Machine Learning

Kernel Methods and Decision Trees

Topics of previous lecture

- ✓ Ingredients of Machine Learning
- ✓ Classification Basics
- ✓ Basic Linear Classifier
- ✓ K-Nearest Neighbours Classifier
- ✓ Naive Bayes Classifier
- ✓ Linear and Quadratic Discriminant Analysis
- ✓ Support Vector Machines (SVM)

Topics of today's lecture

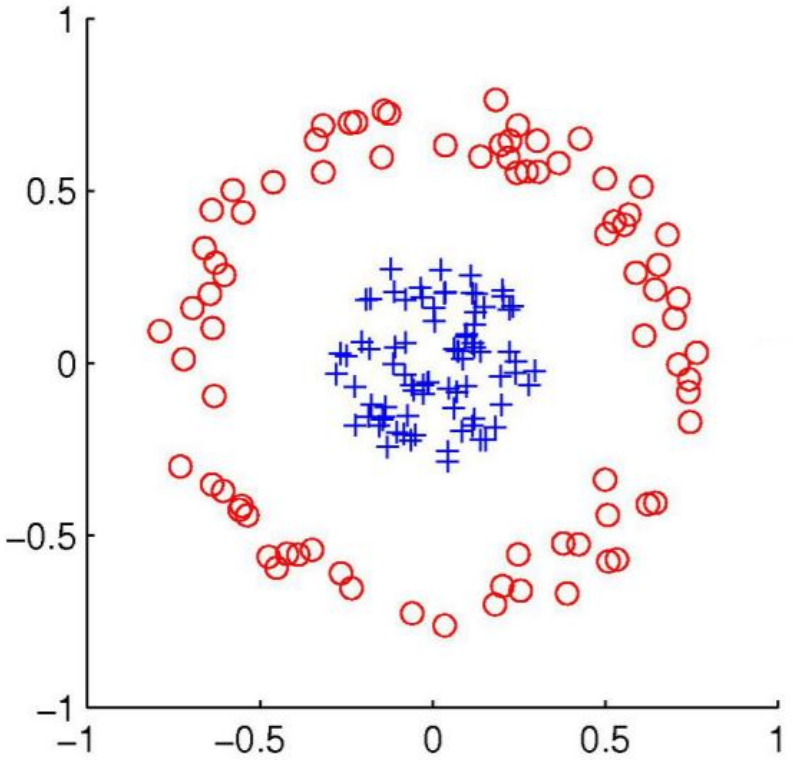
- Kernel methods
- No Free Lunch Theorem
- Decision trees
- Choosing the best split
- Impurity measures
- Decision Tree Learners
- Pruning
- Cross-Validation

Motivation for Kernels

- We know that KNN results in non-linear models (decision boundary is not a straight line / hyperplane)

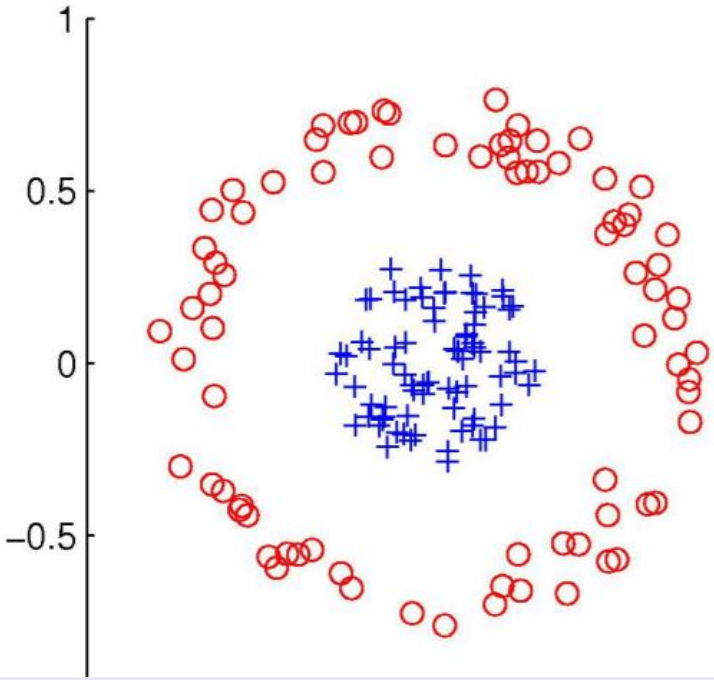
Motivation for Kernels

- We know that KNN results in non-linear models (decision boundary is not a straight line / hyperplane)
- For example, KNN works very nicely on these data:



Motivation for Kernels

- We know that KNN results in non-linear models (decision boundary is not a straight line / hyperplane)
- For example, KNN works very nicely on these data:



It turns out that linear methods can also learn this separation! How?

Non-linearity with linear models

- Linear models are linear in the features

Non-linearity with linear models

- Linear models are linear in the features
- Let us construct new features that are nonlinear in the original features

Non-linearity with linear models

- Linear models are linear in the features
- Let us construct new features that are nonlinear in the original features
- Linear models on the new features are nonlinear in the original features!

Non-linearity with linear models

- Linear models are linear in the features
- Let us construct new features that are nonlinear in the original features
- Linear models on the new features are nonlinear in the original features!
- This way we can fit non-linear models using linear model learning algorithms e.g.

Non-linearity with linear models

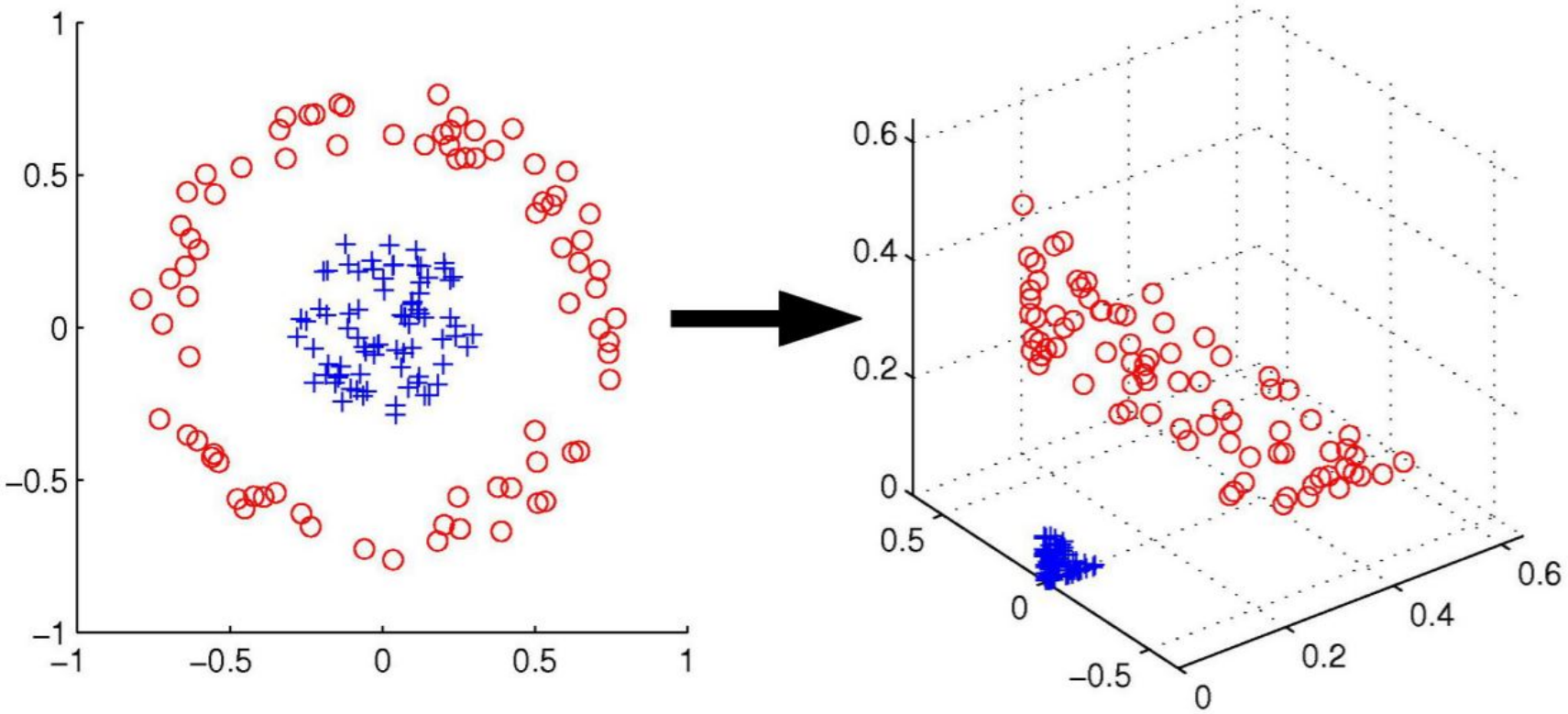
- Linear models are linear in the features
- Let us construct new features that are nonlinear in the original features
- Linear models on the new features are nonlinear in the original features!
- This way we can fit non-linear models using linear model learning algorithms e.g.
  - if the hidden dependency is $y = x^3 + 3x^2 - x + 7$
  - Then if we introduce features x, x^2, x^3 then linear methods can learn this functional dependency

Non-linearity with linear models

Let's construct new features to make data linearly separable:

$$\phi : \mathbb{R}^2 \to \mathbb{R}^3$$

$$(x_1, x_2) \to (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$



Non-linearity with linear models

- Fitting:

Non-linearity with linear models

- Fitting:
  - Transform the training data through $\phi(\mathbf{x})$: $\mathbf{x} \rightarrow \phi(\mathbf{x})$

Non-linearity with linear models

- Fitting:
  - Transform the training data through $\phi(\mathbf{x})$: $\mathbf{x} \rightarrow \phi(\mathbf{x})$
  - Learn a linear model

Non-linearity with linear models

- Fitting:
  - Transform the training data through $\phi(\mathbf{x})$: $\mathbf{x} \rightarrow \phi(\mathbf{x})$
  - Learn a linear model
- Predicting:

Non-linearity with linear models

- Fitting:
  - Transform the training data through $\phi(\mathbf{x})$: $\mathbf{x} \rightarrow \phi(\mathbf{x})$
  - Learn a linear model
- Predicting:
  - Transform the test data through $\phi(\mathbf{x})$: $\mathbf{x} \rightarrow \phi(\mathbf{x})$

Non-linearity with linear models

- Fitting:
  - Transform the training data through $\phi(\mathbf{x})$: $\mathbf{x} \rightarrow \phi(\mathbf{x})$
  - Learn a linear model
- Predicting:
  - Transform the test data through $\phi(\mathbf{x})$: $\mathbf{x} \rightarrow \phi(\mathbf{x})$
  - Apply the fitted linear model

Non-linearity with linear models

- Fitting:
  - Transform the training data through $\phi(\mathbf{x})$: $\mathbf{x} \rightarrow \phi(\mathbf{x})$
  - Learn a linear model
- Predicting:
  - Transform the test data through $\phi(\mathbf{x})$: $\mathbf{x} \rightarrow \phi(\mathbf{x})$
  - Apply the fitted linear model
- This can be very slow, if there are a lot of constructed features. Can we make it faster?

Non-linearity with linear models

- Fitting:
  - Transform the training data through $\phi(\mathbf{x})$: $\mathbf{x} \rightarrow \phi(\mathbf{x})$
  - Learn a linear model
- Predicting:
  - Transform the test data through $\phi(\mathbf{x})$: $\mathbf{x} \rightarrow \phi(\mathbf{x})$
  - Apply the fitted linear model
- This can be very slow, if there are a lot of constructed features. Can we make it faster?
- Let's see how $\phi(\mathbf{x})$ is used in SVM

SVM on constructed features

- Fitting (dual form soft-margin):
  $\mathbf{x}_i \to \phi(\mathbf{x}_i)$ for $i = 1, \ldots, n$

$$\alpha_1^*, \ldots, \alpha_n^* = \underset{\alpha_1,\ldots,\alpha_n}{\operatorname{argmax}} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j) + \sum_{i=1}^{n}\alpha_i$$

$$\text{subject to } 0 \le \alpha_i \le C, \quad i = 1, \ldots, n \text{ and } \sum_{i=1}^{n}\alpha_i y_i = 0$$

$$\mathbf{w}^* = \sum_{i=1}^{n}\alpha_i^* y_i \phi(\mathbf{x}_i) \qquad t^* = \mathbf{w}^*\cdot\phi(\mathbf{x}_j) - y_j = \sum_{i=1}^{n}\alpha_i^* y_i \phi(\mathbf{x}_i)\phi(\mathbf{x}_j) - y_j$$

SVM on constructed features

- Fitting (dual form soft-margin):
  $\mathbf{x}_i \rightarrow \phi(\mathbf{x}_i)$ for $i = 1, \ldots, n$

$$\alpha_1^*, \ldots, \alpha_n^* = \underset{\alpha_1, \ldots, \alpha_n}{\operatorname{argmax}} -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) + \sum_{i=1}^{n} \alpha_i$$

$$\text{subject to } 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, n \text{ and } \sum_{i=1}^{n} \alpha_i y_i = 0$$

$$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \phi(\mathbf{x}_i) \qquad t^* = \mathbf{w}^* \cdot \phi(\mathbf{x}_j) - y_j = \sum_{i=1}^{n} \alpha_i^* y_i \phi(\mathbf{x}_i) \phi(\mathbf{x}_j) - y_j$$

- Prediction:
  $\mathbf{x} \rightarrow \phi(\mathbf{x})$

$$\hat{y} = sign(\mathbf{w}^* \cdot \phi(\mathbf{x}_i) - t^*) = sign(\sum_{i=1}^{n} \alpha_i^* y_i \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) - t^*))$$

Everything through dot products

- In SVM fitting and prediction all uses of instances can be made through dot products

Everything through dot products

- In SVM fitting and prediction all uses of instances can be made through dot products
- Our example transformation:

$$\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

$$(x_1, x_2) \rightarrow (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

Everything through dot products

- In SVM fitting and prediction all uses of instances can be made through dot products
- Our example transformation:

$$\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

$$(x_1, x_2) \rightarrow (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

- Dot product:

$$\phi(\mathbf{x}) \cdot \phi(\mathbf{x}') = \phi(x_1, x_2) \cdot \phi(x_1', x_2') =$$

Everything through dot products

- In SVM fitting and prediction all uses of instances can be made through dot products
- Our example transformation:

$$\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

$$(x_1, x_2) \rightarrow (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

- Dot product:

$$\begin{aligned}\phi(\mathbf{x}) \cdot \phi(\mathbf{x}') &= \phi(x_1, x_2) \cdot \phi(x_1', x_2') = \\ &= (x_1^2, \sqrt{2}x_1x_2, x_2^2) \cdot ({x_1'}^2, \sqrt{2}x_1'x_2', {x_2'}^2) =\end{aligned}$$

Everything through dot products

- In SVM fitting and prediction all uses of instances can be made through dot products
- Our example transformation:

$$\phi : \mathbb{R}^2 \to \mathbb{R}^3$$

$$(x_1, x_2) \to (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

- Dot product:

$$\begin{aligned}\phi(\mathbf{x}) \cdot \phi(\mathbf{x}') &= \phi(x_1, x_2) \cdot \phi(x_1', x_2') = \\ &= (x_1^2, \sqrt{2}x_1x_2, x_2^2) \cdot ({x_1'}^2, \sqrt{2}x_1'x_2', {x_2'}^2) = \\ &= x_1^2{x_1'}^2 + \sqrt{2}x_1x_2\sqrt{2}x_1'x_2' + x_2^2{x_2'}^2 =\end{aligned}$$

Everything through dot products

- In SVM fitting and prediction all uses of instances can be made through dot products
- Our example transformation:

$$\phi : \mathbb{R}^2 \to \mathbb{R}^3$$

$$(x_1, x_2) \to (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

- Dot product:

$$\begin{aligned}
\phi(\mathbf{x}) \cdot \phi(\mathbf{x}') &= \phi(x_1, x_2) \cdot \phi(x_1', x_2') = \\
&= (x_1^2, \sqrt{2}x_1x_2, x_2^2) \cdot ({x_1'}^2, \sqrt{2}x_1'x_2', {x_2'}^2) = \\
&= x_1^2{x_1'}^2 + \sqrt{2}x_1x_2\sqrt{2}x_1'x_2' + x_2^2{x_2'}^2 = \\
&= (x_1x_1')^2 + 2(x_1x_1')(x_2x_2') + (x_2x_2')^2 =
\end{aligned}$$

Everything through dot products

- In SVM fitting and prediction all uses of instances can be made through dot products
- Our example transformation:

$$\phi : \mathbb{R}^2 \to \mathbb{R}^3$$

$$(x_1, x_2) \to (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

- Dot product:

$$\begin{aligned}
\phi(\mathbf{x}) \cdot \phi(\mathbf{x}') &= \phi(x_1, x_2) \cdot \phi(x_1', x_2') = \\
&= (x_1^2, \sqrt{2}x_1x_2, x_2^2) \cdot ({x_1'}^2, \sqrt{2}x_1'x_2', {x_2'}^2) = \\
&= x_1^2{x_1'}^2 + \sqrt{2}x_1x_2\sqrt{2}x_1'x_2' + x_2^2{x_2'}^2 = \\
&= (x_1x_1')^2 + 2(x_1x_1')(x_2x_2') + (x_2x_2')^2 = \\
&= (x_1x_1' + x_2x_2')^2 = (\mathbf{x} \cdot \mathbf{x}')^2
\end{aligned}$$

Kernel trick

- We can now use the kernel trick!

Kernel trick

- We can now use the kernel trick!
- Instead of

Kernel trick

- We can now use the kernel trick!
- Instead of
  - transforming the instances with $\mathbf{x} \rightarrow \phi(\mathbf{x})$

Kernel trick

- We can now use the kernel trick!
- Instead of
  - transforming the instances with $\mathbf{x} \rightarrow \phi(\mathbf{x})$
  - then fitting and predicting on the constructed higher-dimensional instances $\phi(\mathbf{x})$

Kernel trick

- We can now use the kernel trick!
- Instead of
  - transforming the instances with $\mathbf{x} \rightarrow \phi(\mathbf{x})$
  - then fitting and predicting on the constructed higher-dimensional instances $\phi(\mathbf{x})$
- We can now do the kernel trick:

Kernel trick

- We can now use the kernel trick!
- Instead of
  - transforming the instances with $\mathbf{x} \rightarrow \phi(\mathbf{x})$
  - then fitting and predicting on the constructed higher-dimensional instances $\phi(\mathbf{x})$
- We can now do the kernel trick:
  - Work with original instances

Kernel trick

- We can now use the kernel trick!
- Instead of
  - transforming the instances with $\mathbf{x} \rightarrow \phi(\mathbf{x})$
  - then fitting and predicting on the constructed higher-dimensional instances $\phi(\mathbf{x})$
- We can now do the kernel trick:
  - Work with original instances
  - Use a modified method of calculating dot products $\mathbf{x} \cdot \mathbf{x}' \rightarrow \kappa(\mathbf{x}, \mathbf{x}')$

Kernel trick

- We can now use the kernel trick!
- Instead of
  - transforming the instances with $\mathbf{x} \rightarrow \phi(\mathbf{x})$
  - then fitting and predicting on the constructed higher-dimensional instances $\phi(\mathbf{x})$
- We can now do the kernel trick:
  - Work with original instances
  - Use a modified method of calculating dot products $\mathbf{x} \cdot \mathbf{x}' \rightarrow \kappa(\mathbf{x}, \mathbf{x}')$
  - In our example the **kernel** is $\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}')^2$

Kernel SVM

- Fitting (soft-margin kernel SVM):

$$\alpha_1^*, \ldots, \alpha_n^* = \underset{\alpha_1,\ldots,\alpha_n}{\operatorname{argmax}} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{n}\alpha_i$$

$$\text{subject to } 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, n \text{ and } \sum_{i=1}^{n}\alpha_i y_i = 0$$

$$t^* = \sum_{i=1}^{n}\alpha_i^* y_i \kappa(\mathbf{x}_i, \mathbf{x}_j) - y_j,$$

where $\mathbf{x}_j$ is a support vector of class y_j.

Kernel SVM

- Fitting (soft-margin kernel SVM):

$$\alpha_1^*, \ldots, \alpha_n^* = \underset{\alpha_1,\ldots,\alpha_n}{\operatorname{argmax}} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{n}\alpha_i$$

$$\text{subject to } 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, n \text{ and } \sum_{i=1}^{n}\alpha_i y_i = 0$$

$$t^* = \sum_{i=1}^{n}\alpha_i^* y_i \kappa(\mathbf{x}_i, \mathbf{x}_j) - y_j,$$

where $\mathbf{x}_j$ is a support vector of class y_j.

- Prediction:

$$\hat{y} = sign\left(\sum_{i=1}^{n}\alpha_i^* y_i \kappa(\mathbf{x}_i, \mathbf{x}) - t^*\right)$$

When does the kernel trick work?

- Can we choose any function for $\kappa(\mathbf{x}, \mathbf{x}')$?

When does the kernel trick work?

- Can we choose any function for $\kappa(\mathbf{x}, \mathbf{x}')$?
- No! We can only choose kernels for which there exists some transformation ϕ such that:

$$\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x}) \cdot \phi(\mathbf{x}')$$

Otherwise the mathematics of SVM breaks down

When does the kernel trick work?

- Can we choose any function for $\kappa(\mathbf{x}, \mathbf{x}')$?
- No! We can only choose kernels for which there exists some transformation ϕ such that:

$$\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x}) \cdot \phi(\mathbf{x}')$$

  Otherwise the mathematics of SVM breaks down
- Mercer's theorem:
  Such transformation ϕ exists if and only if $\kappa(\mathbf{x}, \mathbf{x}')$ is a positive semi-definite function

Positive semi-definiteness

Definition

Let $\kappa : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, then function $\kappa(\mathbf{x}, \mathbf{x}')$ is positive semi-definite, if for any $\mathbf{x}_1, \ldots, \mathbf{x}_n$ the following matrix (**kernel matrix**) is positive semi-definite:

$$\begin{pmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \kappa(\mathbf{x}_1, \mathbf{x}_2) & \ldots & \kappa(\mathbf{x}_1, \mathbf{x}_n) \\ \kappa(\mathbf{x}_2, \mathbf{x}_1) & \kappa(\mathbf{x}_2, \mathbf{x}_2) & \ldots & \kappa(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_n, \mathbf{x}_1) & \kappa(\mathbf{x}_n, \mathbf{x}_2) & \ldots & \kappa(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix}$$

Definition

A matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ is positive semi-definite, if for any vector $\mathbf{x} \in \mathbb{R}^n$

$$\mathbf{x}^T \mathbf{M} \mathbf{x} \geq 0$$

Checking these criteria might be non-trivial, often new kernels are constructed by combining existing ones

Well-known kernels

- Linear kernel (original space unchanged):

$$\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}'$$

Well-known kernels

- Linear kernel (original space unchanged):

$$\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}'$$

- Polynomial kernel (for any $\gamma, r \in \mathbb{R}$ and $d \in \mathbb{N}$):

$$\kappa(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x} \cdot \mathbf{x}' + r)^d$$

Well-known kernels

- Linear kernel (original space unchanged):

$$\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}'$$

- Polynomial kernel (for any $\gamma, r \in \mathbb{R}$ and $d \in \mathbb{N}$):

$$\kappa(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x} \cdot \mathbf{x}' + r)^d$$

- Gaussian kernel (also known as Radial Basis Function (RBF)) for any $\sigma > 0$ ($\gamma > 0$):

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(- \frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma} \right) = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$$

Polynomial kernel

- For any $\gamma, r \in \mathbb{R}$ and $d \in \mathbb{N}$:

$$\kappa(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x} \cdot \mathbf{x}' + r)^d$$

Polynomial kernel

- For any $\gamma, r \in \mathbb{R}$ and $d \in \mathbb{N}$:

$$\kappa(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x} \cdot \mathbf{x}' + r)^d$$

- For example, take $\gamma = r = 1$ and $d = 2$ with 2 features:

$$\kappa(\mathbf{x}, \mathbf{x}') = (x_1 x_1' + x_2 x_2' + 1)^2 =$$

Polynomial kernel

- For any $\gamma, r \in \mathbb{R}$ and $d \in \mathbb{N}$:

$$\kappa(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x} \cdot \mathbf{x}' + r)^d$$

- For example, take $\gamma = r = 1$ and $d = 2$ with 2 features:

$$\kappa(\mathbf{x}, \mathbf{x}') = (x_1 x_1' + x_2 x_2' + 1)^2 =$$

$$= x_1^2 {x_1'}^2 + x_2^2 {x_2'}^2 + 1 \cdot 1 + \sqrt{2} x_1 x_2 \sqrt{2} x_1' x_2' + \sqrt{2} x_1 \sqrt{2} x_1' + \sqrt{2} x_2 \sqrt{2} x_2' =$$

Polynomial kernel

- For any $\gamma, r \in \mathbb{R}$ and $d \in \mathbb{N}$:

$$\kappa(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x} \cdot \mathbf{x}' + r)^d$$

- For example, take $\gamma = r = 1$ and $d = 2$ with 2 features:

$$\kappa(\mathbf{x}, \mathbf{x}') = (x_1 x_1' + x_2 x_2' + 1)^2 =$$

$$= x_1^2 {x_1'}^2 + x_2^2 {x_2'}^2 + 1 \cdot 1 + \sqrt{2} x_1 x_2 \sqrt{2} x_1' x_2' + \sqrt{2} x_1 \sqrt{2} x_1' + \sqrt{2} x_2 \sqrt{2} x_2' =$$

$$= (x_1^2, x_2^2, 1, \sqrt{2} x_1 x_2, \sqrt{2} x_1, \sqrt{2} x_2) \cdot ({x_1'}^2, {x_2'}^2, 1, \sqrt{2} x_1' x_2', \sqrt{2} x_1', \sqrt{2} x_2') =$$

Polynomial kernel

- For any $\gamma, r \in \mathbb{R}$ and $d \in \mathbb{N}$:

$$\kappa(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x} \cdot \mathbf{x}' + r)^d$$

- For example, take $\gamma = r = 1$ and $d = 2$ with 2 features:

$$\kappa(\mathbf{x}, \mathbf{x}') = (x_1 x_1' + x_2 x_2' + 1)^2 =$$

$$= x_1^2 {x_1'}^2 + x_2^2 {x_2'}^2 + 1 \cdot 1 + \sqrt{2} x_1 x_2 \sqrt{2} x_1' x_2' + \sqrt{2} x_1 \sqrt{2} x_1' + \sqrt{2} x_2 \sqrt{2} x_2' =$$

$$= (x_1^2, x_2^2, 1, \sqrt{2} x_1 x_2, \sqrt{2} x_1, \sqrt{2} x_2) \cdot ({x_1'}^2, {x_2'}^2, 1, \sqrt{2} x_1' x_2', \sqrt{2} x_1', \sqrt{2} x_2') =$$

$$= \phi(\mathbf{x}) \cdot \phi(\mathbf{x}')$$

Polynomial kernel

- For any $\gamma, r \in \mathbb{R}$ and $d \in \mathbb{N}$:

$$\kappa(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x} \cdot \mathbf{x}' + r)^d$$

- For example, take $\gamma = r = 1$ and $d = 2$ with 2 features:

$$\kappa(\mathbf{x}, \mathbf{x}') = (x_1 x_1' + x_2 x_2' + 1)^2 =$$

$$= x_1^2 {x_1'}^2 + x_2^2 {x_2'}^2 + 1 \cdot 1 + \sqrt{2} x_1 x_2 \sqrt{2} x_1' x_2' + \sqrt{2} x_1 \sqrt{2} x_1' + \sqrt{2} x_2 \sqrt{2} x_2' =$$

$$= (x_1^2, x_2^2, 1, \sqrt{2} x_1 x_2, \sqrt{2} x_1, \sqrt{2} x_2) \cdot ({x_1'}^2, {x_2'}^2, 1, \sqrt{2} x_1' x_2', \sqrt{2} x_1', \sqrt{2} x_2') =$$

$$= \phi(\mathbf{x}) \cdot \phi(\mathbf{x}')$$

- Constructs all polynomials of features x_1, x_2 up to $d = 2$ degree

Gaussian kernel

In case of Gaussian kernel the constructed feature space is infinite-dimensional !!! E.g. for $\sigma = 1$ and $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^k$

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(- \frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2} \right) =$$

Gaussian kernel

In case of Gaussian kernel the constructed feature space is infinite-dimensional !!! E.g. for $\sigma = 1$ and $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^k$

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(- \frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2} \right) = e^{-\frac{\|\mathbf{x}\|^2}{2}} e^{-\frac{\|\mathbf{x}'\|^2}{2}} e^{\mathbf{x} \cdot \mathbf{x}'} =$$

Gaussian kernel

In case of Gaussian kernel the constructed feature space is infinite-dimensional !!! E.g. for $\sigma = 1$ and $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^k$

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2}\right) = e^{-\frac{\|\mathbf{x}\|^2}{2}} e^{-\frac{\|\mathbf{x}'\|^2}{2}} e^{\mathbf{x}\cdot\mathbf{x}'} =$$

$$= e^{-\frac{\|\mathbf{x}\|^2}{2}} e^{-\frac{\|\mathbf{x}'\|^2}{2}} \sum_{j=0}^{\infty} \frac{(\mathbf{x}\cdot\mathbf{x}')^j}{j!} =$$

Gaussian kernel

In case of Gaussian kernel the constructed feature space is infinite-dimensional !!! E.g. for $\sigma = 1$ and $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^k$

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(- \frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2} \right) = e^{-\frac{\|\mathbf{x}\|^2}{2}} e^{-\frac{\|\mathbf{x}'\|^2}{2}} e^{\mathbf{x} \cdot \mathbf{x}'} =$$

$$= e^{-\frac{\|\mathbf{x}\|^2}{2}} e^{-\frac{\|\mathbf{x}'\|^2}{2}} \sum_{j=0}^{\infty} \frac{(\mathbf{x} \cdot \mathbf{x}')^j}{j!} =$$

using this formula $(x_1 + \ldots + x_k)^d = \sum_{n_i \geq 0, \sum_i n_i = d} \frac{d!}{n_1! \cdots n_k!} x_1^{n_1} \cdots x_k^{n_k}$

Gaussian kernel

In case of Gaussian kernel the constructed feature space is infinite-dimensional !!! E.g. for $\sigma = 1$ and $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^k$

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2}\right) = e^{-\frac{\|\mathbf{x}\|^2}{2}} e^{-\frac{\|\mathbf{x}'\|^2}{2}} e^{\mathbf{x}\cdot\mathbf{x}'} =$$

$$= e^{-\frac{\|\mathbf{x}\|^2}{2}} e^{-\frac{\|\mathbf{x}'\|^2}{2}} \sum_{j=0}^{\infty} \frac{(\mathbf{x}\cdot\mathbf{x}')^j}{j!} =$$

using this formula $(x_1 + \ldots + x_k)^d = \sum_{n_i \geq 0, \sum_i n_i = d} \frac{d!}{n_1! \cdots n_k!} x_1^{n_1} \cdots x_k^{n_k}$

$$= e^{-\frac{\|\mathbf{x}\|^2}{2}} e^{-\frac{\|\mathbf{x}'\|^2}{2}} \sum_{j=0}^{\infty} \frac{1}{j!} \sum_{\sum_i n_i = j} \frac{j!(x_1 x_1')^{n_1} \cdots (x_k x_k')^{n_k}}{n_1! \cdots n_k!} =$$

Gaussian kernel

In case of Gaussian kernel the constructed feature space is infinite-dimensional !!! E.g. for $\sigma = 1$ and $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^k$

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2}\right) = e^{-\frac{\|\mathbf{x}\|^2}{2}} e^{-\frac{\|\mathbf{x}'\|^2}{2}} e^{\mathbf{x}\cdot\mathbf{x}'} =$$

$$= e^{-\frac{\|\mathbf{x}\|^2}{2}} e^{-\frac{\|\mathbf{x}'\|^2}{2}} \sum_{j=0}^{\infty} \frac{(\mathbf{x}\cdot\mathbf{x}')^j}{j!} =$$

using this formula $(x_1+\ldots+x_k)^d = \sum_{n_i \geq 0, \sum_i n_i = d} \frac{d!}{n_1! \cdots n_k!} x_1^{n_1} \cdots x_k^{n_k}$

$$= e^{-\frac{\|\mathbf{x}\|^2}{2}} e^{-\frac{\|\mathbf{x}'\|^2}{2}} \sum_{j=0}^{\infty} \frac{1}{j!} \sum_{\sum_i n_i = j} \frac{j!(x_1 x_1')^{n_1} \cdots (x_k x_k')^{n_k}}{n_1! \cdots n_k!} =$$

$$= \sum_{j=0}^{\infty} \sum_{\sum_i n_i = j} e^{-\frac{\|\mathbf{x}\|^2}{2}} \frac{x_1^{n_1} \cdots x_k^{n_k}}{\sqrt{n_1! \cdots n_k!}} e^{-\frac{\|\mathbf{x}'\|^2}{2}} \frac{{x_1'}^{n_1} \cdots {x_k'}^{n_k}}{\sqrt{n_1! \cdots n_k!}}$$

Gaussian kernel

In case of Gaussian kernel the constructed feature space is infinite-dimensional !!! E.g. for $\sigma = 1$ and $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^k$

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2}\right) = e^{-\frac{\|\mathbf{x}\|^2}{2}} e^{-\frac{\|\mathbf{x}'\|^2}{2}} e^{\mathbf{x}\cdot\mathbf{x}'} =$$

$$= e^{-\frac{\|\mathbf{x}\|^2}{2}} e^{-\frac{\|\mathbf{x}'\|^2}{2}} \sum_{j=0}^{\infty} \frac{(\mathbf{x}\cdot\mathbf{x}')^j}{j!} =$$

using this formula $(x_1+\ldots+x_k)^d = \sum_{n_i \geq 0, \sum_i n_i = d} \frac{d!}{n_1! \cdots n_k!} x_1^{n_1} \cdots x_k^{n_k}$

$$= e^{-\frac{\|\mathbf{x}\|^2}{2}} e^{-\frac{\|\mathbf{x}'\|^2}{2}} \sum_{j=0}^{\infty} \frac{1}{j!} \sum_{\sum_i n_i = j} \frac{j!(x_1 x_1')^{n_1} \cdots (x_k x_k')^{n_k}}{n_1! \cdots n_k!} =$$

$$= \sum_{j=0}^{\infty} \sum_{\sum_i n_i = j} e^{-\frac{\|\mathbf{x}\|^2}{2}} \frac{x_1^{n_1} \cdots x_k^{n_k}}{\sqrt{n_1! \cdots n_k!}} e^{-\frac{\|\mathbf{x}'\|^2}{2}} \frac{{x_1'}^{n_1} \cdots {x_k'}^{n_k}}{\sqrt{n_1! \cdots n_k!}}$$

Here the feature map is the following: $\phi(\mathbf{x}) = \left(e^{-\frac{\|\mathbf{x}\|^2}{2}} \frac{x_1^{n_1} \cdots x_k^{n_k}}{\sqrt{n_1! \cdots n_k!}}\right)_{\sum_{i=1}^k n_i = j}$

Construction of kernels

If κ_1, κ_2 are kernels, $a > 0$ and $f : \mathbb{X} \to \mathbb{R}$ then the following $\kappa(\mathbf{x}, \mathbf{x}')$ are kernels as well:

$$\kappa(\mathbf{x}, \mathbf{x}') = a\kappa_1(\mathbf{x}, \mathbf{x}')$$

$$\kappa(\mathbf{x}, \mathbf{x}') = \kappa_1(\mathbf{x}, \mathbf{x}')\kappa_2(\mathbf{x}, \mathbf{x}')$$

$$\kappa(\mathbf{x}, \mathbf{x}') = \kappa_1(\mathbf{x}, \mathbf{x}') + \kappa_2(\mathbf{x}, \mathbf{x}')$$

$$\kappa(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})f(\mathbf{x}')$$

Fundamental issue in ML

- Learning theory claims that a machine learning algorithm can generalize well from a finite training set of examples
- Inductive reasoning (inferring general rules) from a limited set of examples, is not logically valid
- Machine learning promises to find rules that are **probably** correct about **most** members of the set they concern

Fundamental issue in ML

- No training instance has exactly the same vector of feature values as the test instance

Fundamental issue in ML

- No training instance has exactly the same vector of feature values as the test instance
- Different methods approach this issue differently:
  - Naïve Bayes assumes independence of features
  - Distance-based and kernel methods assume that similar instances have similar label values

Fundamental issue in ML

- No training instance has exactly the same vector of feature values as the test instance
- Different methods approach this issue differently:
  - Naïve Bayes assumes independence of features
  - Distance-based and kernel methods assume that similar instances have similar label values
- Which is the best approach?

No Free Lunch Theorem

- One of the fundamental theorems on learning

No Free Lunch Theorem

- One of the fundamental theorems on learning

No Free Lunch (NFL) theorem

Averaged over all possible data-generating distributions, every classfication algorithm has the same error rate when classifying previously unobserved points.

No Free Lunch Theorem

- One of the fundamental theorems on learning

No Free Lunch (NFL) theorem

Averaged over all possible data-generating distributions, every classfication algorithm has the same error rate when classifying previously unobserved points.

- Intuition behind NFL:
  - No learning algorithm works best for every problem

No Free Lunch Theorem

- One of the fundamental theorems on learning

No Free Lunch (NFL) theorem

Averaged over all possible data-generating distributions, every classfication algorithm has the same error rate when classifying previously unobserved points.

- Intuition behind NFL:
  - No learning algorithm works best for every problem
  - If training data covers a very small proportion of the instance space, then on the rest of the instance space the ground truth might be anything, unless we make additional assumptions

Example

Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Label
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0
0	1	1	1	0	???

Example

Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Label
0	1				1
0	1				0
0	1				0
0	1				1
0	0				0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0
0	1	1	1	0	???

If the true function from features to labels was fixed by coin tossing then we cannot predict much better than random

Example

Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Label
0	1	0	0	0	1
0	1	0	0	0	0
0	1	0	0	0	0
0	1	0	0	0	1
0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	0	0	1
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
1	0	0	0	0	0
1	0	1	1	0	0
0	1	1	1	0	???

If the true function from features to labels was fixed by coin tossing then we cannot predict much better than random

We can predict a bit better than random, if a small proportion of test instances also belong to the training set

More about NFL

- For any learning algorithm there exist "bad cases", that is datasets where the algorithm performs as bad or worse than random guessing

More about NFL

- For any learning algorithm there exist "bad cases", that is datasets where the algorithm performs as bad or worse than random guessing
- NFL for optimization:
  – If an algorithm performs better than random guessing on some class of problems then it will perform worse than random guessing on the remaining problems.

More about NFL

- For any learning algorithm there exist "bad cases", that is datasets where the algorithm performs as bad or worse than random guessing
- NFL for optimization:
  – If an algorithm performs better than random guessing on some class of problems then it will perform worse than random guessing on the remaining problems.
- The goal of ML is not to find a universal learning algorithm

More about NFL

- For any learning algorithm there exist "bad cases", that is datasets where the algorithm performs as bad or worse than random guessing
- NFL for optimization:
  – If an algorithm performs better than random guessing on some class of problems then it will perform worse than random guessing on the remaining problems.
- The goal of ML is not to find a universal learning algorithm
- The goal is to understand what kinds of distributions are relevant to the "real world" and what kinds of ML algorithms perform well on data drawn from those data-generating distributions.

How to overcome NFL?

- Make assumptions about the domain!

How to overcome NFL?

- Make assumptions about the domain!
- We already know that different models make different assumptions

How to overcome NFL?

- Make assumptions about the domain!
- We already know that different models make different assumptions
- What do decision trees assume?

How to overcome NFL?

- Make assumptions about the domain!
- We already know that different models make different assumptions
- What do decision trees assume?
- Decision trees assume that the instance space can be split into **segments** such that all (or at least majority of) instances in the segment belong to the same class

Lenses dataset

Presbyopic	Young	Spectacle prescription	Astigmatic	Tear production rate	Can use contact lenses
No	Yes	Myope	No	Normal	Yes
No	Yes	Myope	Yes	Reduced	No
No	Yes	Hypermetrope	No	Reduced	No
No	Yes	Hypermetrope	No	Normal	Yes
No	No	Myope	No	Reduced	No
No	No	Myope	No	Normal	Yes
No	No	Myope	Yes	Normal	Yes
No	No	Hypermetrope	No	Reduced	No
No	No	Hypermetrope	Yes	Reduced	No
No	No	Hypermetrope	Yes	Normal	No
Yes	No	Myope	No	Normal	No
Yes	No	Hypermetrope	Yes	Normal	No
No	Yes	Myope	No	Reduced	???
No	Yes	Myope	Yes	Normal	???
No	Yes	Hypermetrope	Yes	Reduced	???
No	Yes	Hypermetrope	Yes	Normal	???
No	No	Myope	Yes	Reduced	???
No	No	Hypermetrope	No	Normal	???
Yes	No	Myope	No	Reduced	???
Yes	No	Myope	Yes	Reduced	???
Yes	No	Myope	Yes	Normal	???
Yes	No	Hypermetrope	No	Reduced	???
Yes	No	Hypermetrope	No	Normal	???
Yes	No	Hypermetrope	Yes	Reduced	???

Lenses dataset

Presbyopic	Young	Spectacle prescription	Astigmatic	Tear production rate	Can use contact lenses
No	Yes	Myope	No	Normal	Yes
No	Yes				No
No	Yes				No
No	Yes				Yes
No	No				No
No	No				Yes
No	No				Yes
No	No				No
No	No	Hypermetrope	Yes	Reduced	No
No	No	Hypermetrope	Yes	Normal	No
Yes	No	Myope	No	Normal	No
Yes	No	Hypermetrope	Yes	Normal	No
No	Yes	Myope	No	Reduced	???
No	Yes	Myope	Yes	Normal	???
No	Yes	Hypermetrope	Yes	Reduced	???
No	Yes	Hypermetrope	Yes	Normal	???
No	No	Myope	Yes	Reduced	???
No	No	Hypermetrope	No	Normal	???
Yes	No	Myope	No	Reduced	???
Yes	No	Myope	Yes	Reduced	???
Yes	No	Myope	Yes	Normal	???
Yes	No	Hypermetrope	No	Reduced	???
Yes	No	Hypermetrope	No	Normal	???
Yes	No	Hypermetrope	Yes	Reduced	???

This dataset is about 24 people who want to start using contact lenses and visit the optician (1 row=1 person)

Lenses dataset

Presbyopic	Young	Spectacle prescription	Astigmatic	Tear production rate	Can use contact lenses
No	Yes	Myope	No	Normal	Yes
No	Yes	Myope	Yes	Reduced	No
No	Yes	Hypermetrope	No	Reduced	No
No	Yes	Hypermetrope	No	Normal	Yes
No	No	Myope	No	Reduced	No
No	No	Myope	No	Normal	Yes
No	No	Myope	Yes	Normal	Yes
No	No	Hypermetrope	No	Reduced	No
No	No	Hypermetrope	Yes	Reduced	No
No	No	Hypermetrope	Yes	Normal	No
Yes	No	[illegible]	[illegible]	[illegible]	No
Yes	No	[illegible]	[illegible]	[illegible]	No
No	Yes	[illegible]	[illegible]	[illegible]	???
No	Yes	[illegible]	[illegible]	[illegible]	???
No	Yes	[illegible]	[illegible]	[illegible]	???
No	Yes	[illegible]	[illegible]	[illegible]	???
No	No	[illegible]	[illegible]	[illegible]	???
No	No	Hypermetrope	No	Normal	???
Yes	No	Myope	No	Reduced	???
Yes	No	Myope	Yes	Reduced	???
Yes	No	Myope	Yes	Normal	???
Yes	No	Hypermetrope	No	Reduced	???
Yes	No	Hypermetrope	No	Normal	???
Yes	No	Hypermetrope	Yes	Reduced	???

Optician gathers information (columns 1-5) and decides if the person can use contact lenses (last column)

Lenses dataset

Presbyopic	Young	Spectacle prescription	Astigmatic	Tear production rate	Can use contact lenses
No	Yes	Myope	No	Normal	Yes
No	Yes	Myope	Yes	Reduced	No
No	Yes	Hypermetrope	No	Reduced	No
No	Yes	Hypermetrope	No	Normal	Yes
No	No	Myope	No	Reduced	No
No	No	Myope	No	Normal	Yes
No	No	Myope	Yes	Normal	Yes
No	No	Hypermetrope	No	Reduced	No
No	No	Hypermetrope	Yes	Reduced	No
No	No	Hypermetrope	Yes	Normal	No
Yes	No	Myope	No	Normal	No
Yes	No	Hypermetrope	Yes	Normal	No
No	Yes	Myope	No	Reduced	???
No	Yes	Myope	Yes	Normal	???
No	Yes	Hypermetrope	Yes	Reduced	???
No	Yes	Hypermetrope	Yes	Normal	???
No	No	Myope	Yes	Reduced	???
No	No	Hypermetrope	No	Normal	???
Yes	No	Myope	No	Reduced	???
Yes	No	Myope	Yes	Reduced	???
Yes	No	Myope	Yes	Normal	???
Yes	No	Hypermetrope	No	Reduced	???
Yes	No	Hypermetrope	No	Normal	???
Yes	No	Hypermetrope	Yes	Reduced	???

Lenses dataset

Presbyopic	Young	Spectacle prescription	Astigmatic	Tear production rate	Can use contact lenses
No	Yes	Myope	No	Normal	Yes
No	Yes	Myope	Yes	Reduced	No
No	Yes	Hypermetrope	No	Reduced	No
No	Yes	Hypermetrope	No	Normal	Yes
No	No	Myope	No	Reduced	No
No	No	Myope	No	Normal	Yes
No	No	Myope	Yes	Normal	Yes
No	No	Hypermetrope	No	Reduced	No
No	No	Hypermetrope	Yes	Reduced	No
No	No	Hypermetrope	Yes	Normal	No
Yes	No	Myope	No	Normal	No
Yes	No	Hypermetrope	Yes	Normal	No
No	Yes	Hypermetrope	Yes	Normal	???

Lenses dataset

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0
0	1	1	1	0	???

Lenses dataset

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0
0	1	1	1	0	???

What would you predict for the test instance?

Lenses dataset

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0
0	1	1	1	0	???

First let's consider only the last feature (R).

How to predict?

R	L
0	1
1	0
1	0
0	1
1	0
0	1
0	1
1	0
1	0
0	0
0	0
0	0
0	???

How to predict?

R	L
0	1
1	0
1	0
0	1
1	0
0	1
0	1
1	0
1	0
0	0
0	0
0	0
0	???

Distribution

	L=0	L=1	Majority
R=0	3	4	L=1

How to predict?

R	L
0	1
1	0
1	0
0	1
1	0
0	1
0	1
1	0
1	0
0	0
0	0
0	0
0	???

Distribution

	L=0	L=1	Majority
R=0	3	4	L=1
R=1	5	0	L=0

How to predict?

R	L
0	1
1	0
1	0
0	1
1	0
0	1
0	1
1	0
1	0
0	0
0	0
0	0
0	???

Distribution

	L=0	L=1	Majority
R=0	3	4	L=1
R=1	5	0	L=0

We agreed to predict the most frequent class in the training data!

If R=0 then L=1
If R=1 then L=0

How to predict?

R	L
0	1
1	0
1	0
0	1
1	0
0	1
0	1
1	0
1	0
0	0
0	0
0	0
0	???

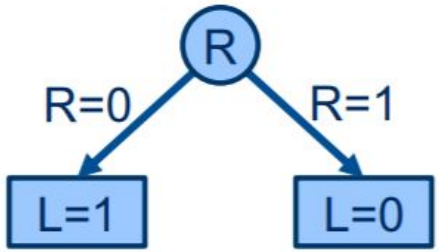
Distribution

	L=0	L=1	Majority
R=0	3	4	L=1
R=1	5	0	L=0

We agreed to predict the most frequent class in the training data!

If R=0 then L=1
If R=1 then L=0

This is called a **decision stump**, that is a decision tree with 1 decision node



How to predict in this case?

A	R	L
0	0	1
1	1	0
0	1	0
0	0	1
0	1	0
0	0	1
1	0	1
0	1	0
1	1	0
1	0	0
0	0	0
1	0	0
1	0	???

How to predict in this case?

A	R	L
0	0	1
1	1	0
0	1	0
0	0	1
0	1	0
0	0	1
1	0	1
0	1	0
1	1	0
1	0	0
0	0	0
1	0	0
1	0	???

Distribution

	L=0	L=1	Majority
A=0, R=0	1	3	L=1
A=0, R=1	3	0	L=0
A=1, R=0	2	1	L=0
A=1, R=1	2	0	L=0

How to predict in this case?

A	R	L
0	0	1
1	1	0
0	1	0
0	0	1
0	1	0
0	0	1
1	0	1
0	1	0
1	1	0
1	0	0
0	0	0
1	0	0
1	0	???

Distribution

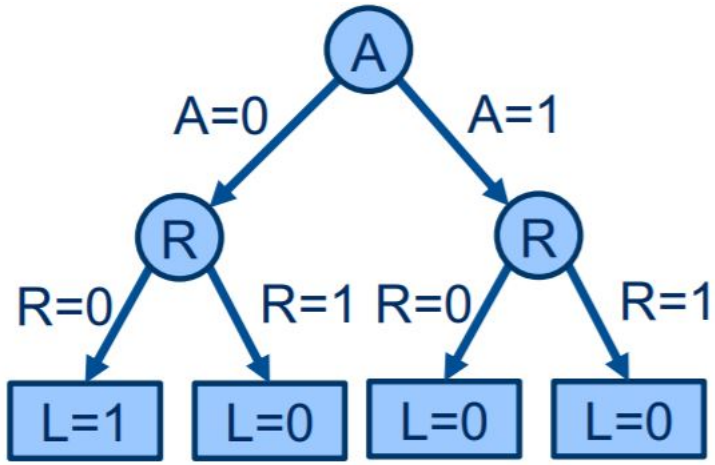
	L=0	L=1	Majority
A=0, R=0	1	3	L=1
A=0, R=1	3	0	L=0
A=1, R=0	2	1	L=0
A=1, R=1	2	0	L=0

We agreed to predict the most frequent class in the training data!

If A=0, R=0 then L=1
If A=0, R=1 then L=0
If A=1, R=0 then L=0
If A=1, R=1 then L=0

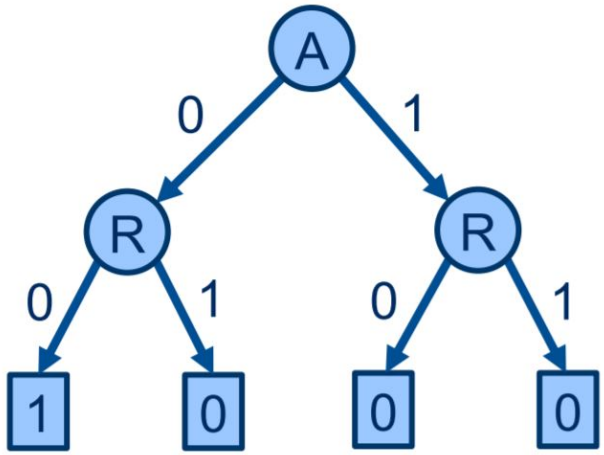
Decision Tree (features A, R)

If A=0, R=0 then L=1
If A=0, R=1 then L=0
If A=1, R=0 then L=0
If A=1, R=1 then L=0



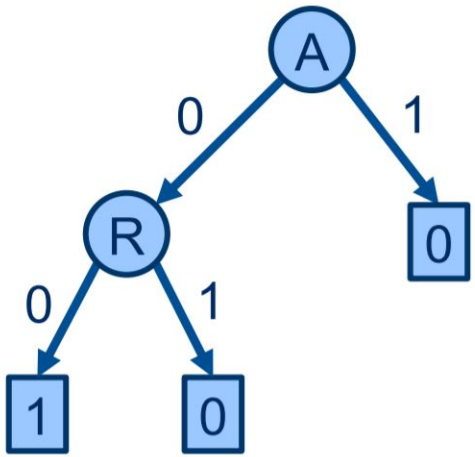
Decision Tree (features A, R)

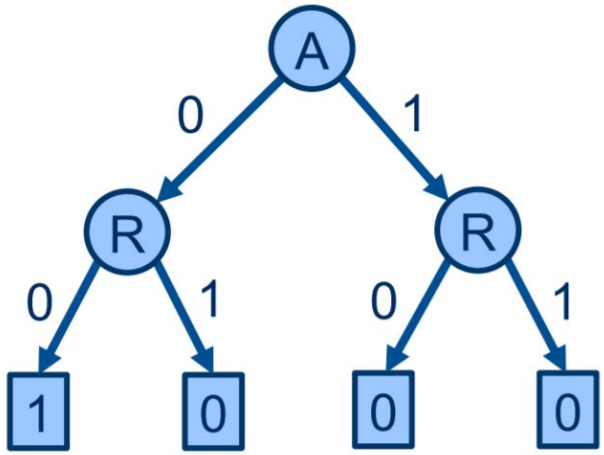
If A=0, R=0 then L=1
If A=0, R=1 then L=0
If A=1, R=0 then L=0
If A=1, R=1 then L=0



Decision Tree (features A, R)

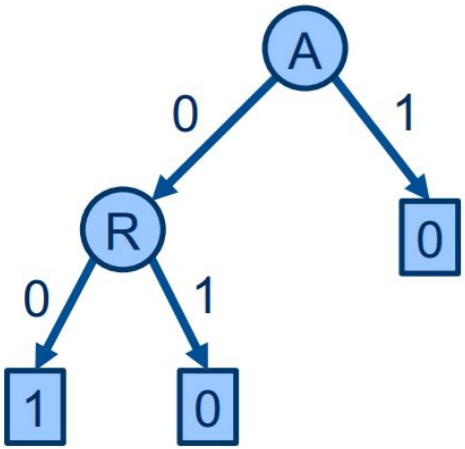
If A=0, R=0 then L=1
If A=0, R=1 then L=0
If A=1, R=0 then L=0
If A=1, R=1 then L=0

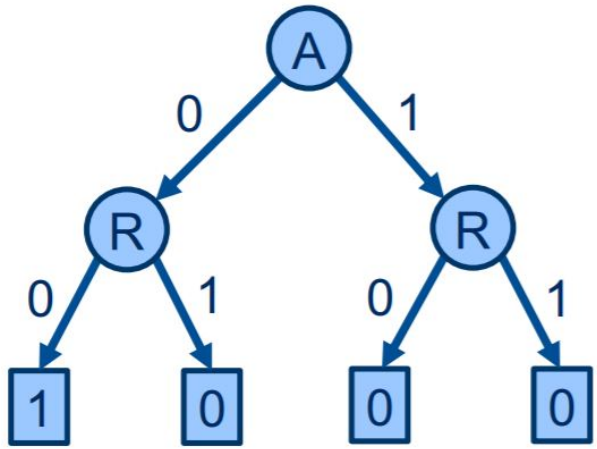


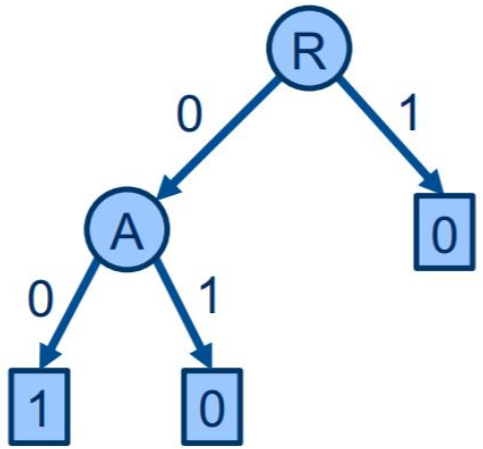


Decision Tree (features A, R)

If A=0, R=0 then L=1
If A=0, R=1 then L=0
If A=1, R=0 then L=0
If A=1, R=1 then L=0

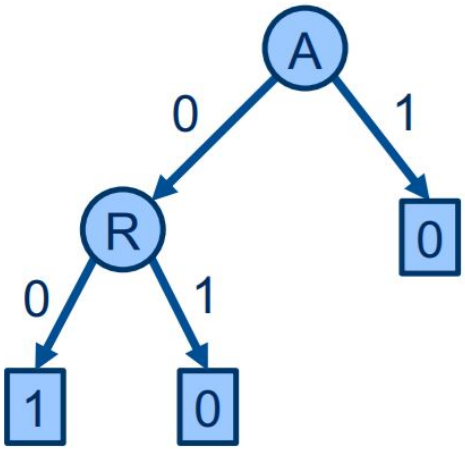


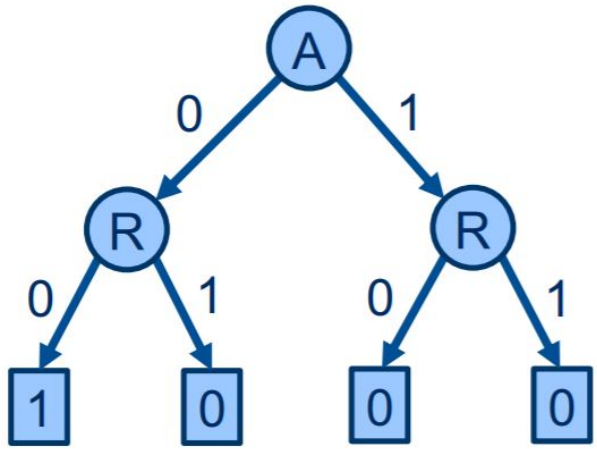


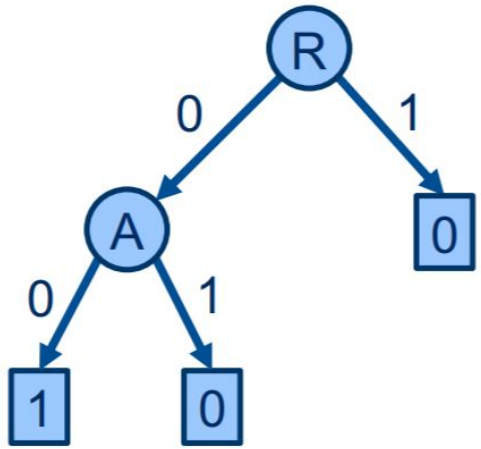


Decision Tree (features A, R)

If A=0, R=0 then L=1
If A=0, R=1 then L=0
If A=1, R=0 then L=0
If A=1, R=1 then L=0





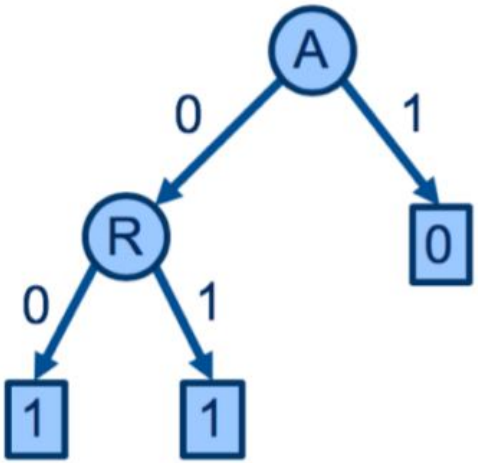


These decision trees are equivalent!

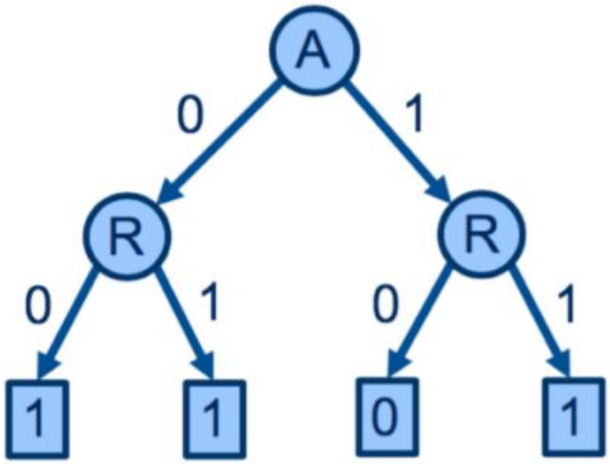
Question

Which of these decision trees are equivalent?

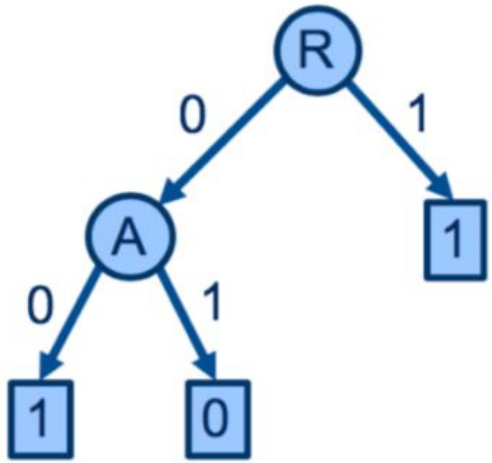
Tree 1



Tree 2



Tree 3



Adding one more feature

H	A	R	L
0	0	0	1
0	1	1	0
1	0	1	0
1	0	0	1
0	0	1	0
0	0	0	1
0	1	0	1
1	0	1	0
1	1	1	0
1	1	0	0
0	0	0	0
1	1	0	0
1	1	0	???

Adding one more feature

H	A	R	L
0	0	0	1
0	1	1	0
1	0	1	0
1	0	0	1
0	0	1	0
0	0	0	1
0	1	0	1
1	0	1	0
1	1	1	0
1	1	0	0
0	0	0	0
1	1	0	0
1	1	0	???

Distribution

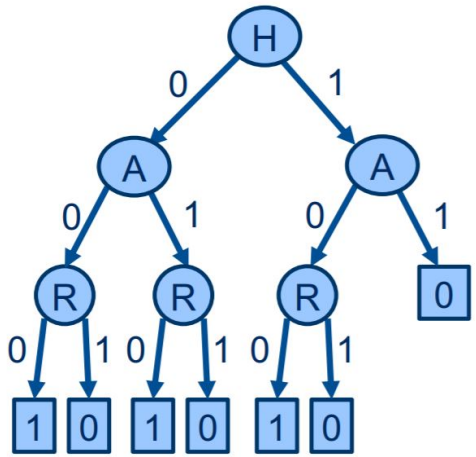
	L=0	L=1	Majority
H=0, A=0, R=0	1	2	L=1
H=0, A=0, R=1	1	0	L=0
H=0, A=1, R=0	0	1	L=1
H=0, A=1, R=1	1	0	L=0
H=1, A=0, R=0	0	1	L=1
H=1, A=0, R=1	2	0	L=0
H=1, A=1, R=0	2	0	L=0
H=1, A=1, R=1	1	0	L=0

Decision Tree (features H, A, R)

	L=0	L=1	Majority
H=0, A=0, R=0	1	2	L=1
H=0, A=0, R=1	1	0	L=0
H=0, A=1, R=0	0	1	L=1
H=0, A=1, R=1	1	0	L=0
H=1, A=0, R=0	0	1	L=1
H=1, A=0, R=1	2	0	L=0
H=1, A=1, R=0	2	0	L=0
H=1, A=1, R=1	1	0	L=0

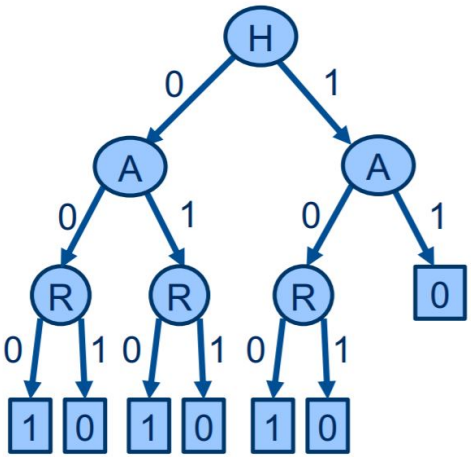
Decision Tree (features H, A, R)

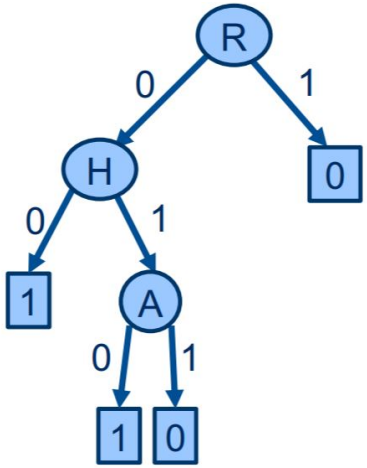
	L=0	L=1	Majority
H=0, A=0, R=0	1	2	L=1
H=0, A=0, R=1	1	0	L=0
H=0, A=1, R=0	0	1	L=1
H=0, A=1, R=1	1	0	L=0
H=1, A=0, R=0	0	1	L=1
H=1, A=0, R=1	2	0	L=0
H=1, A=1, R=0	2	0	L=0
H=1, A=1, R=1	1	0	L=0



Decision Tree (features H, A, R)

	L=0	L=1	Majority
H=0, A=0, R=0	1	2	L=1
H=0, A=0, R=1	1	0	L=0
H=0, A=1, R=0	0	1	L=1
H=0, A=1, R=1	1	0	L=0
H=1, A=0, R=0	0	1	L=1
H=1, A=0, R=1	2	0	L=0
H=1, A=1, R=0	2	0	L=0
H=1, A=1, R=1	1	0	L=0





Decision tree on all features?

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0
0	1	1	1	0	???

Decision tree on all features?

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0
0	1	1	1	0	???

Distribution

	L=0	L=1
...	...	...
P=0, Y=1, H=1, A=0, R=1	1	0
P=0, Y=1, H=1, A=1, R=0	0	0
...	...	...

Decision tree on all features?

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0
0	1	1	1	0	???

Distribution

	L=0	L=1
...	...	...
P=0, Y=1, H=1, A=0, R=1	1	0
P=0, Y=1, H=1, A=1, R=0	0	0
...	...	...

What to predict for this case?

Building a decision tree by recursively splitting the training data

- Let's fix the order of decision nodes to be P, Y, H, A, R

Building a decision tree by recursively splitting the training data

- Let's fix the order of decision nodes to be P, Y, H, A, R
- Split by first feature P:

	L=0	L=1	Pure?
P=0	6	4	No
P=1	2	0	Yes, L=0

Building a decision tree by recursively splitting the training data

- Let's fix the order of decision nodes to be P, Y, H, A, R
- Split by first feature P:

	L=0	L=1	Pure?
P=0	6	4	No
P=1	2	0	Yes, L=0

- P=1 is **pure** (all points belong to the same class), no need to split further

Building a decision tree by recursively splitting the training data

- Let's fix the order of decision nodes to be P, Y, H, A, R
- Split by first feature P:

	L=0	L=1	Pure?
P=0	6	4	No
P=1	2	0	Yes, L=0

- P=1 is **pure** (all points belong to the same class), no need to split further
- Split P=0 by feature Y:

	L=0	L=1	Pure?
P=0, Y=0	4	2	No
P=0, Y=1	2	2	No
P=1	2	0	Yes, L=0

Building a decision tree by recursively splitting the training data

- Let's fix the order of decision nodes to be P, Y, H, A, R
- Split by first feature P:

	L=0	L=1	Pure?
P=0	6	4	No
P=1	2	0	Yes, L=0

- P=1 is **pure** (all points belong to the same class), no need to split further
- Split P=0 by feature Y:

	L=0	L=1	Pure?
P=0, Y=0	4	2	No
P=0, Y=1	2	2	No
P=1	2	0	Yes, L=0

- Two new nodes are not pure, continue splitting

Building a decision tree by recursively splitting the training data

- Let's fix the order of decision nodes to be P, Y, H, A, R
- Split by first feature P:

	L=0	L=1	Pure?
P=0	6	4	No
P=1	2	0	Yes, L=0

- P=1 is **pure** (all points belong to the same class), no need to split further
- Split P=0 by feature Y:

	L=0	L=1	Pure?
P=0, Y=0	4	2	No
P=0, Y=1	2	2	No
P=1	2	0	Yes, L=0

- Two new nodes are not pure, continue splitting
- Split recursively P=0, Y=0 and P=0, Y=1 by H

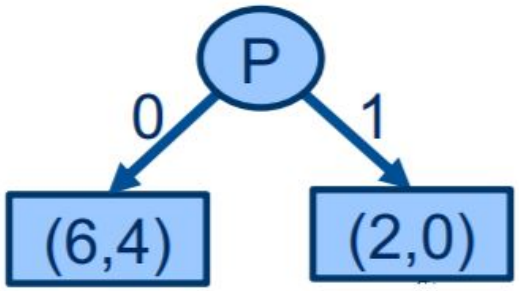
Building a decision tree by recursively splitting the training data

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0

(8,4)

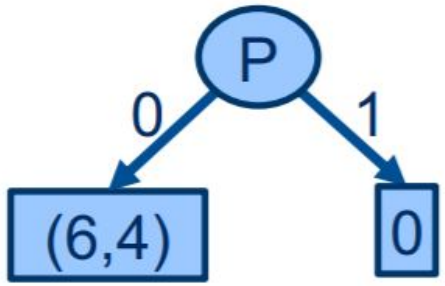
Building a decision tree by recursively splitting the training data

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



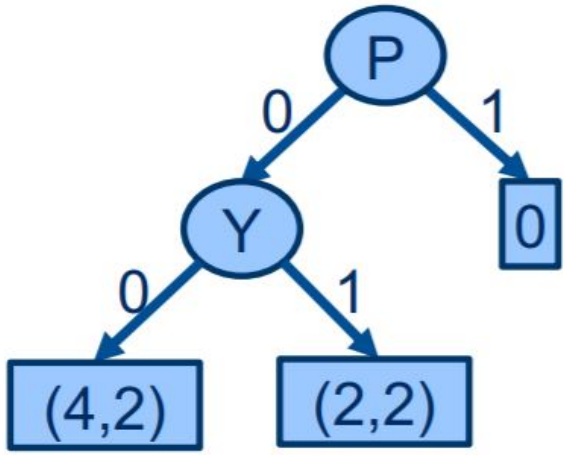
Building a decision tree by recursively splitting the training data

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



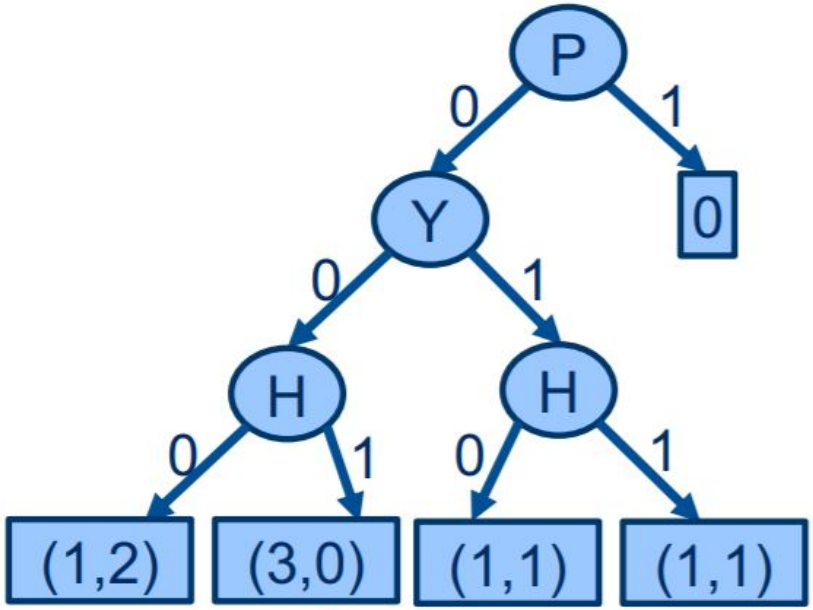
Building a decision tree by recursively splitting the training data

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



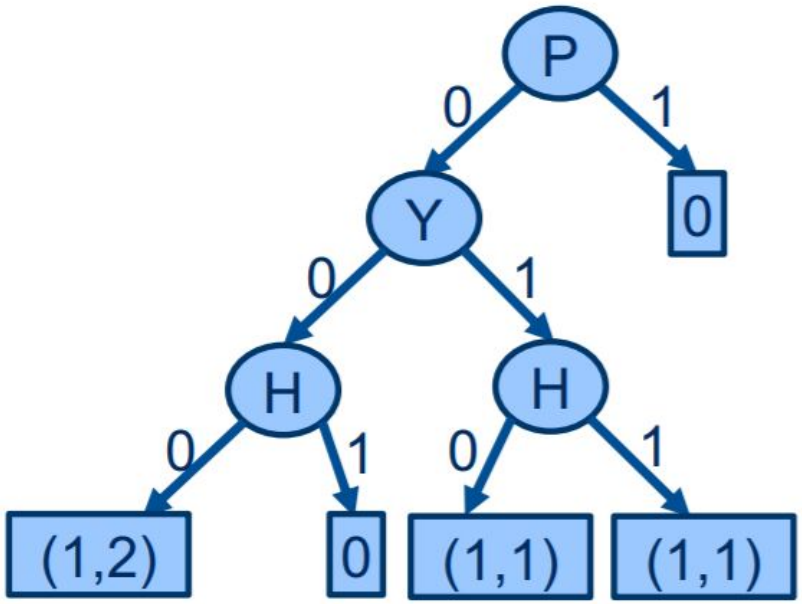
Building a decision tree by recursively splitting the training data

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



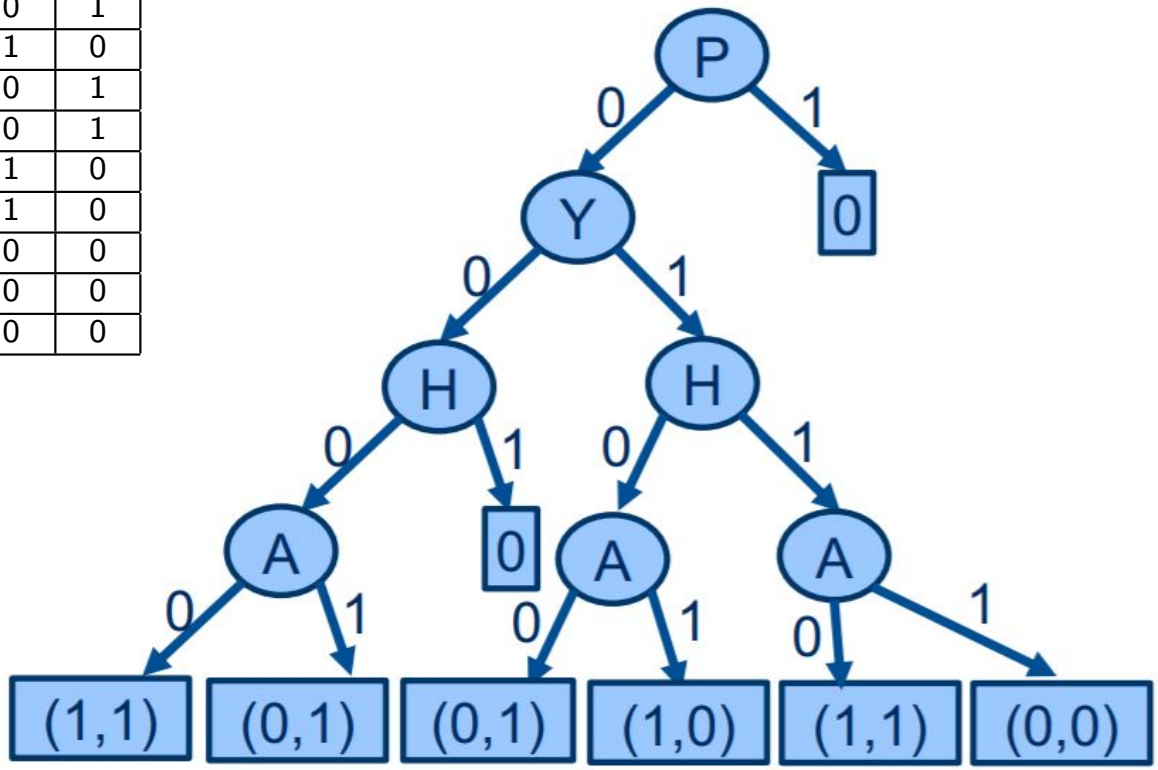
Building a decision tree by recursively splitting the training data

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



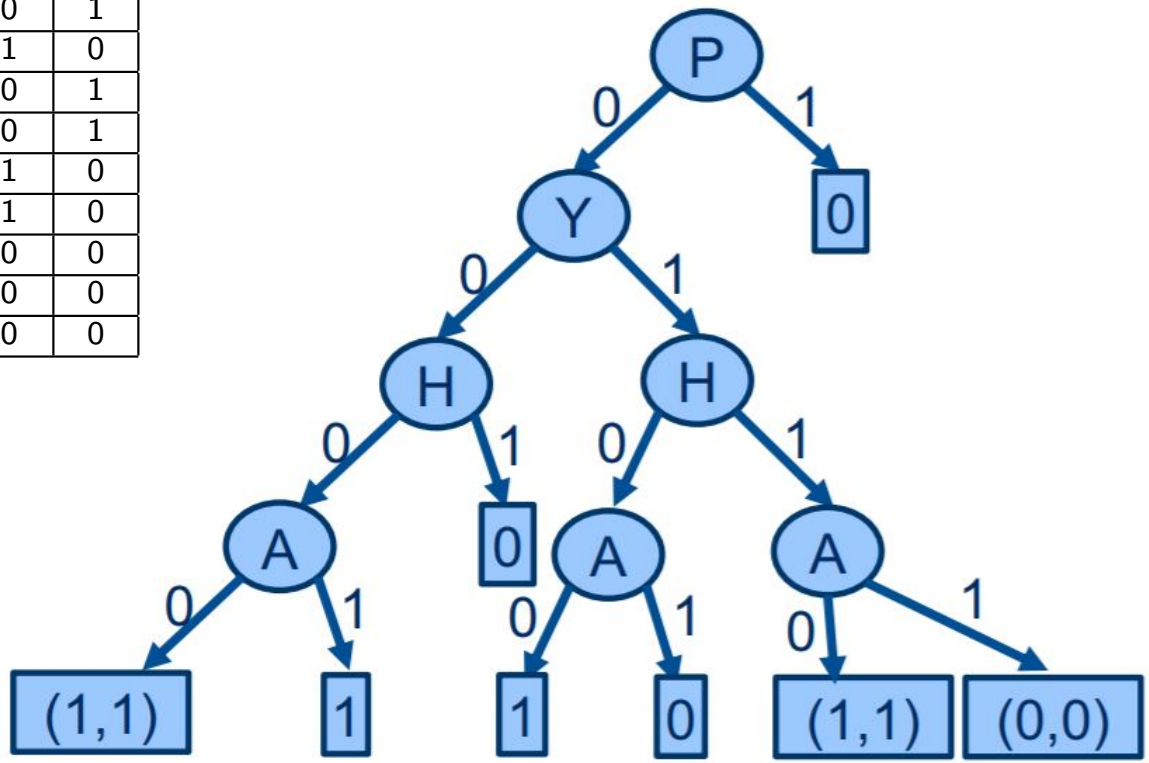
Building a decision tree by recursively splitting the training data

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



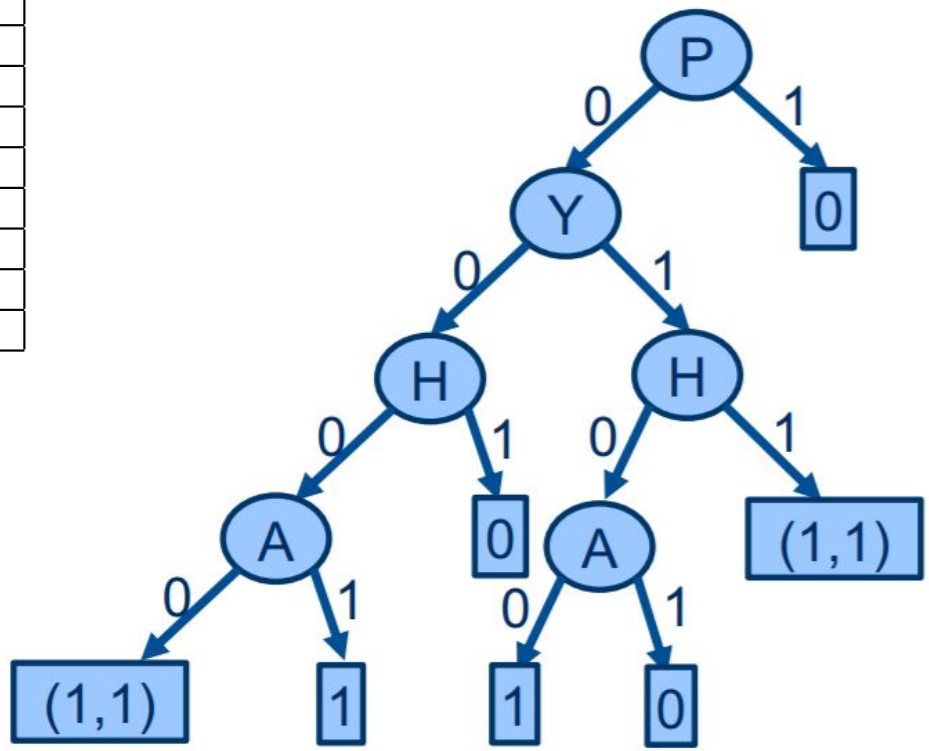
Building a decision tree by recursively splitting the training data

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



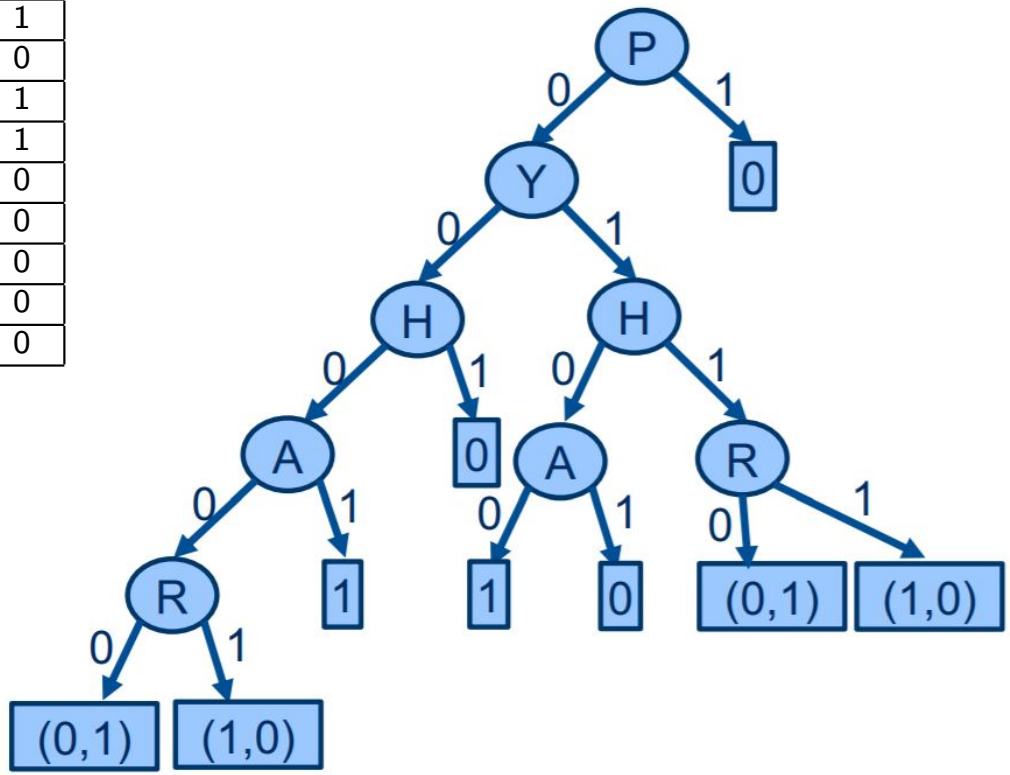
Building a decision tree by recursively splitting the training data

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



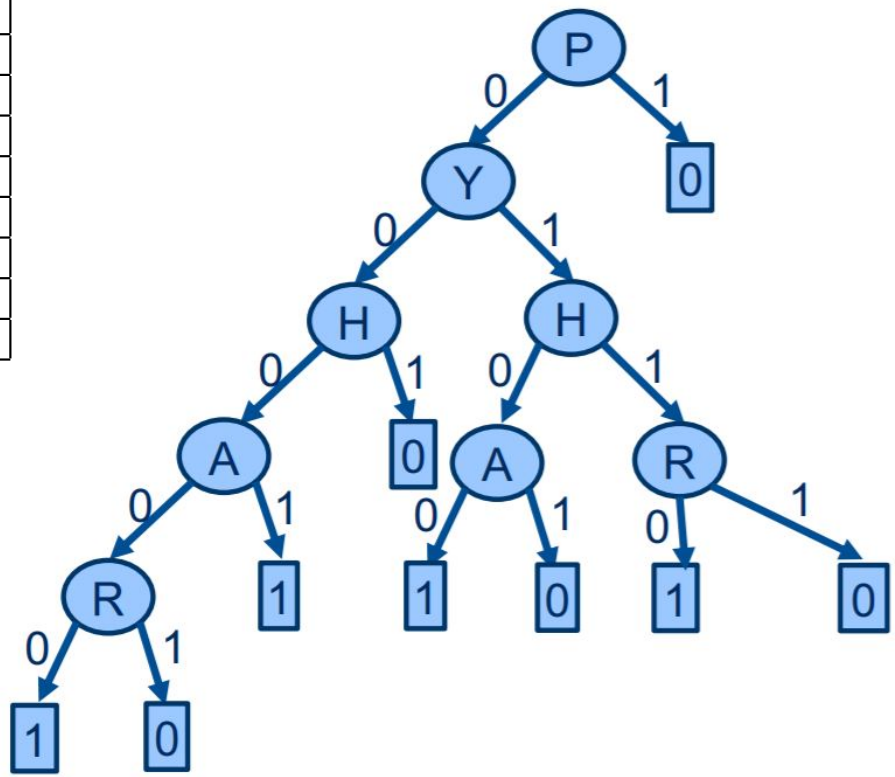
Building a decision tree by recursively splitting the training data

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



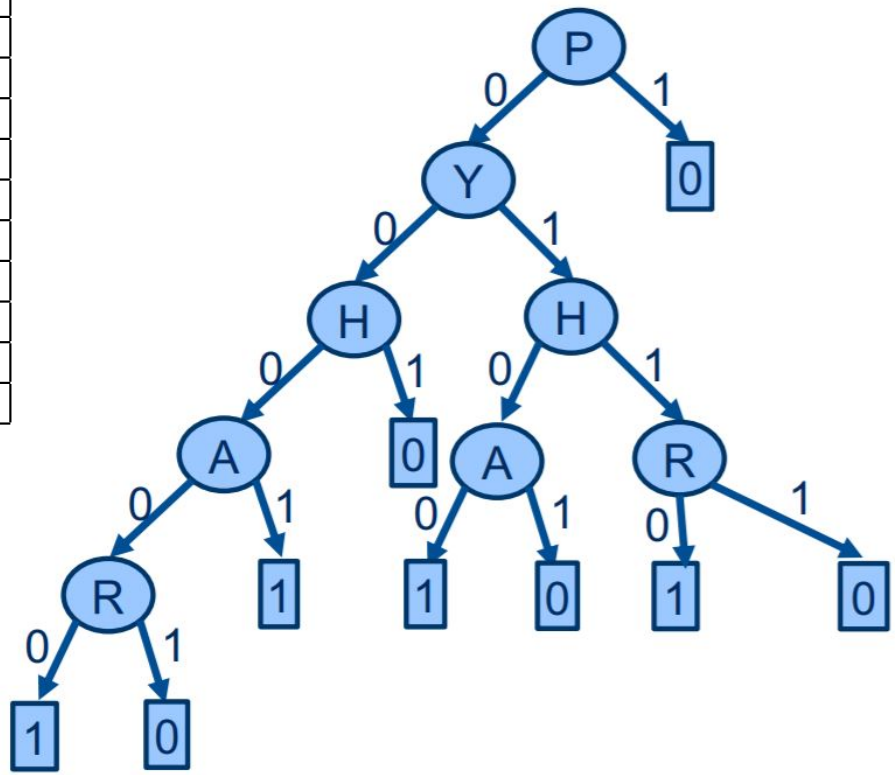
Building a decision tree by recursively splitting the training data

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



Building a decision tree by recursively splitting the training data

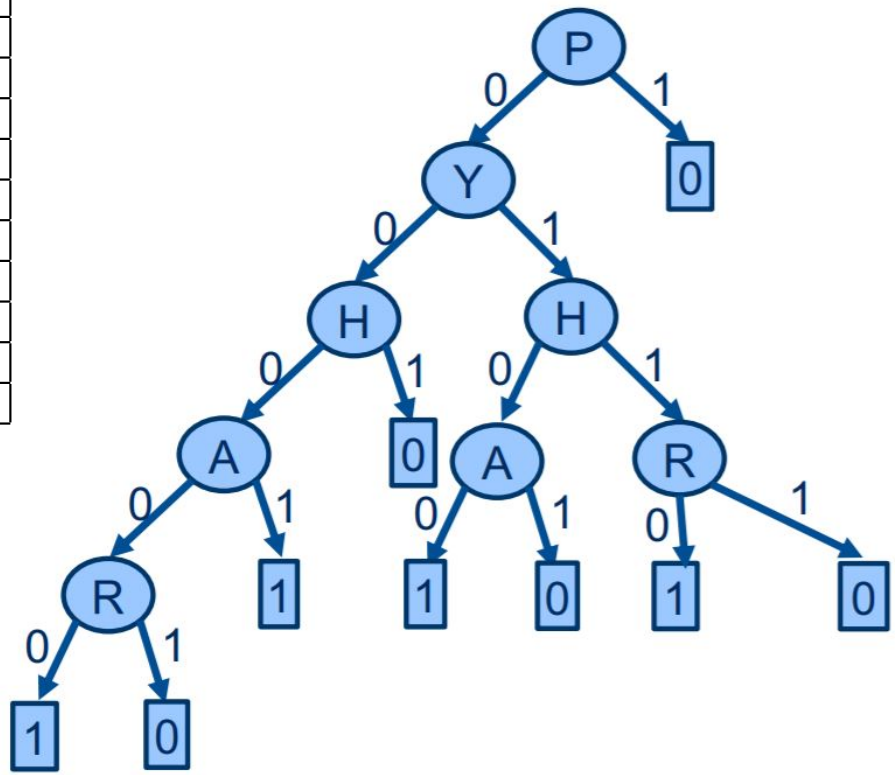
P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



Predict for test instance **P=0, Y=1, H=1, A=1, R=0**:

Building a decision tree by recursively splitting the training data

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



Predict for test instance **P=0, Y=1, H=1, A=1, R=0**: **L=1**

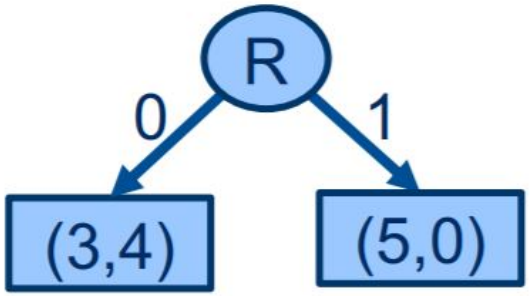
Building a decision tree by recursively splitting the training data (different fixed order: R, P, H, A, Y)

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0

(8,4)

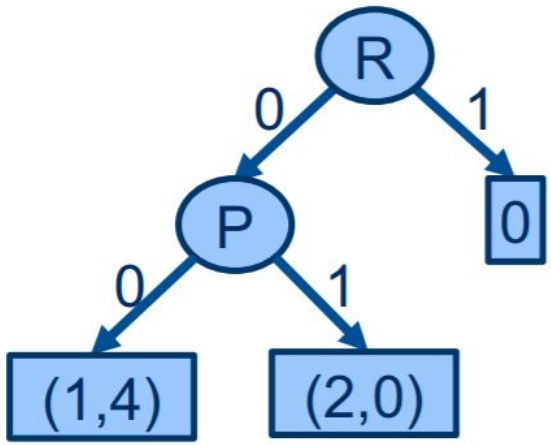
Building a decision tree by recursively splitting the training data (different fixed order: R, P, H, A, Y)

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



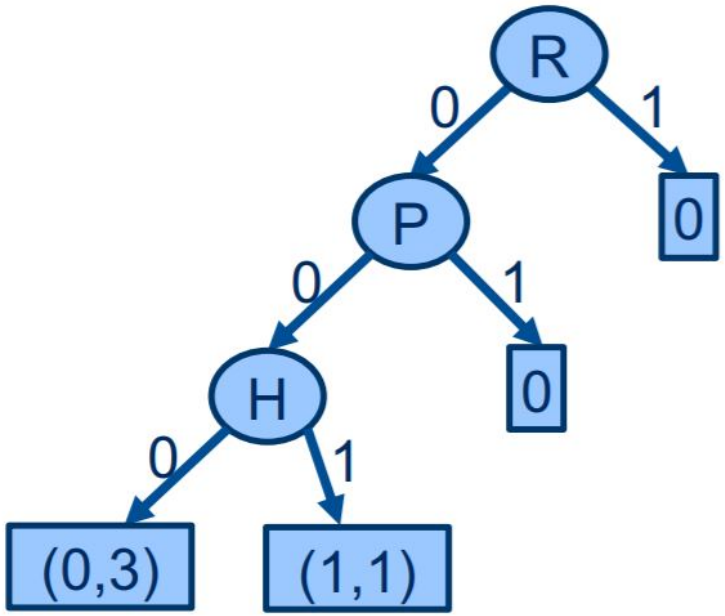
Building a decision tree by recursively splitting the training data (different fixed order: R, P, H, A, Y)

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



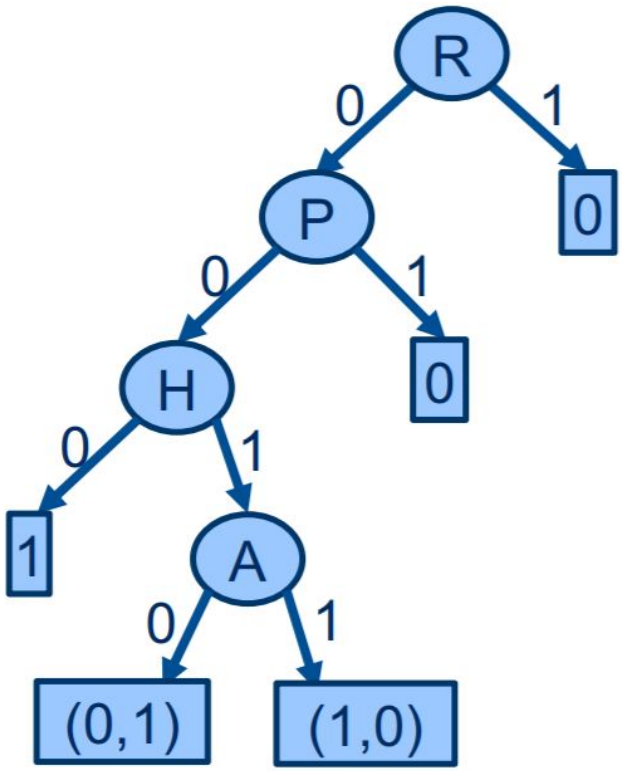
Building a decision tree by recursively splitting the training data (different fixed order: R, P, H, A, Y)

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



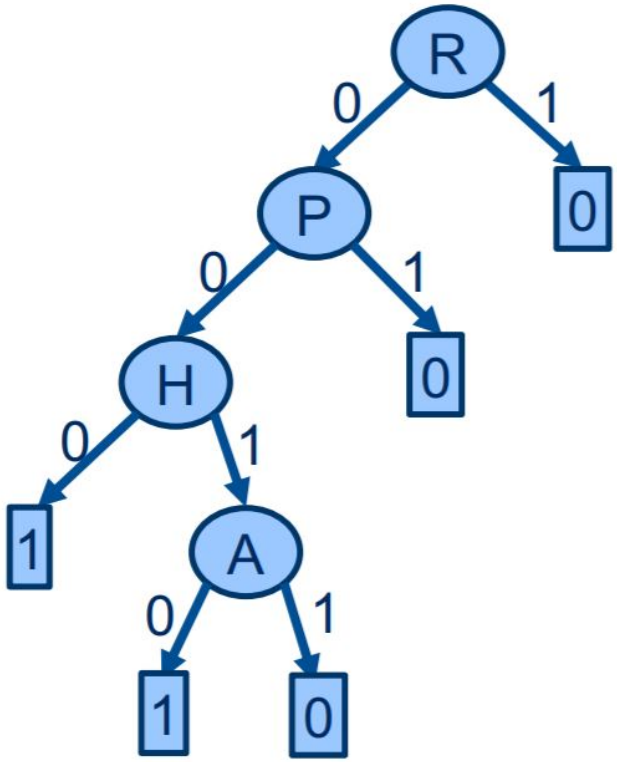
Building a decision tree by recursively splitting the training data (different fixed order: R, P, H, A, Y)

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



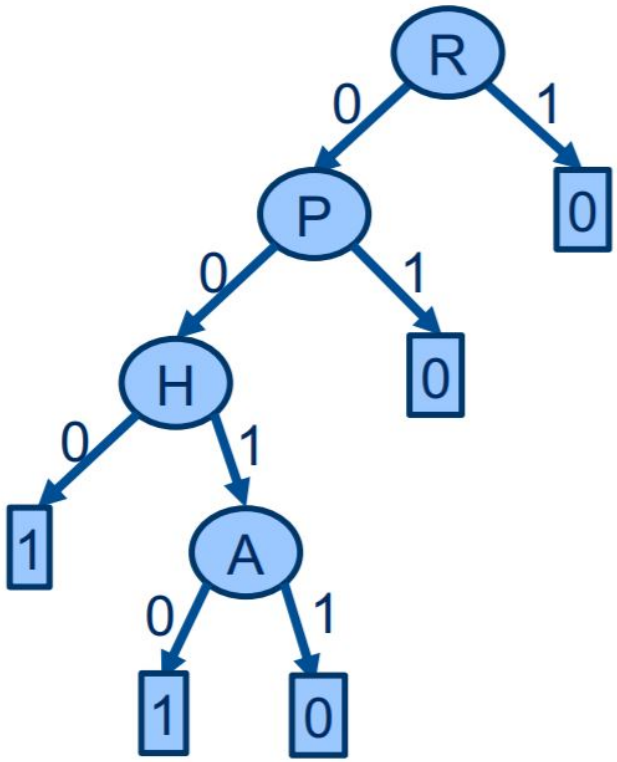
Building a decision tree by recursively splitting the training data (different fixed order: R, P, H, A, Y)

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



Building a decision tree by recursively splitting the training data (different fixed order: R, P, H, A, Y)

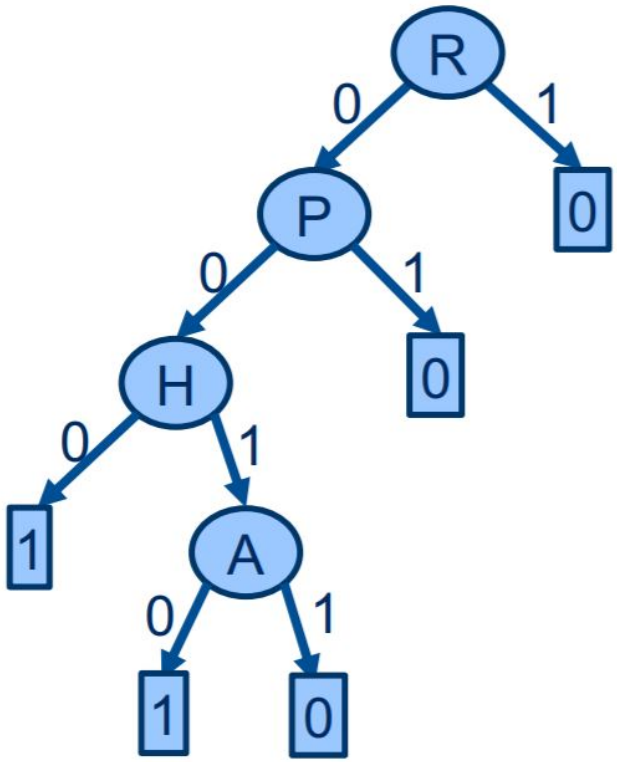
P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



Predict for test instance **P=0, Y=1, H=1, A=1, R=0**:

Building a decision tree by recursively splitting the training data (different fixed order: R, P, H, A, Y)

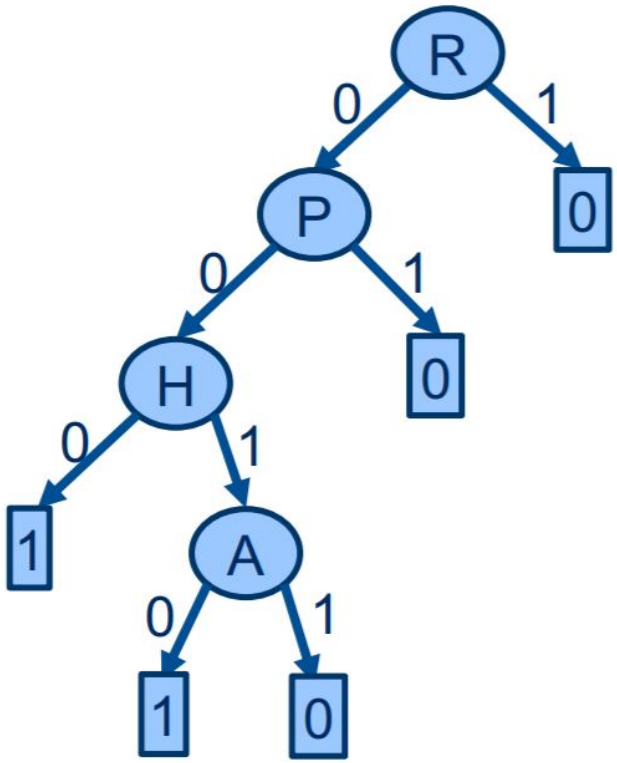
P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



Predict for test instance **P=0, Y=1, H=1, A=1, R=0**: **L=0**

Building a decision tree by recursively splitting the training data (different fixed order: R, P, H, A, Y)

P	Y	H	A	R	L
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	0	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	0



Predict for test instance **P=0, Y=1, H=1, A=1, R=0**: **L=0**

Different order of features resulted in a different tree and prediction

Which decision tree is better?

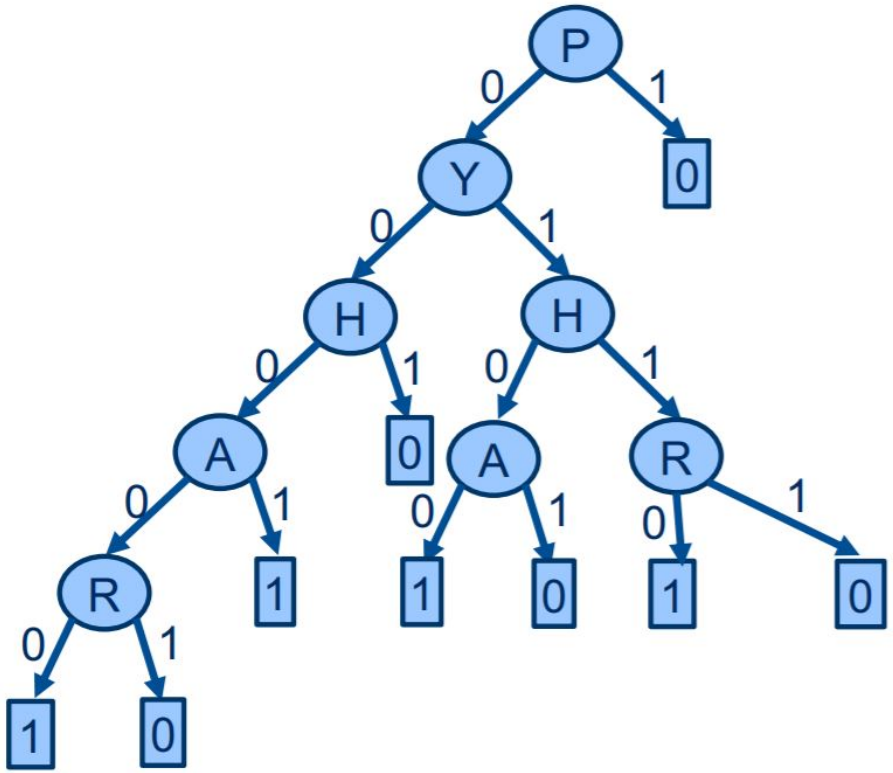
- Consider another test instance: **P=0, Y=0, H=0, A=1, R=1**

Which decision tree is better?

- Consider another test instance: **P=0, Y=0, H=0, A=1, R=1**
- What do the 2 trees predict?

Which decision tree is better?

- Consider another test instance: **P=0, Y=0, H=0, A=1, R=1**
- What do the 2 trees predict?
  - Decision tree from order P, Y, H, A, R:

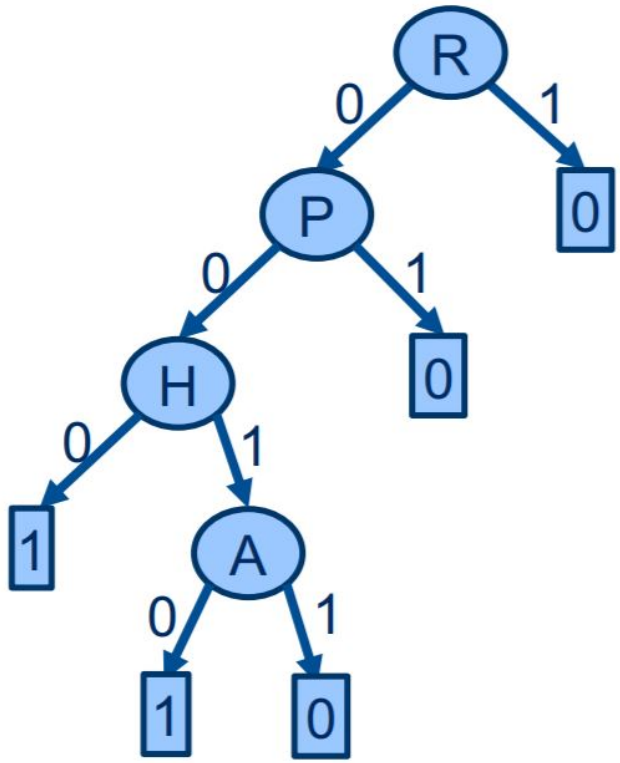


Which decision tree is better?

- Consider another test instance: **P=0, Y=0, H=0, A=1, R=1**
- What do the 2 trees predict?
  - Decision tree from order P, Y, H, A, R: **L=1** (learned from 1 training instance)

Which decision tree is better?

- Consider another test instance: **P=0, Y=0, H=0, A=1, R=1**
- What do the 2 trees predict?
  - Decision tree from order P, Y, H, A, R: **L=1** (learned from 1 training instance)
  - Decision tree from order R, P, H, A, Y:



Which decision tree is better?

- Consider another test instance: **P=0, Y=0, H=0, A=1, R=1**
- What do the 2 trees predict?
  - Decision tree from order P, Y, H, A, R: **L=1** (learned from 1 training instance)
  - Decision tree from order R, P, H, A, Y: **L=0** (learned from 5 training instance)

Which decision tree is better?

- Consider another test instance: **P=0, Y=0, H=0, A=1, R=1**
- What do the 2 trees predict?
  - Decision tree from order P, Y, H, A, R:
    "**I predict L=1 because the only training instance with P=0,Y=0,H=0,A=1 (but R=0) had L=1**"
  - Decision tree from order R, P, H, A, Y:
    "**I predict L=0 because all 5 training instances with R=1 always had L=0**"

Which decision tree is better?

- Consider another test instance: **P=0, Y=0, H=0, A=1, R=1**
- What do the 2 trees predict?
  - Decision tree from order P, Y, H, A, R:
    "**I predict L=1 because the only training instance with P=0,Y=0,H=0,A=1 (but R=0) had L=1**"
  - Decision tree from order R, P, H, A, Y:
    "**I predict L=0 because all 5 training instances with R=1 always had L=0**"
- Which one would you believe?

Which decision tree is better?

- Consider another test instance: **P=0, Y=0, H=0, A=1, R=1**
- What do the 2 trees predict?
  - Decision tree from order P, Y, H, A, R:
    "**I predict it is ok for you to use contact lenses because optician said ok to another person who was also prepresbyopic, myopic and astigmatic except that unlike you he/she had normal tear production rate**"
  - Decision tree from order R, P, H, A, Y:
    "**I predict you should not use contact lenses because optician suggested so to all 5 people with reduced tear production rate**"

Which decision tree is better?

- Consider another test instance: **P=0, Y=0, H=0, A=1, R=1**
- What do the 2 trees predict?
  - Decision tree from order P, Y, H, A, R:
    "**I predict it is ok for you to use contact lenses because optician said ok to another person who was also prepresbyopic, myopic and astigmatic except that unlike you he/she had normal tear production rate**"
  - Decision tree from order R, P, H, A, Y:
    "**I predict you should not use contact lenses because optician suggested so to all 5 people with reduced tear production rate**"
- Which one would you believe now?

Which decision tree is better?

- Usually, pure decision nodes covered by more training instances predict better
- Therefore, **usually** smaller trees are better, because on average, they have higher coverage
- Often it is better to stop splitting if the node becomes small (< 10 or < 20 instances), even if the node is not yet pure

How to order the features in splitting?

- What is a good order?

How to order the features in splitting?

- What is a good order?
  - Achieving purity sooner is better

How to order the features in splitting?

- What is a good order?
  - Achieving purity sooner is better
  - Because then the tree would be smaller

How to order the features in splitting?

- What is a good order?
  - Achieving purity sooner is better
  - Because then the tree would be smaller
- How can we achieve purity sooner?

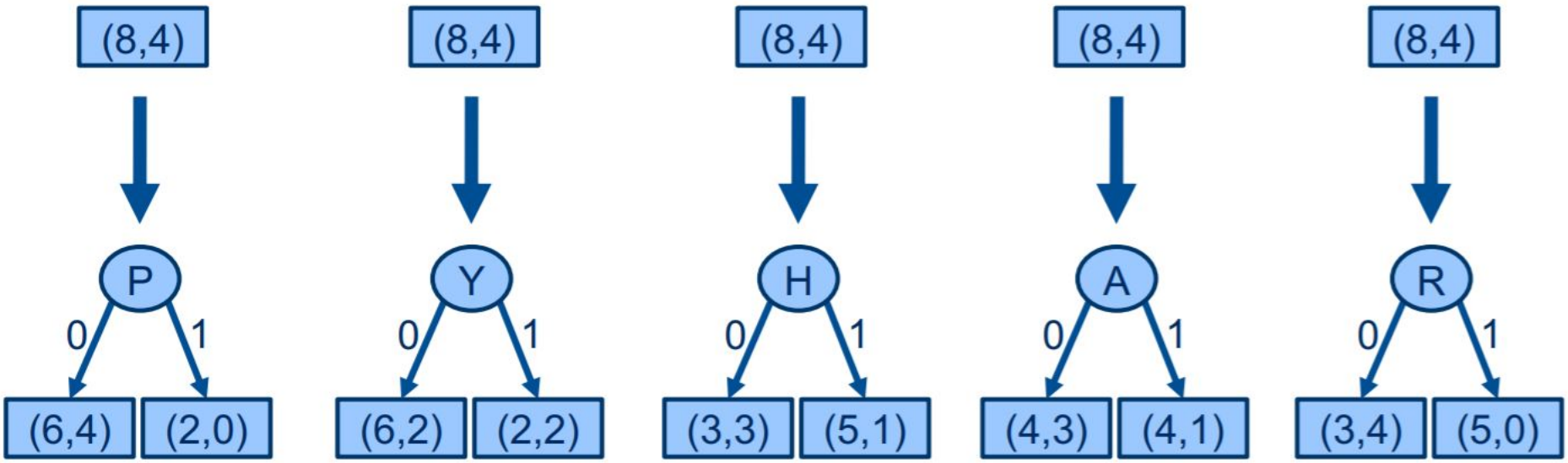
How to order the features in splitting?

- What is a good order?
  - Achieving purity sooner is better
  - Because then the tree would be smaller
- How can we achieve purity sooner?
- Choose the feature which increases purity the most

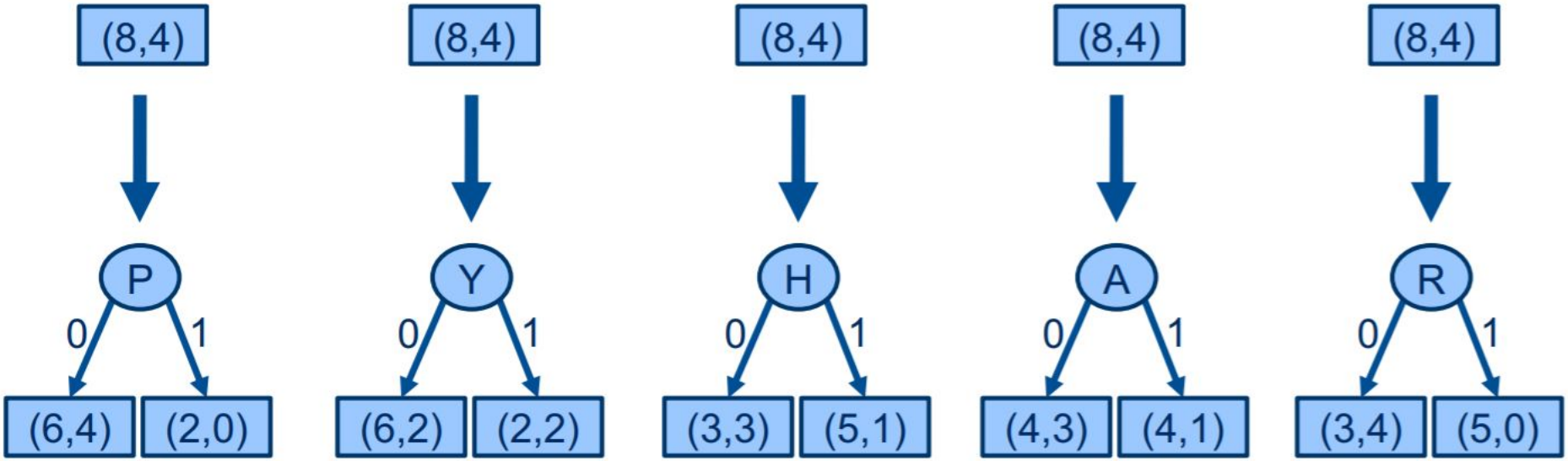
How to order the features in splitting?

- What is a good order?
  - Achieving purity sooner is better
  - Because then the tree would be smaller
- How can we achieve purity sooner?
- Choose the feature which increases purity the most
- Therefore, we need to be able to measure purity

Which split increases purity the most?

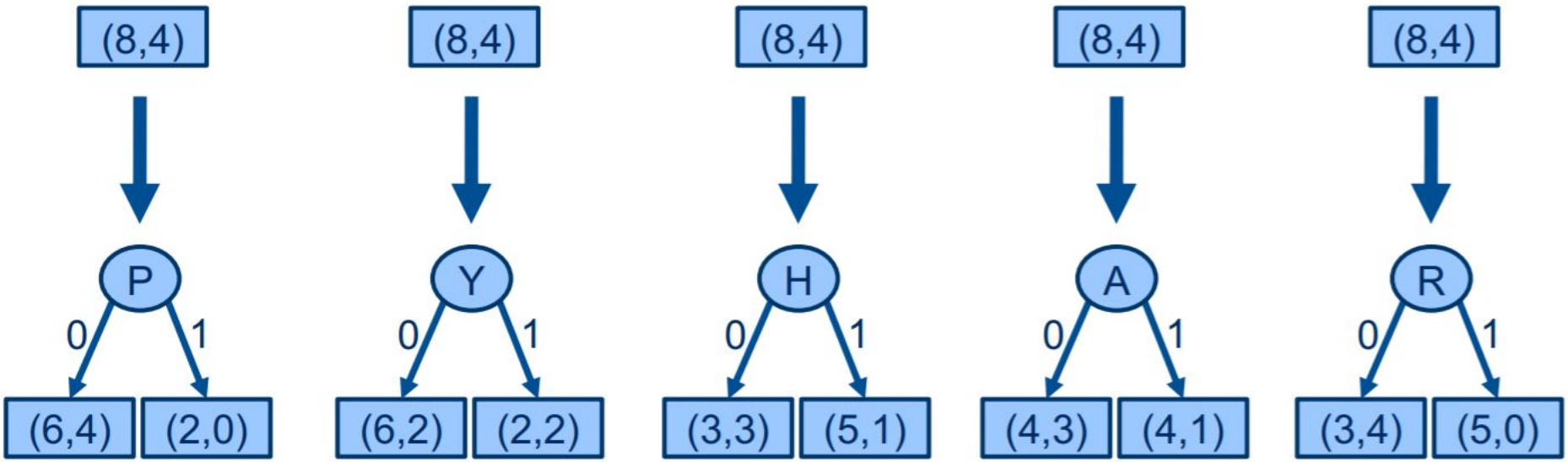


Which split increases purity the most?



Gini impurity of (N_0, N_1): $Gini = 2p(1-p)$ where $p = \dfrac{N_1}{N_0 + N_1}$

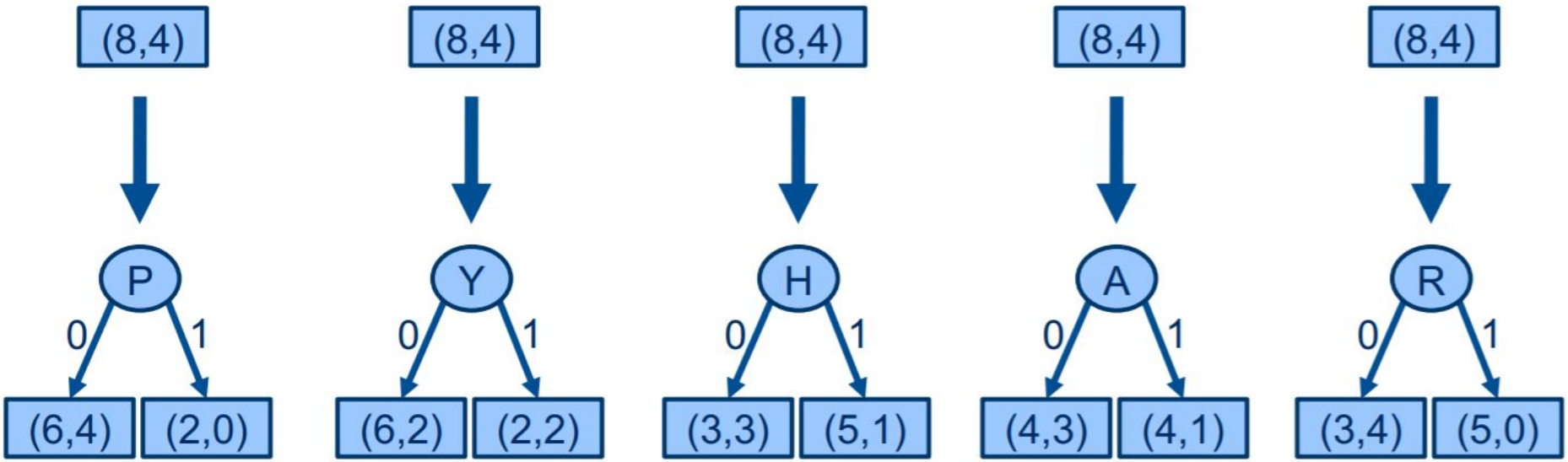
Which split increases purity the most?



p : 0.4 0 0.25 0.5 0.5 0.17 0.43 0.2 0.57 0

Gini impurity of (N_0, N_1): $Gini = 2p(1-p)$ where $p = \dfrac{N_1}{N_0 + N_1}$

Which split increases purity the most?

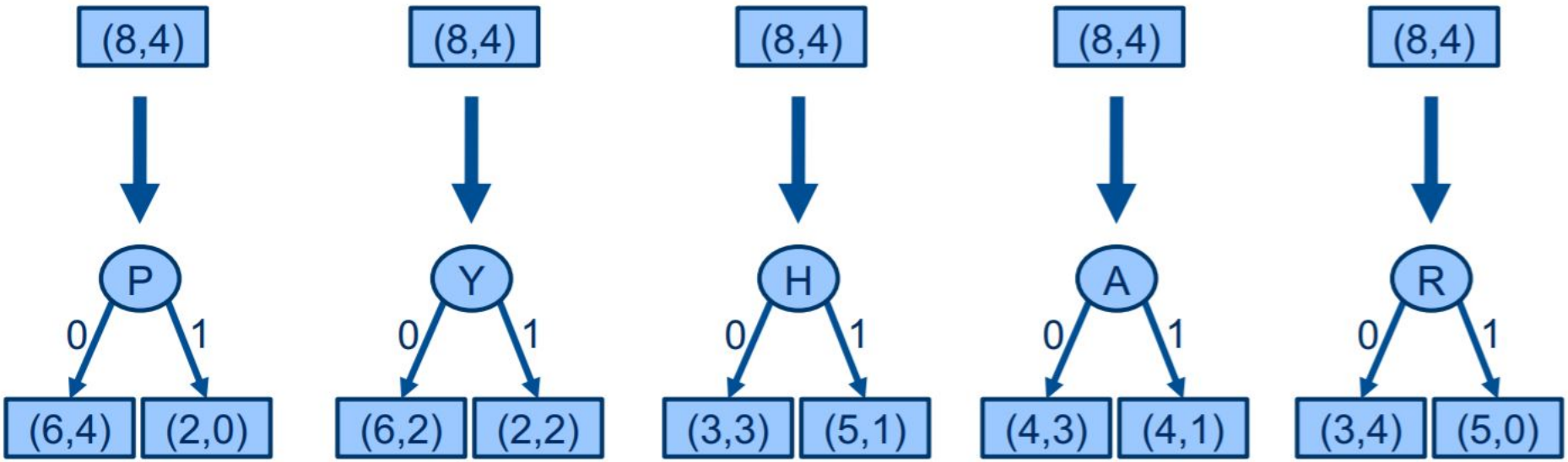


p : 0.4 0 0.25 0.5 0.5 0.17 0.43 0.2 0.57 0

$Gini$: 0.48 0 0.38 0.5 0.5 0.28 0.49 0.32 0.49 0

Gini impurity of (N_0, N_1): $Gini = 2p(1-p)$ where $p = \dfrac{N_1}{N_0 + N_1}$

Which split increases purity the most?



p :	0.4	0		0.25	0.5		0.5	0.17	
$Gini$:	0.48	0		0.38	0.5		0.5	0.28	
$AvgGini$:	0.4			0.42			0.39		

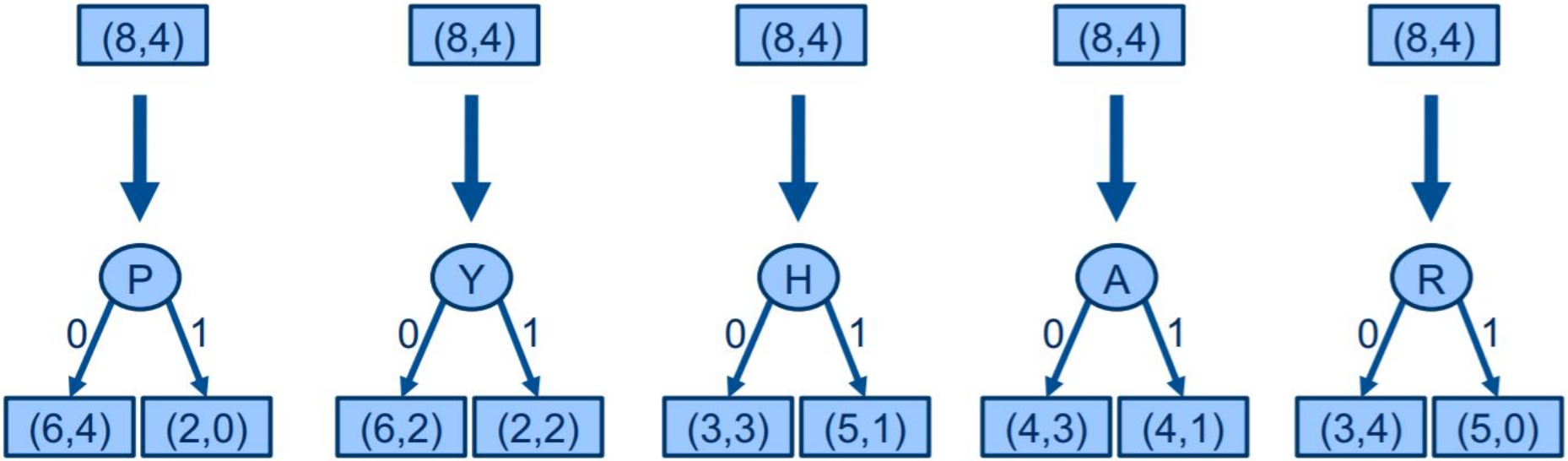
	0.43	0.2		0.57	0
	0.49	0.32		0.49	0
	0.42			0.29	

Gini impurity of (N_0, N_1): $Gini = 2p(1-p)$ where $p = \dfrac{N_1}{N_0 + N_1}$

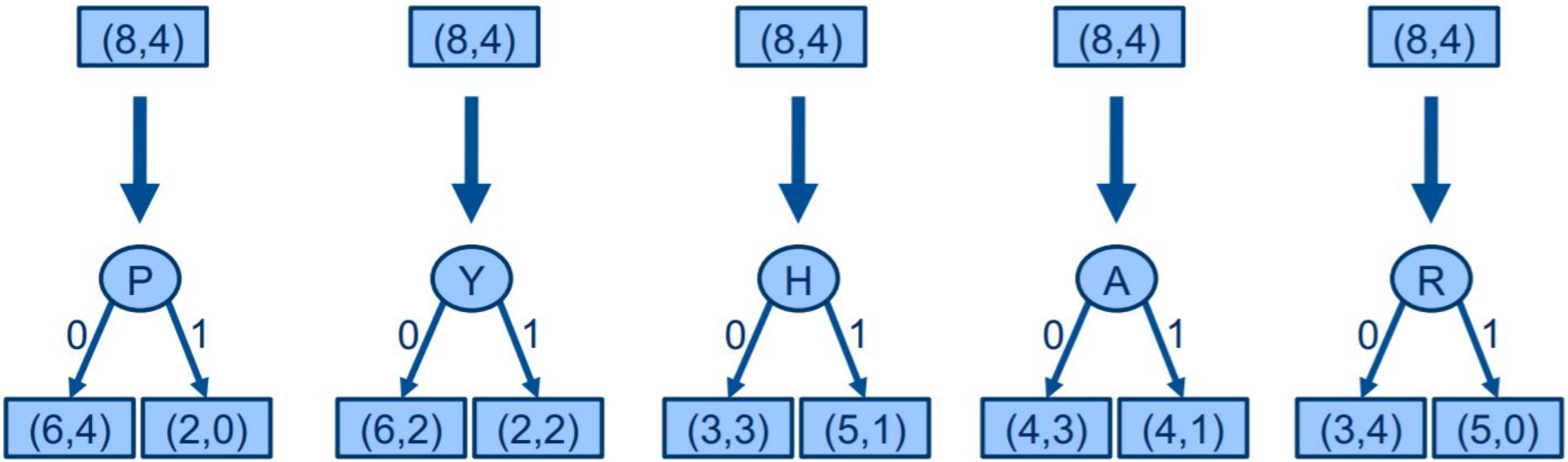
Average Gini impurity after split:

$$AvgGini = \frac{Size_{left} Gini_{left} + Size_{right} Gini_{right}}{Size_{left} + Size_{right}}$$

Which split increases purity the most?

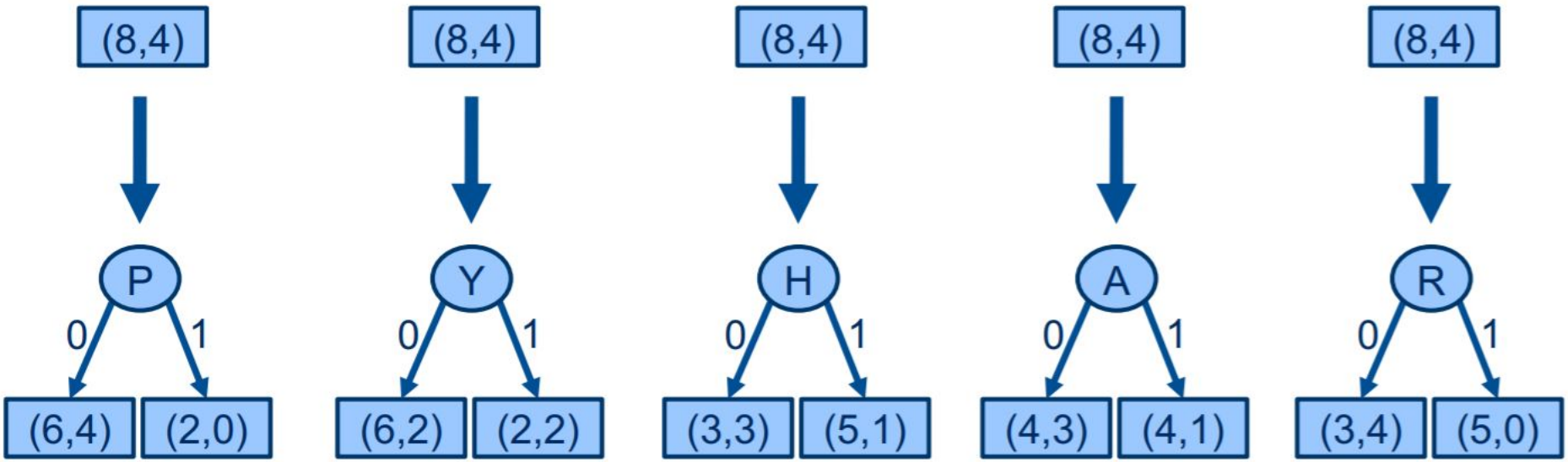


Which split increases purity the most?



Entropy of (N_0, N_1): $E = -p\log_2(p) - (1-p)\log_2(1-p)$ where $p = \dfrac{N_1}{N_0 + N_1}$

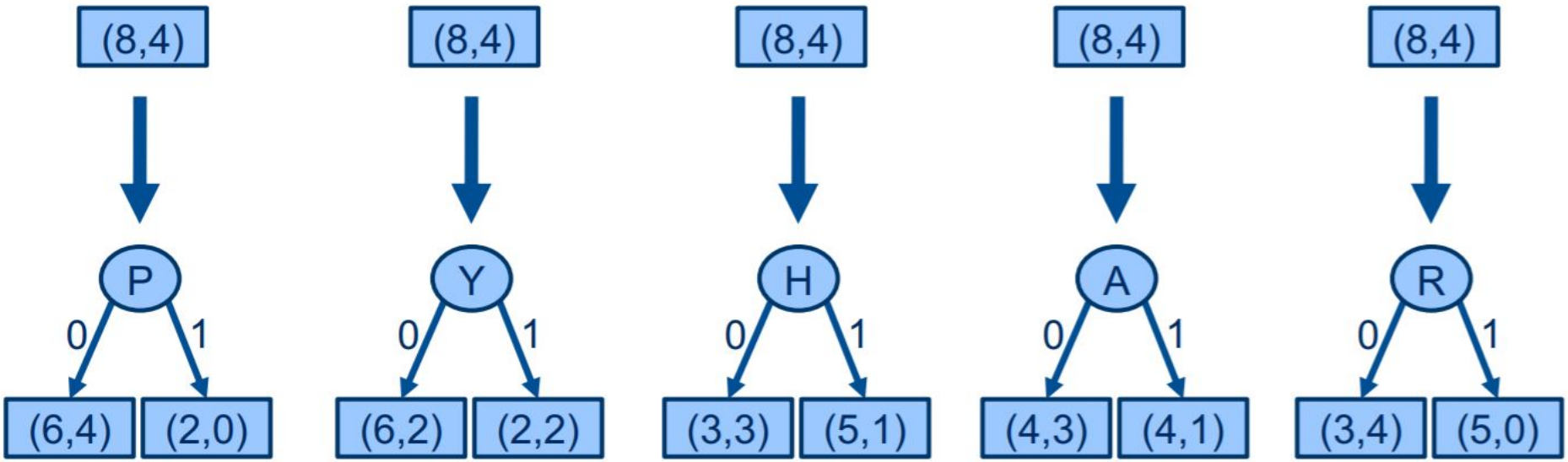
Which split increases purity the most?



p : 0.4 0 0.25 0.5 0.5 0.17 0.43 0.2 0.57 0

Entropy of (N_0, N_1): $E = -p\log_2(p) - (1-p)\log_2(1-p)$ where $p = \dfrac{N_1}{N_0 + N_1}$

Which split increases purity the most?

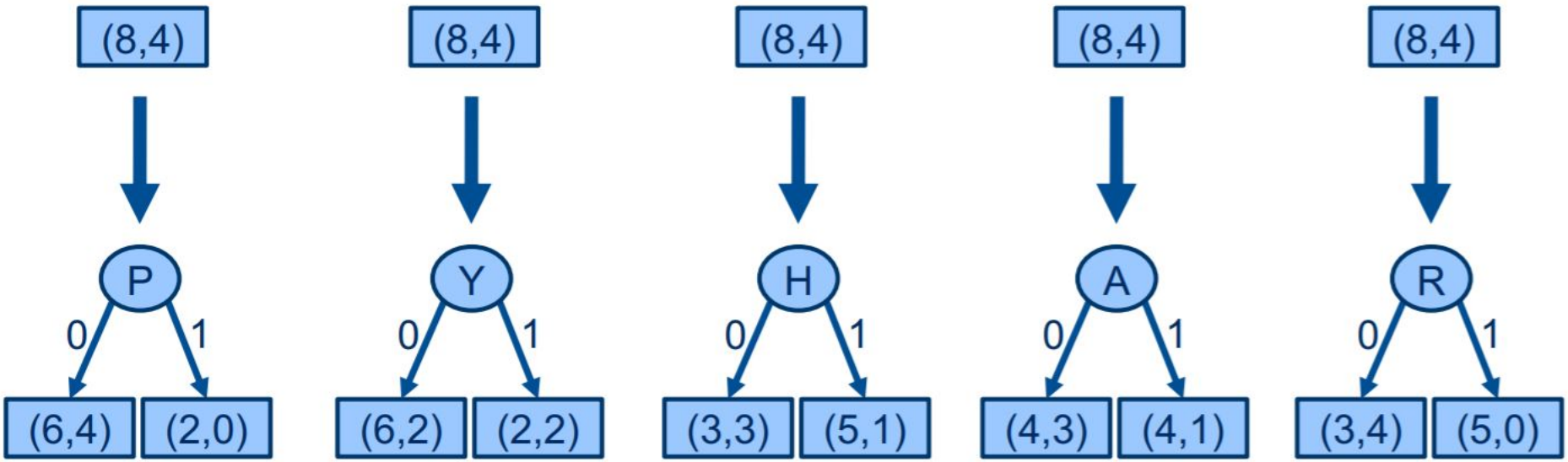


p : 0.4 0 | 0.25 0.5 | 0.5 0.17 | 0.43 0.2 | 0.57 0

E : 0.97 0 | 0.81 1 | 1 0.65 | 0.99 0.72 | 0.99 0

Entropy of (N_0, N_1): $E = -p\log_2(p) - (1-p)\log_2(1-p)$ where $p = \dfrac{N_1}{N_0 + N_1}$

Which split increases purity the most?



p :	0.4	0	0.25	0.5	0.5	0.17	0.43	0.2	0.57	0
E :	0.97	0	0.81	1	1	0.65	0.99	0.72	0.99	0
IG :	0.11		0.04		0.09		0.04		0.34	

Entropy of (N_0, N_1): $E = -p\log_2(p) - (1-p)\log_2(1-p)$ where $p = \dfrac{N_1}{N_0 + N_1}$

Information gain after split: $IG = E_{parent} - \dfrac{Size_{left}E_{left} + Size_{right}E_{right}}{Size_{left} + Size_{right}}$

Decision Tree Learning

Greedily recursively choosing best splits

Algorithm GrowDecisionTree(D, F) – grow a decision tree from training data.

Input : data D; set of features F.
Output : decision tree T with labelled leaves.

```
1 if Pure(D) then return Label(D);
2 S ←BestSplit-Class(D, F) ;                    // find best feature to split on
3 split D into subsets D_i according to the literals in S;
4 for each i do
5 |   if D_i ≠ ∅ then T_i ←GrowDecisionTree(D_i, F) ;
6 |   else T_i is a leaf labelled with Label(D);
7 end
8 return a tree whose root is labelled with S and whose children are T_i
```

Choosing the best split

- Which is the best feature to split on?
  - The one which minimizes the size of the tree
  - Cannot know the size without recursively building the tree (this would be computationally very hard)
- Intuitively, the splits which result in class distributions closer to purity potentially lead to smaller trees
- DT learners estimate how much purity improves (or how much impurity is reduced) after the considered split

Finding the best feature to split on

Algorithm BestSplit-Class(D, F) – find the best split for a decision tree.

Input : data D; set of features F.
Output : feature f to split on.

```
1  I_min ← 1;
2  for each f ∈ F do
3      split D into subsets D_1, ..., D_l according to the values v_j of f;
4      if Imp({D_1, ..., D_l}) < I_min ;          // if the split reduces impurity
5      then
6          I_min ← Imp({D_1, ..., D_l});
7          f_best ← f;
8      end
9  end
10 return f_best
```

Why these impurity measures?

- What is the intuition behind these measures?
- Guiding principles:
  - Maximize impurity gain:

$$Imp(D) - Imp(\{D_1, \ldots, D_l\})$$

    the overall impurity of child nodes should be maximally smaller than the impurity of the parent D
  - Impurity of child nodes should depend on individual impurities proportionally to their size:

$$Imp(\{D_1, \ldots, D_l\}) = \sum_{j=1}^{l} \frac{|D_j|}{|D|} Imp(D_j)$$

More guiding principles

- Impurity should be defined through the empirical class distribution in the node: in binary classification through $\dot{p}$:

  the proportion of positives, i.e. $\dot{p} = \dfrac{n^{\oplus}}{n^{\oplus} + n^{\ominus}}$

More guiding principles

- Impurity should be defined through the empirical class distribution in the node: in binary classification through $\dot{p}$:

  the proportion of positives, i.e. $\dot{p} = \frac{n^{\oplus}}{n^{\oplus} + n^{\ominus}}$
- Impurity should be 0 (full purity) if $\dot{p} = 0$ or $\dot{p} = 1$

More guiding principles

- Impurity should be defined through the empirical class distribution in the node: in binary classification through $\dot{p}$:

  the proportion of positives, i.e. $\dot{p} = \frac{n^{\oplus}}{n^{\oplus} + n^{\ominus}}$
- Impurity should be 0 (full purity) if $\dot{p} = 0$ or $\dot{p} = 1$
- Impurity should remain the same if we swap the classes, i.e. we replace $\dot{p}$ by $1 - \dot{p}$

More guiding principles

- Impurity should be defined through the empirical class distribution in the node: in binary classification through $\dot{p}$:

  the proportion of positives, i.e. $\dot{p} = \frac{n^{\oplus}}{n^{\oplus} + n^{\ominus}}$
- Impurity should be 0 (full purity) if $\dot{p} = 0$ or $\dot{p} = 1$
- Impurity should remain the same if we swap the classes, i.e. we replace $\dot{p}$ by $1 - \dot{p}$
- Impurity should be maximal if $\dot{p} = \frac{1}{2}$

More guiding principles

- Impurity should be defined through the empirical class distribution in the node: in binary classification through $\dot{p}$:

  the proportion of positives, i.e. $\dot{p} = \frac{n^{\oplus}}{n^{\oplus} + n^{\ominus}}$
- Impurity should be 0 (full purity) if $\dot{p} = 0$ or $\dot{p} = 1$
- Impurity should remain the same if we swap the classes, i.e. we replace $\dot{p}$ by $1 - \dot{p}$
- Impurity should be maximal if $\dot{p} = \frac{1}{2}$
- These are among the well known impurity measures:
  - Minority Class
  - Gini Index
  - Entropy

Minority class

- Minority class: $min(\dot{p}, 1 - \dot{p})$
- Measures the proportion of misclassified instances if the leaf was labelled with the majority class
- Equivalently written as $\frac{1}{2} - |\dot{p} - \frac{1}{2}|$

Gini Index

- Gini index: $2\dot{p}(1-\dot{p})$
- Measures the expected proportion of misclassified instances if the leaf was labelled randomly: positive with probability $\dot{p}$ and negative with probability $1-\dot{p}$
- The probability of a false positive is then $\dot{p}(1-\dot{p})$ and the probability of a false negative is $(1-\dot{p})\dot{p}$
- In case of multiclass classification (there are C classes) the formula is

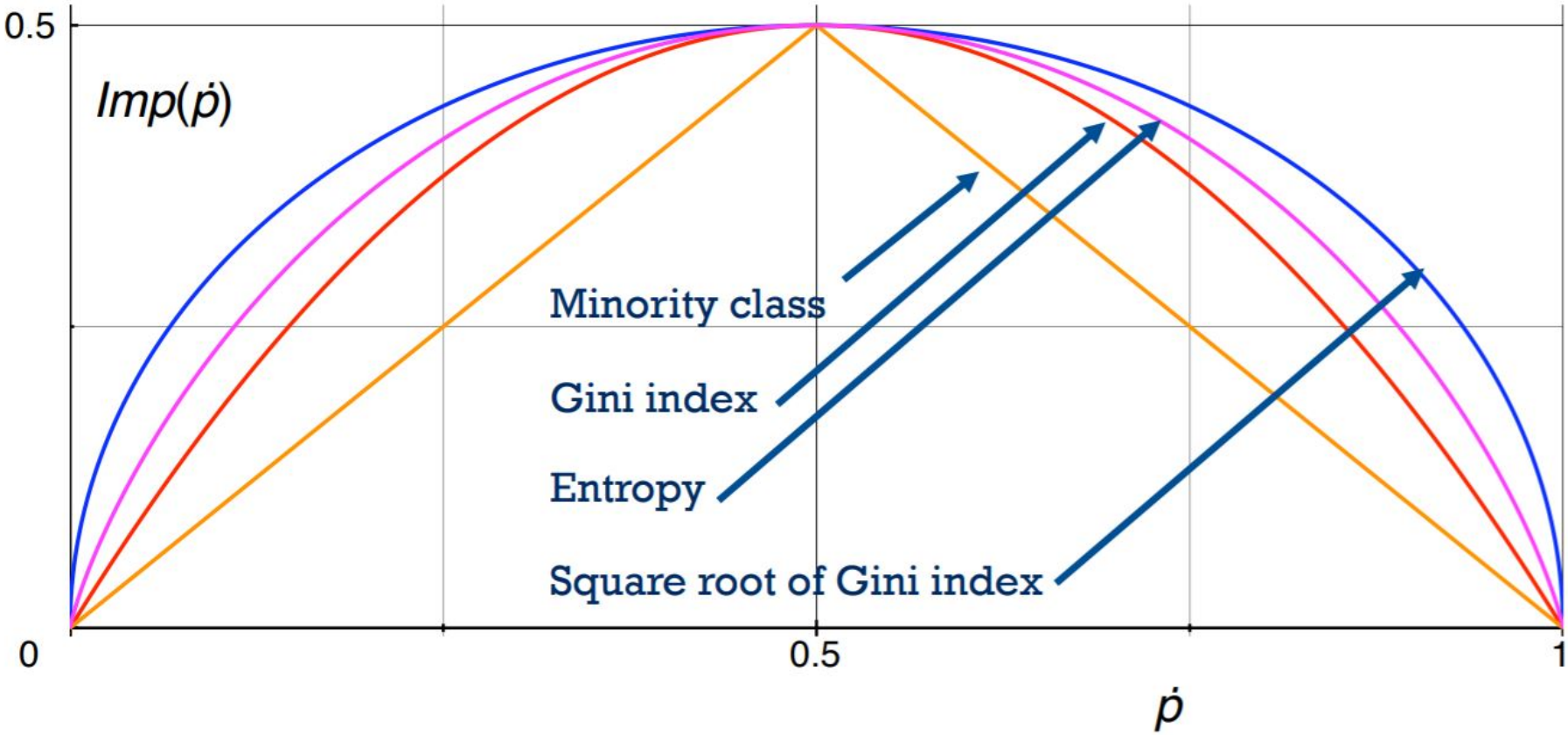
$$Gini = 1 - \sum_{i=1}^{C} p_i^2$$

Entropy

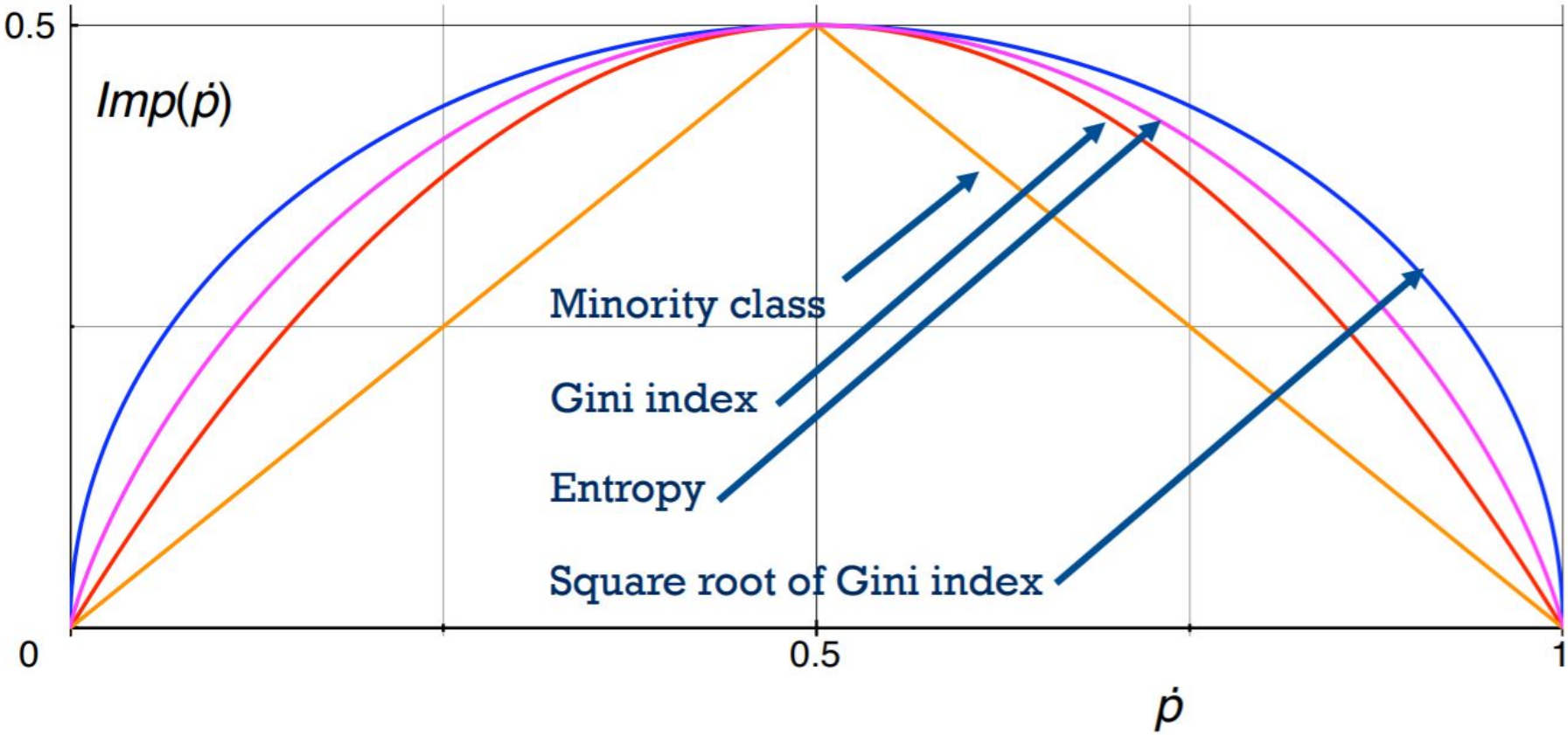
- Entropy: $-\dot{p}\log_2\dot{p} - (1-\dot{p})\log_2(1-\dot{p})$
- Measures the expected information, in bits, conveyed by somebody telling you the class of a randomly drawn example
- The purer the set of examples, the more predictable this message becomes and the smaller the expected information
- In case of multiclass classification (there are C classes) the formula is

$$Entropy = -\sum_{i=1}^{C} p_i \cdot \log_2 p_i$$

Impurity Functions

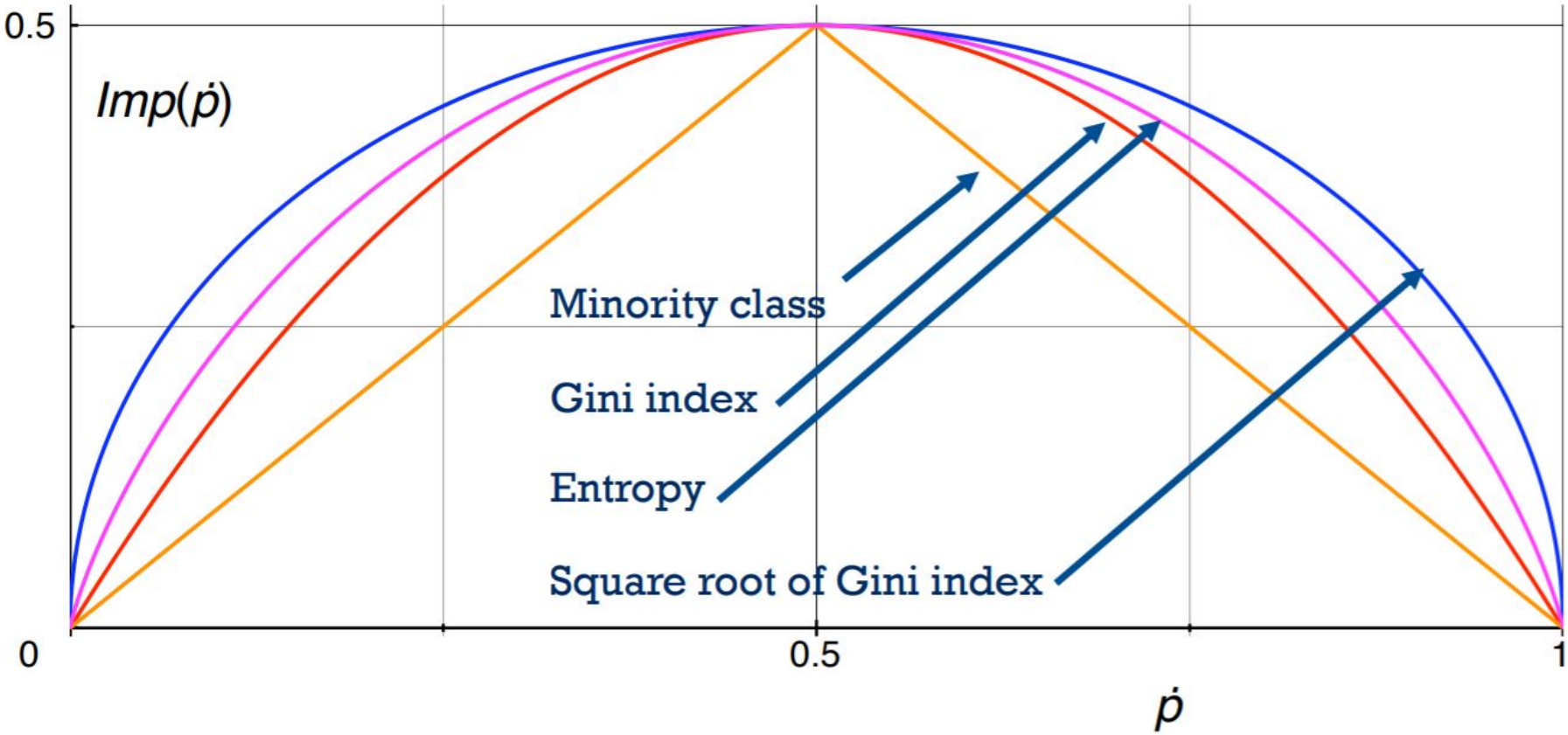


Impurity Functions



What is the best measure?

Impurity Functions



What is the best measure? Depends on the data!

Decision tree learning algorithm

ID3: Iterative Dichotomizer 3

1. Start with a single node
2. Find the feature with the largest information gain
3. Split the node according to this feature
4. Repeat recursively on subnodes

ID3 with categorical non-binary features

- Apply ID3 as usual
- When considering to split using a categorical feature with more than 2 values (non-binary):
  1. Split into k subtrees (instead of just left and right)
  2. Measure entropy in each subtree
  3. Calculate information gain (average entropy per instance after splitting vs before splitting)

ID3 for multi-class classification

- Multi-class classification: Label is categorical, with more than 2 values (classes)
- Apply ID3 as usual
- Formula for entropy for a node with counts $(N_1, N_2, \ldots, N_k)$:

$$E = -\sum_{i=1}^{k} p_i \log_2(p_i),$$

where $p_i = \dfrac{N_i}{N_1 + N_2 + \ldots + N_k}$

C4.5 - extension of ID3

C4.5

- Supports continuous attributes
  Considers all possible thresholds for splitting, for example $age < 18$; $weight < 100$; $height > 1.95$
- Supports missing values
- Supports pruning
  Pruning of a subtree means cancelling all its splits and keeping it as one leaf node

There is also C5 algorithm with some improvements in speed and memory usage.

Pre-pruning

- Pre-pruning means that during decision tree learning some nodes are not split further
- Decision not to split is typically made if one of the following conditions hold:
  - Node size is below a threshold
  - One of the child nodes would have size below a threshold
  - Depth of the tree is above a threshold
  - Impurity gain is below a threshold

Post-pruning

- Post-pruning means that after the tree has been learned, some of the subtrees are removed (replaced by their parent node)
- Decision to prune usually done based on tree performance on a hold-out dataset
- Reduced-error pruning removes subtrees which perform worse than majority class (within the respective parent node)

Pre- or Post-pruning?

- Generally good to do both
- Pre-pruning is faster and does not need any extra data
- Post-pruning is slower, needs extra data, but can "undo" some of the (potentially) bad greedy decisions made at the end of decision tree learning

How to overcome NFL?

- Make assumptions about the domain!

How to overcome NFL?

- Make assumptions about the domain!
- We already know that different models make different assumptions

How to overcome NFL?

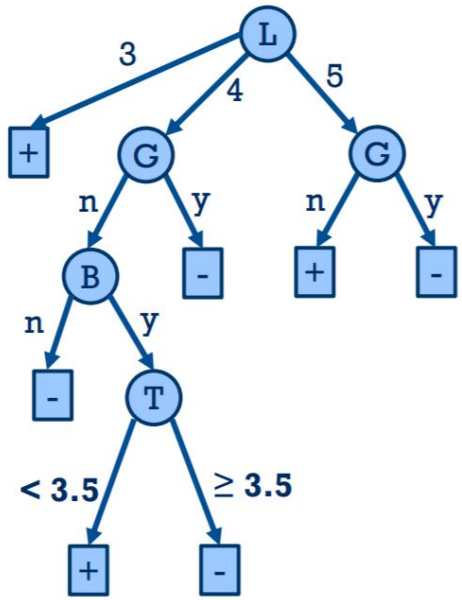
- Make assumptions about the domain!
- We already know that different models make different assumptions
- What do decision trees assume?

How to overcome NFL?

- Make assumptions about the domain!
- We already know that different models make different assumptions
- What do decision trees assume?
- Decision trees assume that the instance space can be split into **segments** such that all (or at least majority of) instances in the segment belong to the same class

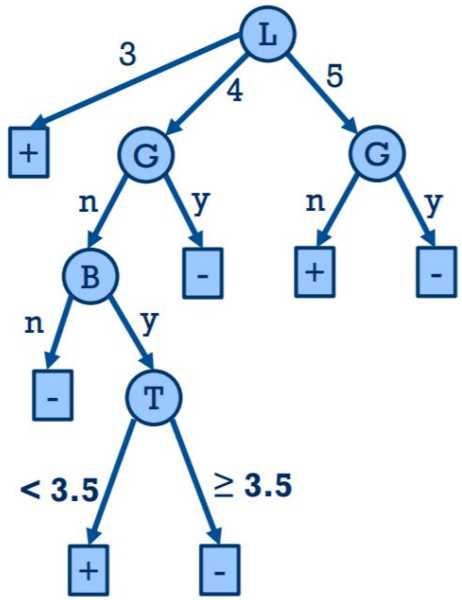
Segments in decision trees

- What is a segment in decision trees?



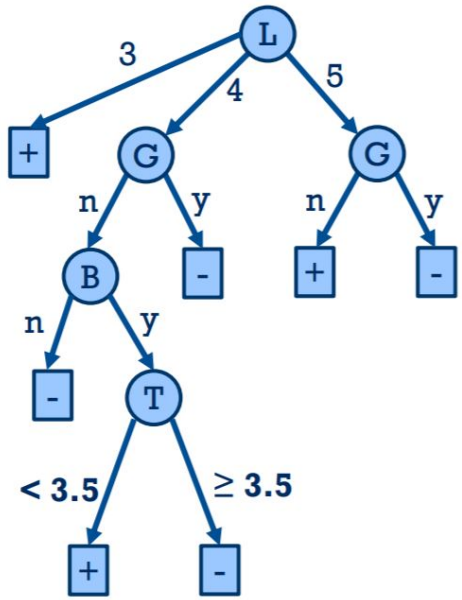
Segments in decision trees

- What is a segment in decision trees?
- Segment:



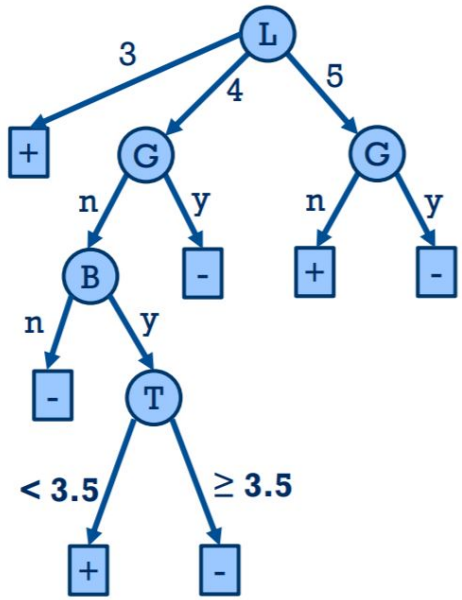
Segments in decision trees

- What is a segment in decision trees?
- Segment:
  - Corresponds to a leaf in the decision tree



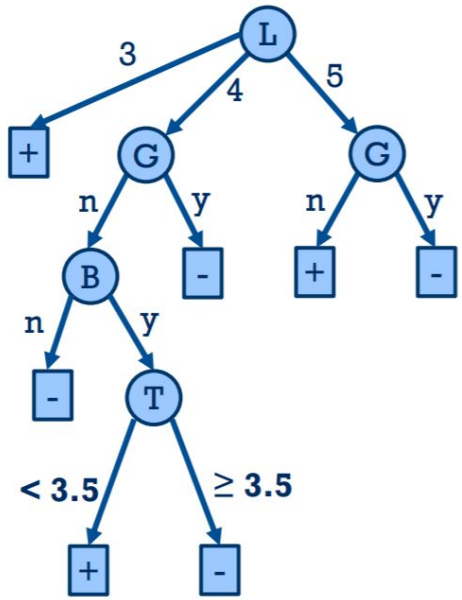
Segments in decision trees

- What is a segment in decision trees?
- Segment:
  - Corresponds to a leaf in the decision tree
  - A logical conjunction ($\&$) of terms



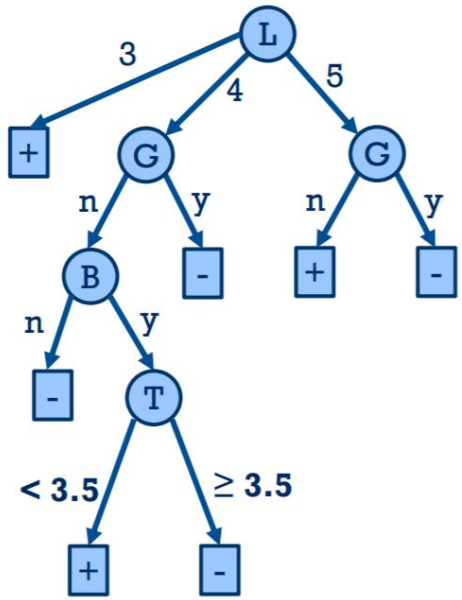
Segments in decision trees

- What is a segment in decision trees?
- Segment:
  - Corresponds to a leaf in the decision tree
  - A logical conjunction ($\&$) of terms
  - Each term compares a feature to a constant



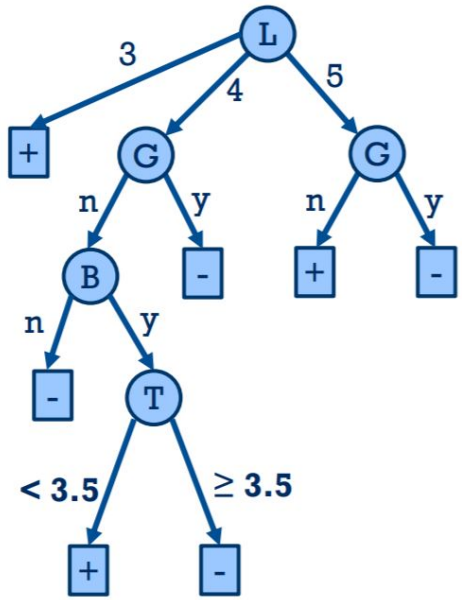
Segments in decision trees

- What is a segment in decision trees?
- Segment:
  - Corresponds to a leaf in the decision tree
  - A logical conjunction ($\&$) of terms
  - Each term compares a feature to a constant
  - Allowed comparison operators: $=, \neq, <, \leq, >, \geq$



Segments in decision trees

- What is a segment in decision trees?
- Segment:
  - Corresponds to a leaf in the decision tree
  - A logical conjunction ($\&$) of terms
  - Each term compares a feature to a constant
  - Allowed comparison operators: $=, \neq, <, \leq, >, \geq$
- Example: $(L = 4)\&(G = y)$



Why does a decision tree "work"?

- Why does a decision tree usually predict better than random?

Why does a decision tree "work"?

- Why does a decision tree usually predict better than random?
- Because real-life tasks have usually a structure that favours decision tree learning

Why does a decision tree "work"?

- Why does a decision tree usually predict better than random?
- Because real-life tasks have usually a structure that favours decision tree learning
- Decision trees have a practically useful inductive bias (set of assumptions about the data) like the other popular learning algorithms

Choosing hyper-parameter values

- As we know already, in practice we hold out part of our training set for testing purposes (**test set**)

Choosing hyper-parameter values

- As we know already, in practice we hold out part of our training set for testing purposes (**test set**)
- Evaluating different values of a hyper-parameter and choosing the optimal one on the test set, there is still a high chance of overfitting

Choosing hyper-parameter values

- As we know already, in practice we hold out part of our training set for testing purposes (**test set**)
- Evaluating different values of a hyper-parameter and choosing the optimal one on the test set, there is still a high chance of overfitting
- To solve the latter we may hold out another part of our training set as a **validation set**

Choosing hyper-parameter values

- As we know already, in practice we hold out part of our training set for testing purposes (**test set**)
- Evaluating different values of a hyper-parameter and choosing the optimal one on the test set, there is still a high chance of overfitting
- To solve the latter we may hold out another part of our training set as a **validation set**

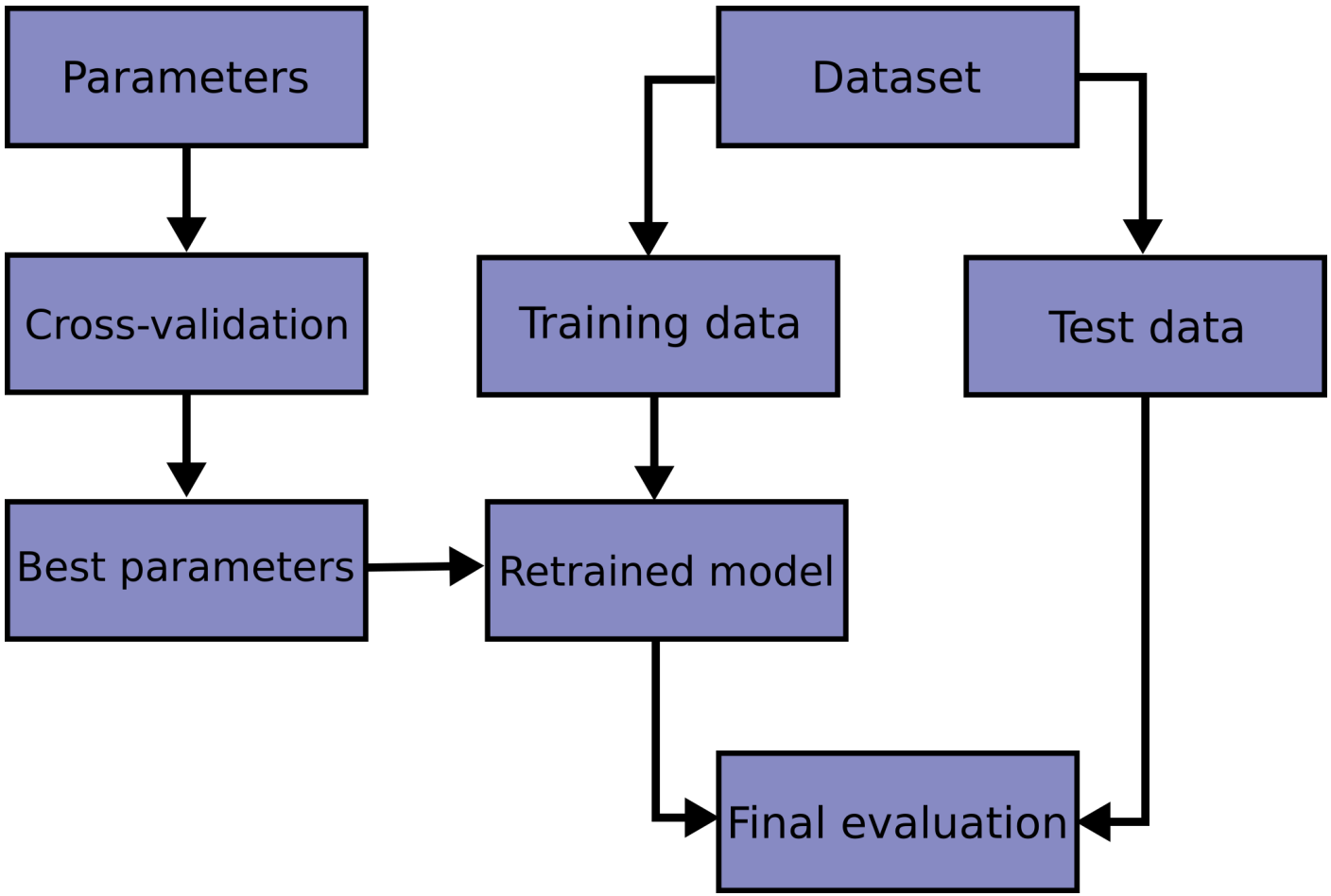
- Partitioning the available data into three sets reduces the number of samples which can be used for learning the model

Choosing hyper-parameter values

- As we know already, in practice we hold out part of our training set for testing purposes (**test set**)
- Evaluating different values of a hyper-parameter and choosing the optimal one on the test set, there is still a high chance of overfitting
- To solve the latter we may hold out another part of our training set as a **validation set**

- Partitioning the available data into three sets reduces the number of samples which can be used for learning the model
- The results can depend on a particular random choice for the pair of (train, validation) sets

Choosing hyper-parameter values



K-Fold Cross-Validation

- Test set is still hold out for final evaluation

K-Fold Cross-Validation

- Test set is still hold out for final evaluation
- The training set is split into k smaller sets (folds)

K-Fold Cross-Validation

- Test set is still hold out for final evaluation
- The training set is split into k smaller sets (folds)
- For each k:

K-Fold Cross-Validation

- Test set is still hold out for final evaluation
- The training set is split into k smaller sets (folds)
- For each k:
  - A model is trained using $k-1$ of the folds as training data

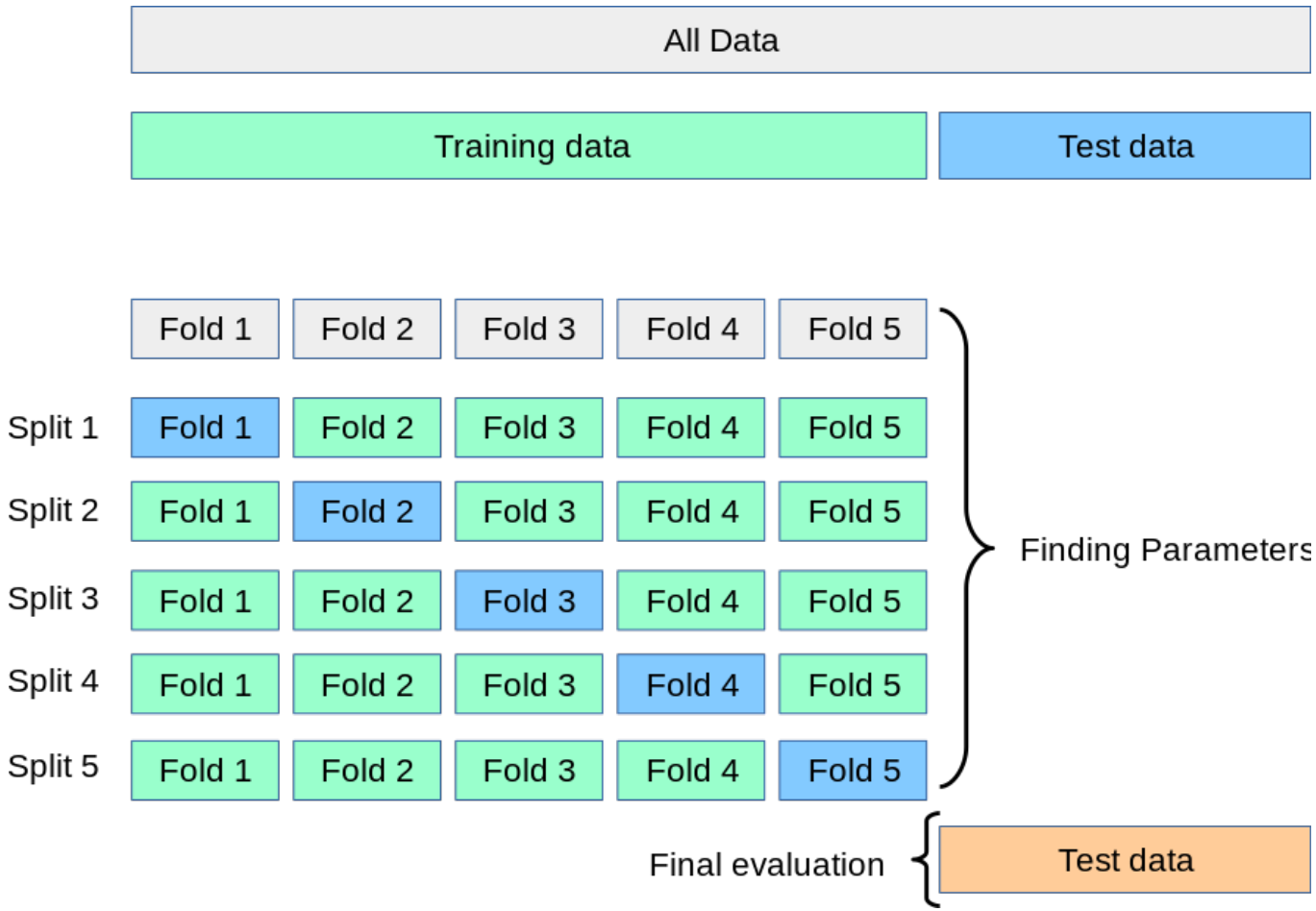
K-Fold Cross-Validation

- Test set is still hold out for final evaluation
- The training set is split into k smaller sets (folds)
- For each k:
  - A model is trained using $k-1$ of the folds as training data
  - the resulting model is validated on the remaining part of the data

K-Fold Cross-Validation

- Test set is still hold out for final evaluation
- The training set is split into k smaller sets (folds)
- For each k:
  - A model is trained using $k-1$ of the folds as training data
  - the resulting model is validated on the remaining part of the data
- Average over the k evaluations is considered as a result

K-Fold Cross-Validation



What have we learned today?

- ✓ Kernel Methods
- ✓ No Free Lunch Theorem
- ✓ Decision trees
- ✓ Choosing the best split
- ✓ Impurity measures
- ✓ Decision Tree Learners
- ✓ Pruning
- ✓ Cross-Validation