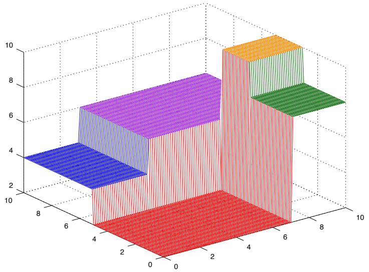
$$h(\mathbf{x}) = \mathbb{E}[y \mid \mathbf{x}] = \sum_{k=1}^{K} \theta_k \mathbf{1}_{R_k}(\mathbf{x}) = \sum_{k=1}^{K} \theta_k \phi(\mathbf{x}, \boldsymbol{\omega}_k)$$

specifies a **CART model**.

Where:

- R_k is the k'th region,
- θ_k is the **mean response** in R_k
- $\boldsymbol{\omega}_k$ encodes both the **choice of variables to split** and the **threshold values** for splitting each variable along the path from the root to the k'th leaf.

Decision Trees



Based on Murphy (2012, Figure 16.1)

Mean response in each region

Define $\mathbf{1}_R(x) = 1$ if $x \in R$ and 0 otherwise.

Then, the equation

$$h(\boldsymbol{x}) = \mathbb{E}[y \mid \boldsymbol{x}] = \sum_{k=1}^{K} \theta_k \mathbf{1}_{R_k}(\boldsymbol{x}) = \sum_{k=1}^{K} \theta_k \phi(\boldsymbol{x}, \boldsymbol{\omega}_k)$$

specifies a **CART model**.

CART is an **adaptive basis-function model**, where

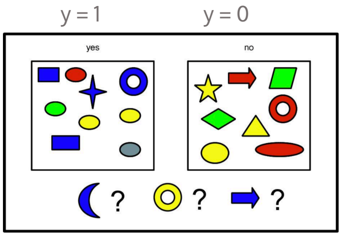
- **basis functions** define regions;
- **parameters** specify the response value in each region.

CART Classification

Classification

Instead of storing the mean response at each leaf, store the **distribution over class labels** at each leaf.

CART Classification



Example from Murphy (2012, Figure 1.1)

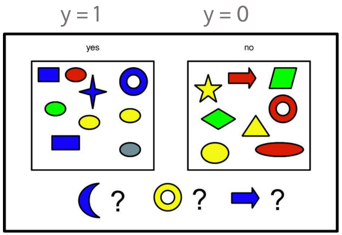
Classification

Instead of storing the mean response at each leaf, store the **distribution over class labels** at each leaf.

Example:

- Suppose x matches the color criterion **blue**. What is the probability that $y = 1$?

CART Classification



Example from Murphy (2012, Figure 1.1)

CLASSIFICATION

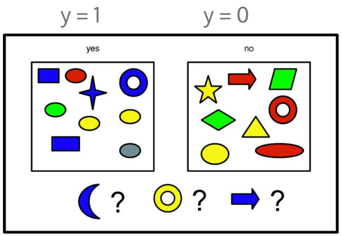
Instead of storing the mean response at each leaf, store the **distribution over class labels** at each leaf.

Example:

- Suppose x matches the color criterion **blue**. What is the probability that $y = 1$?

$$p(y = 1 \mid x \in \textbf{blue}) = 1$$

CART Classification



Example from Murphy (2012, Figure 1.1)

Classification

Instead of storing the mean response at each leaf, store the **distribution over class labels** at each leaf.

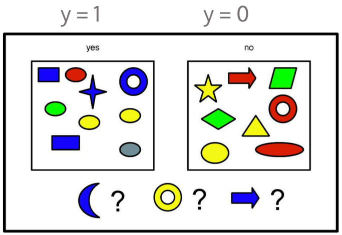
Example:

- Suppose x matches the color criterion **blue**. What is the probability that $y = 1$?

$$p(y = 1 \mid x \in \textbf{blue}) = 1$$

- Suppose x matches the color criterion **red** and the shape criterion **ellipse**. What is the probability that $y = 1$?

CART Classification



Example from Murphy (2012, Figure 1.1)

Classification

Instead of storing the mean response at each leaf, store the **distribution over class labels** at each leaf.

Example:

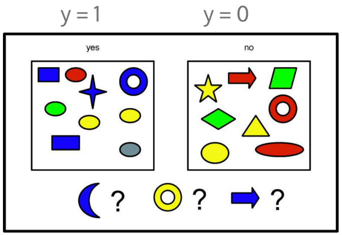
- Suppose x matches the color criterion **blue**. What is the probability that $y = 1$?

$$p(y = 1 \mid x \in \textbf{blue}) = 1$$

- Suppose x matches the color criterion **red** and the shape criterion **ellipse**. What is the probability that $y = 1$?

$$p(y = 1 \mid x \in \textbf{red} \cap \textbf{ellipse}) = \frac{1}{2}$$

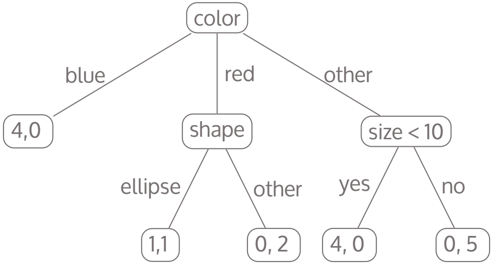
CART Classification



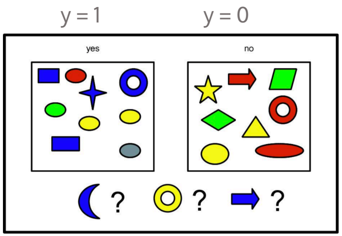
Example from Murphy (2012, Figure 1.1)

CLASSIFICATION

Instead of storing the mean response at each leaf, store the **distribution over class labels** at each leaf.



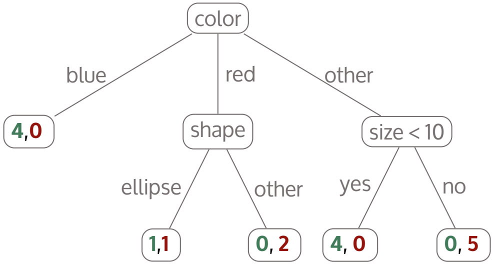
CART Classification



Example from Murphy (2012, Figure 1.1)

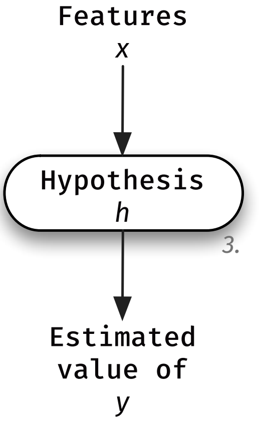
Classification

Instead of storing the mean response at each leaf, store the **distribution over class labels** at each leaf.



Each leaf (n_1, n_o) says there are n_1 **positive** examples in that path that match and n_o **negative** examples that do not match.

Hypothesis Representation

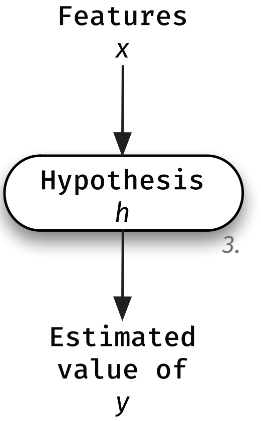


CART:

Hypothesis h computes the **expectation** of y given features $\mathbf{x}$:

$$h(\mathbf{x}) = \mathbb{E}[y \mid \mathbf{x}] = \sum_{k=1}^{K} \theta_k \mathbf{1}_{R_k}(\mathbf{x}) = \sum_{k=1}^{K} \theta_k \phi(\mathbf{x}, \boldsymbol{\omega}_k)$$

Hypothesis Representation



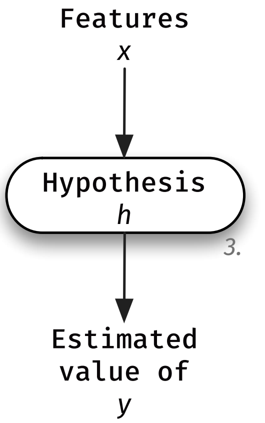
CART:

Hypothesis h computes the **expectation** of y given features $\mathbf{x}$:

$$h(\mathbf{x}) = \mathbb{E}\left[y \mid \mathbf{x}\right] = \sum_{k=1}^{K} \theta_k \mathbf{1}_{R_k}(\mathbf{x}) = \sum_{k=1}^{K} \theta_k \phi\left(\mathbf{x}, \boldsymbol{\omega}_k\right)$$

The form of the hypothesis depends on what variables are **split** at which **threshold**, which is encoded in $\boldsymbol{\omega}_k$ for each $k \in K$ leaf.

Hypothesis Representation



CART:

Hypothesis h computes the **expectation** of y given features $\mathbf{x}$:

$$h(\mathbf{x}) = \mathbb{E}\left[y \mid \mathbf{x}\right] = \sum_{k=1}^{K} \theta_k \mathbf{1}_{R_k}(\mathbf{x}) = \sum_{k=1}^{K} \theta_k \phi\left(\mathbf{x}, \boldsymbol{\omega}_k\right)$$

The form of the hypothesis depends on what variables are **split** at which **threshold**, which is encoded in $\boldsymbol{\omega}_k$ for each $k \in K$ leaf.

Each $\boldsymbol{\omega}_k$ is selected by a **split function**.

Growing a Tree

Optimal Partitions

The **split function** chooses the best feature j^* and the best threshold value t^* for that feature, as follows:

$$(j^*, t^*) = \arg \min_{\substack{j \in \{1,\ldots,n\} \\ t \in \mathcal{T}_j}} \min \boldsymbol{Cost}\left(\{\mathbf{x}_i, y_i : x_{i,j} \leqslant t\}\right) + \boldsymbol{Cost}\left(\{\mathbf{x}_i, y_i : x_{i,j} > t\}\right)$$

where:

- n is the dimension of the features
- $\mathcal{T}_j$ is the set of possible thresholds for feature j
- ***Cost*** is a cost function, to be defined
- j^* is the best feature from the n features
- t^* is the best threshold value for feature j^*

Cost Functions

$$(j^*, t^*) = \arg\min_{\substack{j \in \{1,\ldots,n\} \\ t \in \mathcal{T}_j}} \min \textbf{\textit{Cost}}\left(\{\mathbf{x}_i, y_i : x_{i,j} \leqslant t\}\right) + \textbf{\textit{Cost}}\left(\{\mathbf{x}_i, y_i : x_{i,j} > t\}\right)$$

Regression Cost Function:

$$\textbf{\textit{Cost}}\left(\mathcal{D}\right) = \sum_{i=1}^{|\mathcal{D}|} (\bar{y} - y_i)^2$$

where $\mathcal{D}$ is the specified data set, $|\mathcal{D}| = m$, and $\bar{y}$ is the **mean** of the target y within the set of data $\mathcal{D}$:

$$\bar{y} = \frac{1}{m} \sum_{i=1}^{m} y_i$$

Cost Functions

$$(j^*, t^*) = \arg\min_{\substack{j \in \{1,\dots,n\} \\ t \in \mathcal{T}_j}} \min \textbf{Cost}\left(\{\mathbf{x}_i, y_i : x_{i,j} \leqslant t\}\right) + \textbf{Cost}\left(\{\mathbf{x}_i, y_i : x_{i,j} > t\}\right)$$

Regression Cost Function:

$$\textbf{Cost}\,(\mathcal{D}) = \sum_{i=1}^{|\mathcal{D}|} (\bar{y} - y_i)^2$$

where $\mathcal{D}$ is the specified data set, $|\mathcal{D}| = m$, and $\bar{y}$ is the **mean** of the target y within the set of data $\mathcal{D}$:

$$\bar{y} = \frac{1}{m}\sum_{i=1}^{m} y_i$$

Alternatively, a linear regression model can be **fit to each leaf** using the features selected for that path as input, then residual error can be measured.

Cost Functions

$$(j^*, t^*) = \arg\min_{\substack{j\in\{1,\ldots,n\}\\ t\in\mathcal{T}_j}} \min \textbf{\textit{Cost}}\left(\{\mathbf{x}_i, y_i : x_{i,j} \leqslant t\}\right) + \textbf{\textit{Cost}}\left(\{\mathbf{x}_i, y_i : x_{i,j} > t\}\right)$$

Classification:

Step 1.

Compute the proportion of class c observations in leaf j passing test $X_j < t$, written $\hat{\pi}_{c,j}$, by

$$\hat{\pi}_{c,j} = \frac{1}{|\mathcal{D}_j|} \sum_{i=1}^{|\mathcal{D}_j|} \mathbf{1}(y_i = c)$$

where $\mathcal{D}_j$ is the data in leaf j and the indicator $\mathbf{1}(y_i = c) = 1$, if y_i has class label c, 0 otherwise.

Cost Functions

$$(j^*, t^*) = \arg\min_{\substack{j \in \{1,\ldots,n\} \\ t \in \mathcal{T}_j}} \min \textbf{\textit{Cost}}\left(\{\mathbf{x}_i, y_i : x_{i,j} \leqslant t\}\right) + \textbf{\textit{Cost}}\left(\{\mathbf{x}_i, y_i : x_{i,j} > t\}\right)$$

Classification:

Step 1.

Compute the **proportion** of class c observations in leaf j passing test $X_j < t$, written $\hat{\pi}_{c,j}$, by

$$\hat{\pi}_{c,j} = \frac{1}{|\mathcal{D}_j|} \sum_{i=1}^{|\mathcal{D}_j|} \mathbf{1}(y_i = c)$$

where $\mathcal{D}_j$ is the data in leaf j and the **indicator** $\mathbf{1}(y_i = c) = 1$, if y_i has class label c, 0 otherwise.

$\hat{\pi}_{c,j}$ is a **Maximum Likelihood Estimator** for the distribution $p(c \mid X_j < t)$

Cost Functions

$$(j^*, t^*) = \arg\min_{\substack{j \in \{1,\ldots,n\} \\ t \in \mathcal{T}_j}} \min \textit{\textbf{Cost}}\left(\{\mathbf{x}_i, y_i : x_{i,j} \leqslant t\}\right) + \textit{\textbf{Cost}}\left(\{\mathbf{x}_i, y_i : x_{i,j} > t\}\right)$$

Classification:

Step 2.

There are many **cost functions** to measure the **quality of a split** in a CART classification model.

Cost Functions

$$(j^*, t^*) = \arg\min_{\substack{j \in \{1,\ldots,n\} \\ t \in \mathcal{T}_j}} \min \textbf{\textit{Cost}}\left(\{\boldsymbol{x}_i, y_i : x_{i,j} \leqslant t\}\right) + \textbf{\textit{Cost}}\left(\{\boldsymbol{x}_i, y_i : x_{i,j} > t\}\right)$$

Classification:

Step 2.

There are many **cost functions** to measure the **quality of a split** in a CART classification model.

Three common error measures for evaluating a proposed partition:

1. **Misclassification rate**
2. **Entropy**
3. **Gini index**

Misclassification Rate

1. Misclassification Error

Let $\hat{y}_c = \arg\max_c \hat{\pi}_c$ denote the **most probable class** label c for some (index omitted) region.

The **misclassification rate** error measure is then

$$\frac{1}{|\mathcal{D}_j|}\sum_{i=1}^{|\mathcal{D}_j|}\mathbf{1}\left(y_i \neq \hat{y}_{c,j}\right) = 1 - \hat{\pi}_{\hat{y},j}$$

Entropy

2. Entropy

The **entropy** measure is

$$-\sum_{c=1}^{C} \hat{\pi}_c \log \hat{\pi}_c$$

Entropy

2. Entropy

The **entropy** measure is

$$\mathbb{H}(\hat{\pi}) = -\sum_{c=1}^{C} \hat{\pi}_c \log \hat{\pi}_c$$

Entropy

2. Entropy

The **entropy** measure is

$$\mathbb{H}(\hat{\pi}) = -\sum_{c=1}^{C} \hat{\pi}_c \log \hat{\pi}_c$$

Minimizing entropy is equivalent to **maximizing information gain** between the test $X_j < t$ and the class label $Y = y$

Entropy

2. Entropy

The **entropy** measure is

$$\mathbb{H}(\hat{\pi}) = -\sum_{c=1}^{C} \hat{\pi}_c \log \hat{\pi}_c$$

Minimizing entropy is equivalent to **maximizing information gain** between the test $X_j < t$ and the class label $Y = y$

Information Gain

The **info gain** of test $X_j < t$ *wrt* to label Y:

$$\text{InfoGain}(X_j < t, Y) := \mathbb{H}(Y) - \mathbb{H}(Y|X_j < t)$$

Entropy

2. Entropy

The **entropy** measure is

$$\mathbb{H}(\hat{\pi}) = -\sum_{c=1}^{C} \hat{\pi}_c \log \hat{\pi}_c$$

Minimizing entropy is equivalent to **maximizing information gain** between the test $X_j < t$ and the class label $Y = y$

Information Gain

The **info gain** of test $X_j < t$ *wrt* to label Y:

$$\text{InfoGain}(X_j < t, Y) := \mathbb{H}(Y) - \mathbb{H}(Y|X_j < t)$$

where:

$$\begin{aligned}\text{InfoGain}(X_j < t, Y) = &\left(-\sum_c p(y = c) \log p(y = c)\right)\\ &+ \left(-\sum_c p(y = c|X_j < t) \log p(y = c|X_j < t)\right)\end{aligned}$$

since $\hat{\pi}$ is an **maximum likelihood estimate** for the conditional distribution

$$p(c|X_j < t).$$

Gini index

3. Gini Index

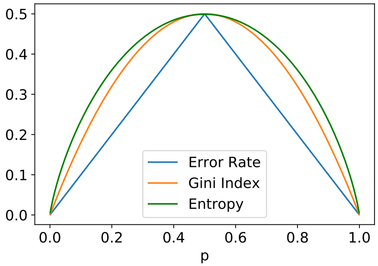
The **Gini index**

$$\sum_{c=1}^{C} \hat{\pi}_c (1 - \hat{\pi}_c) = 1 - \sum_{c=1}^{C} \hat{\pi}_c^2$$

is the **expected error rate**, where

- $\hat{\pi}_c$ is the probability a random entry in a leaf belongs to class c
- $(1 - \hat{\pi}_c)$ is the probability of being misclassified to class c

Comparison



Two-class Example

Suppose $c = 0$ or $c = 1$ and $p = \hat{\pi}_{c=1}$. Then,

Misclassification Rate: $1 - \max(p, 1 - p)$

Entropy: $-p \log p - (1 - p) \log(1 - p)$

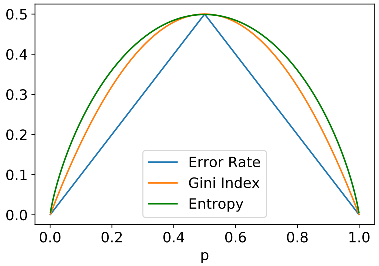
- range is 0 to 1.

Gini index: $p(1 - p)$

- range is 0 to ½.

[1]Entropy is scaled in the graph to pass through 0.5

Comparison



Two-class Example

Suppose $c = 0$ or $c = 1$ and $p = \hat{\pi}_{c=1}$. Then,

Misclassification Rate: $1 - \max(p, 1 - p)$

Entropy: $-p \log p - (1 - p) \log(1 - p)$

- range is 0 to 1.

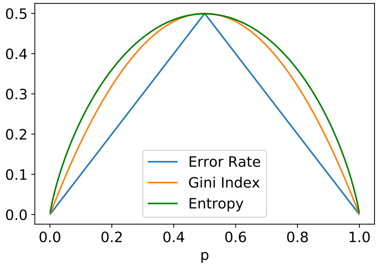
Gini index: $p(1 - p)$

- range is 0 to ½.

Gini and Entropy[1] are very similar. Both are more sensitive to changes in class probability than misclassification.

[1]Entropy is scaled in the graph to pass through 0.5

Comparison

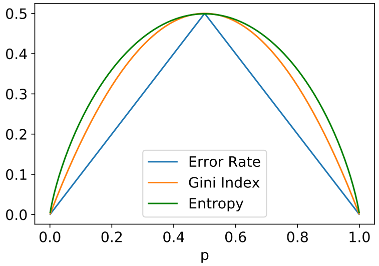


Example with Four Observations

Entropy: $-p \log p - (1-p)\log(1-p)$

Gini index: $p(1-p)$

Comparison



Example with Four Observations

Entropy: $-p \log p - (1-p) \log(1-p)$

Gini index: $p(1-p)$

PPPP: $p = {}^4/_4$

Entropy: $-\frac{4}{4} \log \frac{4}{4} - (1 - \frac{4}{4}) \log(1 - \frac{4}{4}) = 0$

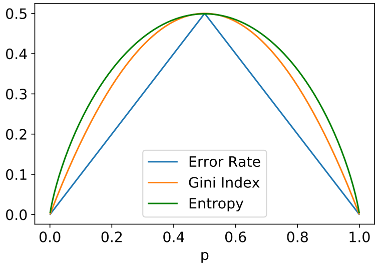
Gini: $\frac{4}{4}(1 - \frac{4}{4}) = 0$

PPNN: $p = {}^2/_4$

Entropy: $-\frac{1}{2} \log \frac{1}{2} - (1 - \frac{1}{2}) \log(1 - \frac{1}{2}) = 1.0$

Gini: $2 \cdot \frac{1}{2}(1 - \frac{1}{2}) = 1.0$ *# rescaled to [0,1]*

Comparison



Example with Four Observations

Entropy: $-p \log p - (1-p)\log(1-p)$

Gini index: $p(1-p)$

PPPP: $p = {}^{4}/_{4}$

Entropy: $-\frac{4}{4}\log\frac{4}{4} - (1-\frac{4}{4})\log(1-\frac{4}{4}) = 0$

Gini: $\frac{4}{4}(1-\frac{4}{4}) = 0$

PPPN: $p = {}^{3}/_{4}$

Entropy: $-\frac{3}{4}\log\frac{3}{4} - (1-\frac{3}{4})\log(1-\frac{3}{4}) \approx 0.81$

Gini: $2 \cdot \frac{3}{4}(1-\frac{3}{4}) = 0.75$ *# rescaled to [0,1]*

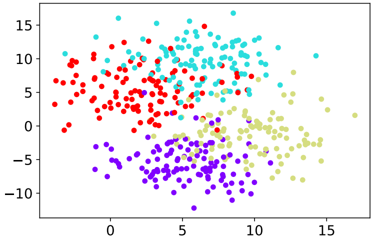
PPNN: $p = {}^{2}/_{4}$

Entropy: $-\frac{1}{2}\log\frac{1}{2} - (1-\frac{1}{2})\log(1-\frac{1}{2}) = 1.0$

Gini: $2 \cdot \frac{1}{2}(1-\frac{1}{2}) = 1.0$ *# rescaled to [0,1]*

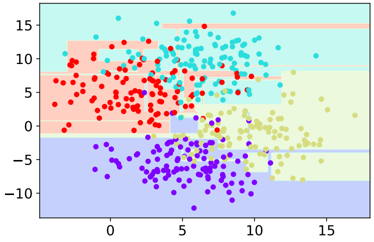
Overfitting

CART Models tend to Overfit



4 classes

Overfitting

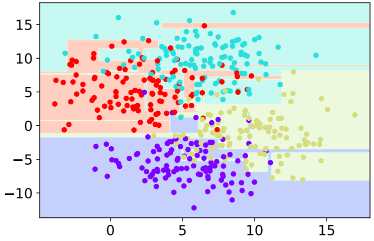


4 classes

CART Models tend to Overfit

This CART classification model has **low** training error but **high** CV error.

Overfitting

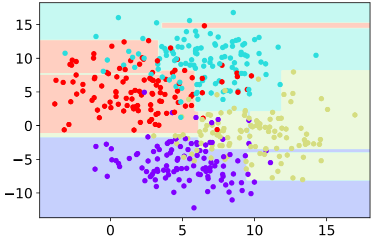


`max_depth` = 10

CART Models tend to Overfit

This CART classification model has **low** training error but **high** CV error.

Overfitting



max_depth = 5

CART Models tend to Overfit

By reducing the **maximum tree depth**, you can address overfitting.

For example, adjusting the hyperparamter **max_depth** changes the regions as follows.

» max_depth = 5

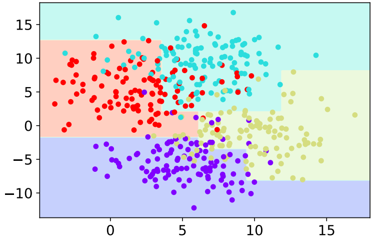
max_depth = 4

max_depth = 3

max_depth = 2

max_depth = 1

Overfitting



max_depth = 4

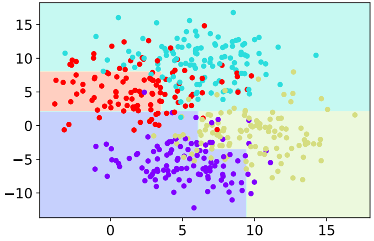
CART Models tend to Overfit

By reducing the **maximum tree depth**, you can address overfitting.

For example, adjusting the hyperparamter **max_depth** changes the regions as follows.

- max_depth = 5
- ≫ max_depth = 4
- max_depth = 3
- max_depth = 2
- max_depth = 1

Overfitting



max_depth = 3

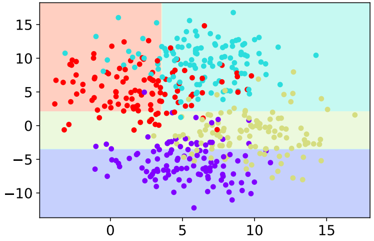
CART Models tend to Overfit

By reducing the **maximum tree depth**, you can address overfitting.

For example, adjusting the hyperparamter **max_depth** changes the regions as follows.

- max_depth = 5
- max_depth = 4
- ≫ max_depth = 3
- max_depth = 2
- max_depth = 1

Overfitting



`max_depth` = 2

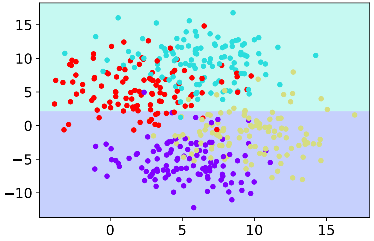
CART Models tend to Overfit

By reducing the **maximum tree depth**, you can address overfitting.

For example, adjusting the hyperparamter **max_depth** changes the regions as follows.

- `max_depth` = 5
- `max_depth` = 4
- `max_depth` = 3
- ≫ `max_depth` = 2
- `max_depth` = 1

Overfitting



max_depth = 1

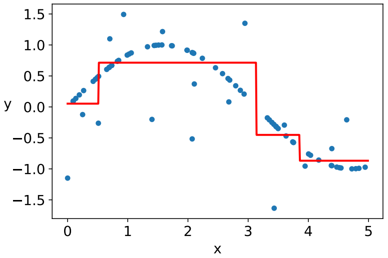
CART Models tend to Overfit

By reducing the **maximum tree depth**, you can address overfitting.

For example, adjusting the hyperparamter **max_depth** changes the regions as follows.

- max_depth = 5
- max_depth = 4
- max_depth = 3
- max_depth = 2
- ≫ max_depth = 1

Overfitting



max_depth = 2

CART Models tend to Overfit

The same applies to CART regression models.

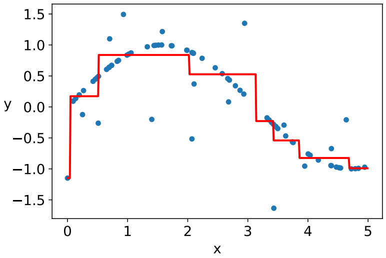
≫ max_depth = 2

max_depth = 3

max_depth = 5

max_depth = 10

Overfitting



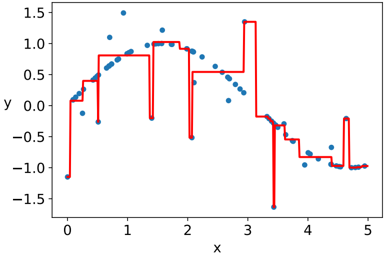
max_depth = 3

CART Models tend to Overfit

The same applies to CART regression models.

- max_depth = 2
- ≫ max_depth = 3
- max_depth = 5
- max_depth = 10

Overfitting



max_depth = 5

CART Models tend to Overfit

The same applies to CART regression models.

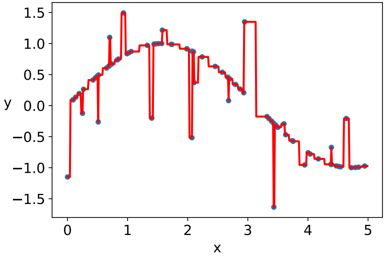
max_depth = 2

max_depth = 3

≫ max_depth = 5

max_depth = 10

Overfitting



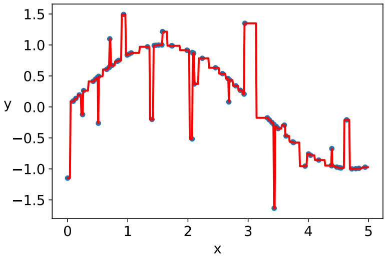
max_depth = 10

CART Models tend to Overfit

The same applies to CART regression models.

- max_depth = 2
- max_depth = 3
- max_depth = 5
- ≫ max_depth = 10

Overfitting



max_depth = 10

CART Models tend to Overfit

The same applies to CART regression models.

max_depth = 2

max_depth = 3

max_depth = 5

» max_depth = 10

Technical note: The standard approach is to grow a full tree, then **prune** the tree back to the maximum depth.

Advantages and Disadvantages

CART Pros

+ Easy to interpret
+ Easily handle discrete and continuous input
+ Insensitive to monotone transformations of input
+ Perform automatic variable selection
+ Scale reasonably well
+ Relatively robust to outliers
+ Can be modified to handle missing inputs by by looking for "backup" variables.

CART Cons

– Poor prediction accuracy
– Instability: small changes to input data can have large effects on structure of tree

Random Forests

Bagging

Bootstrap Aggregating

Trees are **high variance** estimators.

One technique for reducing variance of an estimate is to **average together** several estimates

Bagging

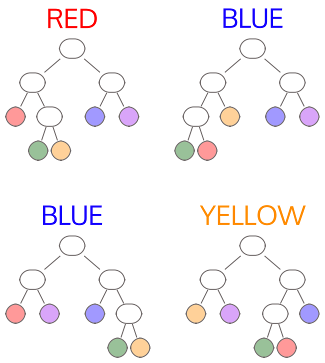
Bootstrap Aggregating

Suppose we train K different trees on subsets of data, randomly drawn with replacement, then calculate the **ensemble** h:

$$h(\boldsymbol{x}) = \sum_{t=1}^{K} \frac{1}{K} h_k(\boldsymbol{x})$$

where $h_k(\boldsymbol{x})$ is the k'th tree.

Bagging



Bootstrap Aggregating

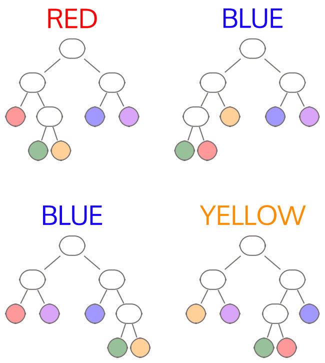
Suppose we train K different trees on subsets of data, randomly drawn with replacement, then calculate the **ensemble** h:

$$h(\boldsymbol{x}) = \sum_{t=1}^{K} \frac{1}{K} h_k(\boldsymbol{x})$$

where $h_k(\boldsymbol{x})$ is the k'th tree.

The result $h(\boldsymbol{x})$:
- *combines the results (regression)*, or
- *tallies votes for each class (classification).*

Bagging



$h(\boldsymbol{x}) =$ BLUE

Bootstrap Aggregating

Suppose we train K different trees on subsets of data, randomly drawn with replacement, then calculate the **ensemble** h:

$$h(\boldsymbol{x}) = \sum_{t=1}^{K} \frac{1}{K} h_k(\boldsymbol{x})$$

where $h_k(\boldsymbol{x})$ is the k'th tree.

The result $h(\boldsymbol{x})$:
- *combines the results (regression)*, or
- *tallies votes for each class (classification).*

This method is called **bagging**, which is short for 'bootstrap aggregating'.

Bagging

Bootstrap Aggregating for Classification

Bagging creates a **committee** of trees to vote on a class, and the predicted class is the majority winner.

The idea behind bagging is to average many noisy but approximately unbiased models.

Bagging

Bootstrap Aggregating for Classification

Bagging creates a **committee** of trees to vote on a class, and the predicted class is the majority winner.

The idea behind bagging is to average many noisy but approximately unbiased models.

However, one **problem** running the same algorithm on different samples of data is highly correlated predictors, which limits the amount of variance reduction.

Random Forests

De-correlated trees

Random forests:

- randomly sample a **subset of data** (like bagging), and
- randomly sample a **subset of input variables**.

random forests build a large collection of de-correlated trees and averages them.

Random Forests

Random Forests for Regression or Classification

```
1:  for $b = 1$ to $B$ do:
2:      Draw a bootstrap sample $Z$ of size $N$ from training data
3:      Grow a random forest tree $T_b$ to the bootstrapped data:
4:      repeat
5:          (i) Select $k$ variables at random from the $n$ variables
6:          (ii) Pick the best variable/split-thresholds among the $k$
7:          (iii) Split the node into two daughter nodes.
8:      until the minimum node size $n_{\min}$ is reached.
9:  end for
10: Output the ensemble of trees $\{T_1, T_2, \dots, T_B\}$.
```

To make a prediction given a new datapoint x:

Regression: $\frac{1}{B}\sum_{b=1}^{B} T_b(x)$

Classification: Let $\hat{\pi}_c(x)$ be the class prediction of the b'th random-forest tree. Then the random forest prediction, $\hat{\pi}_{rf}(x)$, is the **majority vote** of

$$\{\hat{\pi}_1(x), \hat{\pi}_2(x), \dots, \hat{\pi}_b(x), \dots, \hat{\pi}_B(x)\}$$

References

Hastie, T., R. Tibshirani, and J. Friedman (2009).
The Elements of Statistical Learning (2nd ed.).
Springer Series in Statistics. Springer.

Murphy, K. P. (2012).
Machine Learning: A Probabilistic Perspective.
Cambridge, MA: MIT Press.