Introduction to Machine Learning (for AI)

Supervised Learning II - Non-Linear Models

Dr. Matias Valdenegro, Dr. Andreea Sburlea

November 21, 2024

Today's Lecture

- Today we cover some non-linear methods, mainly decision trees, random forests, and nearest neighbor methods.
- Finally, we cover ensemble methods, that is, combining multiple models which almost always improves predictions.

Outline

Non-Linear Classification - Kernels

There is another advantage commonly associated with SVMs, that they can also perform non-linear classification using what is called a **kernel**.

Non-Linear Classification - Kernels

There is another advantage commonly associated with SVMs, that they can also perform non-linear classification using what is called a **kernel**.

The idea of the Kernel is that it can transform the SVM from a linear classifier, into a non-linear classifier. This is done by **changing the shape of the decision boundary**, and thus producing a non-linear margin.

Non-Linear Classification - Kernels

There is another advantage commonly associated with SVMs, that they can also perform non-linear classification using what is called a **kernel**.

The idea of the Kernel is that it can transform the SVM from a linear classifier, into a non-linear classifier. This is done by **changing the shape of the decision boundary**, and thus producing a non-linear margin.

The *kernel trick* is applied for this purpose, the formulation of the SVM does not change, only a small part is replaced with a kernel function. We will see this with a concrete example.

Kernel - Example

Let's assume that we have two vectors $\phi(x)$ and $\phi(y)$ given by:

$$\phi(x) = (x_d^2, ..., x_1^2, \sqrt{2}x_d x_{d-1}, ..., \sqrt{2}x_{d-1}x_{d-2}, ..., \quad (1)$$

$$\sqrt{2}x_{d-1}x_1, ..., \sqrt{2}x_2 x_1, ..., \sqrt{2}cx_d, ..., \sqrt{2}cx_1, c) \quad (2)$$

Kernel - Example

Let's assume that we have two vectors $\phi(x)$ and $\phi(y)$ given by:

$$\phi(x) = (x_d^2, ..., x_1^2, \sqrt{2}x_d x_{d-1}, ..., \sqrt{2}x_{d-1}x_{d-2}, ..., \tag{1}$$

$$\sqrt{2}x_{d-1}x_1, ..., \sqrt{2}x_2 x_1, ..., \sqrt{2}cx_d, ..., \sqrt{2}cx_1, c) \tag{2}$$

Then we compute the following dot product (inner product):

$$\phi(x)^\top \cdot \phi(y) = \sum_{i=1}^{d}(x_i^2)(y_i^2) + \sum_{i=2}^{d}\sum_{j=1}^{i-1}(\sqrt{2}x_i x_j)(\sqrt{2}y_i y_j)^\top \tag{3}$$

$$+ \sum_{i=1}^{d}(\sqrt{2}cx_i)(\sqrt{2}cy_i) + c^2 \tag{4}$$

Kernel - Example

This can be simplified with algebra to the following:

$$K(\mathbf{x}, \mathbf{y}) = \phi(x)^\top \cdot \phi(y) = \left(\sum_{i=1}^{d} x_i y_i + c \right)^2 \tag{5}$$

Kernel - Example

This can be simplified with algebra to the following:

$$K(\mathbf{x}, \mathbf{y}) = \phi(x)^\top \cdot \phi(y) = \left(\sum_{i=1}^{d} x_i y_i + c \right)^2 \tag{5}$$

This means that the dot product $\phi(x)^\top \cdot \phi(y)$ for $\phi(x)$ has a closed form that is easily computable.

Kernel - Example

This can be simplified with algebra to the following:

$$K(\mathbf{x}, \mathbf{y}) = \phi(x)^\top \cdot \phi(y) = \left(\sum_{i=1}^{d} x_i y_i + c \right)^2 \tag{5}$$

This means that the dot product $\phi(x)^\top \cdot \phi(y)$ for $\phi(x)$ has a closed form that is easily computable.
The essence of the Kernel Trick is to use an easily computable function that corresponds to a dot product of the features in a highly dimensional space.

Kernel - Example

Now, if we go back to the general form of a linear model:

$$f(x) = \underbrace{\mathbf{x} \cdot \mathbf{w}}_{\text{Dot Product}} + b \tag{6}$$

There is a dot product between the input features and the weight vector.

Kernel - Example

Now, if we go back to the general form of a linear model:

$$f(x) = \underbrace{\mathbf{x} \cdot \mathbf{w}}_{\text{Dot Product}} + b \tag{6}$$

There is a dot product between the input features and the weight vector. In a *Kernelized* linear model, we can do the following:

$$f(x) = \phi(\mathbf{w}) \cdot \phi(\mathbf{x}) + b \tag{7}$$

Kernel - Example

Now, if we go back to the general form of a linear model:

$$f(x) = \underbrace{\mathbf{x} \cdot \mathbf{w}}_{\text{Dot Product}} + b \tag{6}$$

There is a dot product between the input features and the weight vector. In a *Kernelized* linear model, we can do the following:

$$f(x) = \phi(\mathbf{w}) \cdot \phi(\mathbf{x}) + b \tag{7}$$

For the vectors we previously defined, we can replace the dot product $\phi(\mathbf{w})\phi(\mathbf{x})$ with the closed form solution:

$$f(x) = \left(\sum_{i=1}^{n} x_i w_i + c \right)^2 + b \tag{8}$$

Kernel - Example

This process is called Kernelization or the Kernel Trick. And it works because in Kernel space, this is still a linear model.

Kernel - Example

This process is called Kernelization or the Kernel Trick. And it works because in Kernel space, this is still a linear model.

What this means is that the model is now non-linear in w space, but it is still linear in $\phi(w)$ space.

Kernel - Example

This process is called Kernelization or the Kernel Trick. And it works because in Kernel space, this is still a linear model.

What this means is that the model is now non-linear in w space, but it is still linear in $\phi(w)$ space.

The Kernel Trick is a bit difficult to understand in an abstract way, but it is widely used as a way to make a linear classifier non-linear.

Kernel - Example

This process is called Kernelization or the Kernel Trick. And it works because in Kernel space, this is still a linear model.

What this means is that the model is now non-linear in w space, but it is still linear in $\phi(w)$ space.

The Kernel Trick is a bit difficult to understand in an abstract way, but it is widely used as a way to make a linear classifier non-linear.

The trick for understanding is that the kernel projects the data into a highly dimensional space ($3n$ for the example here), and it is more likely to be linearly separable in this new space.

Kernel Trick - Kernel Definition

A Kernel function is defined as:

$$K(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y}) \tag{9}$$

For some function $\phi(x)$.

For the kernel trick, we are only interested in the function $K(x, y)$.

Kernel Trick - Kernel Definition

A Kernel function is defined as:

$$K(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y}) \tag{9}$$

For some function $\phi(x)$.
For the kernel trick, we are only interested in the function $K(x, y)$.

The Kernel Trick is to replace every dot product in the linear model by $K(x, y)$.

Kernel Trick - Kernel Definition

A Kernel function is defined as:

$$K(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y}) \tag{9}$$

For some function $\phi(x)$.
For the kernel trick, we are only interested in the function $K(x, y)$.

The Kernel Trick is to replace every dot product in the linear model by $K(x, y)$.

This works because the data is projected/transformed into a higher dimensional space (where it is more likely to be linearly separable), but this is done *without* having to compute the feature transformation, only the dot product with the Kernel function.

Kernel Trick

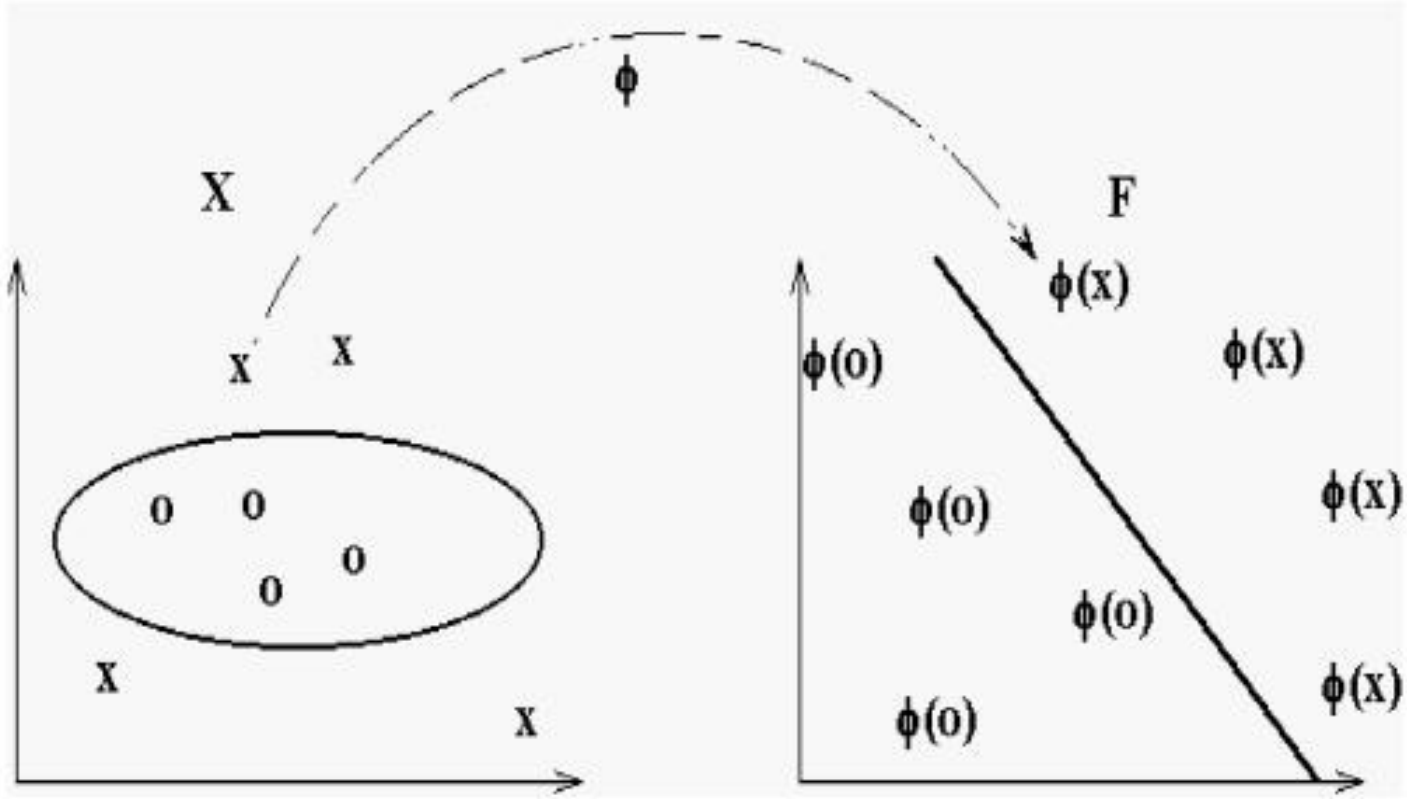


Figure from `https://en.wikipedia.org/wiki/Polynomial_kernel#/media/File:Svm_8_polinomial.JPG`

Kernel Trick

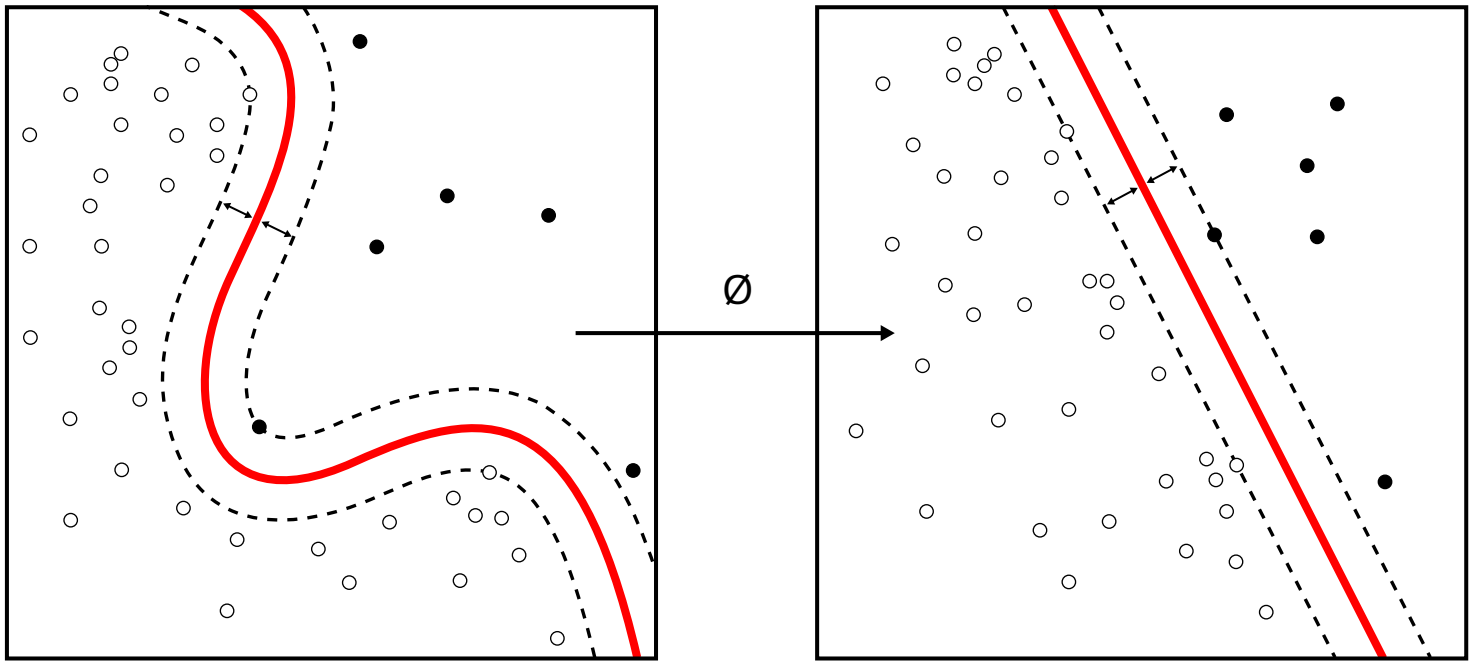


Figure: Left is the original feature space, and on the right is the kernel transformed feature space, where the features are more likely to be linearly separable.

Figure from https://en.wikipedia.org/wiki/Support_vector_machine#/media/File:Kernel_Machine.svg

Kernel Trick

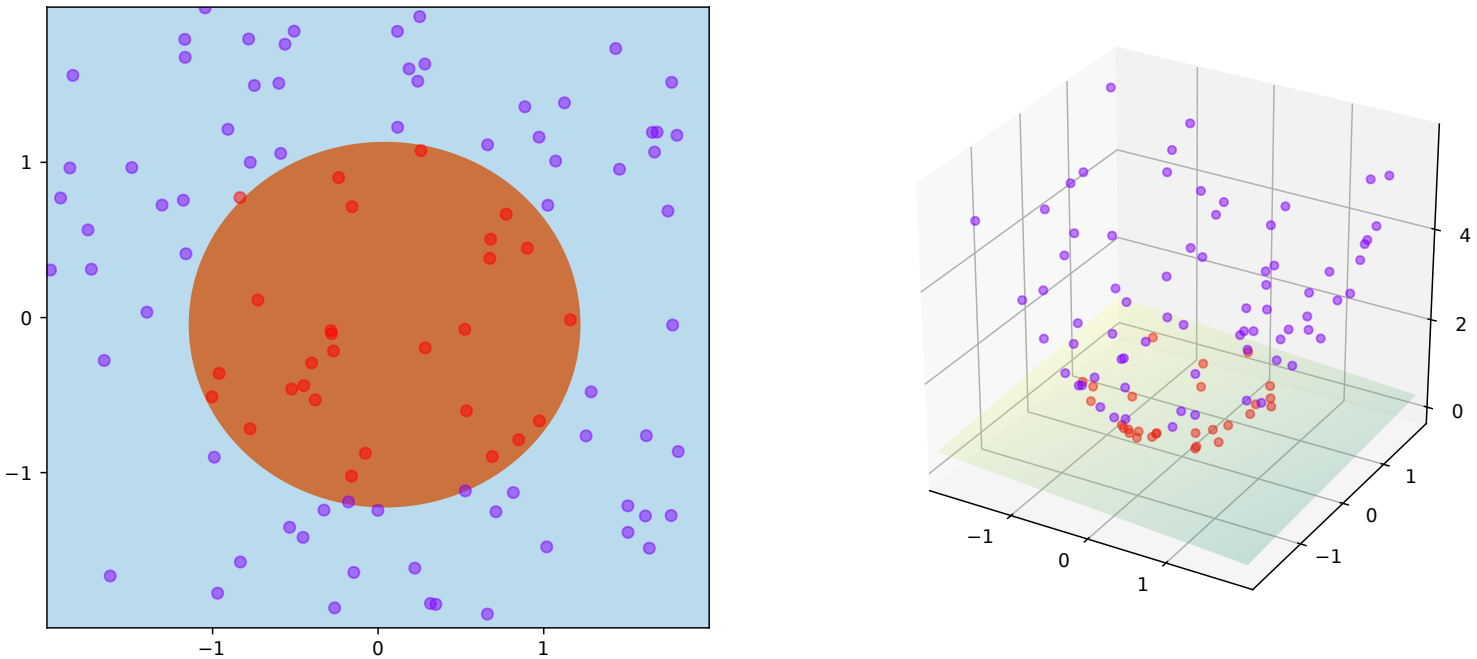


Figure: Left is the data in the original space, while on the right the data has been transformed into a 3D space, where it is linearly separable, unlike in the original space.

Kernel Functions

Polynomial Kernel

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + c)^d \tag{10}$$

Gaussian Kernel or Radial Basis Function (RBF)

$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{(||\mathbf{x} - \mathbf{y}||)^2}{2\sigma^2}\right) \tag{11}$$

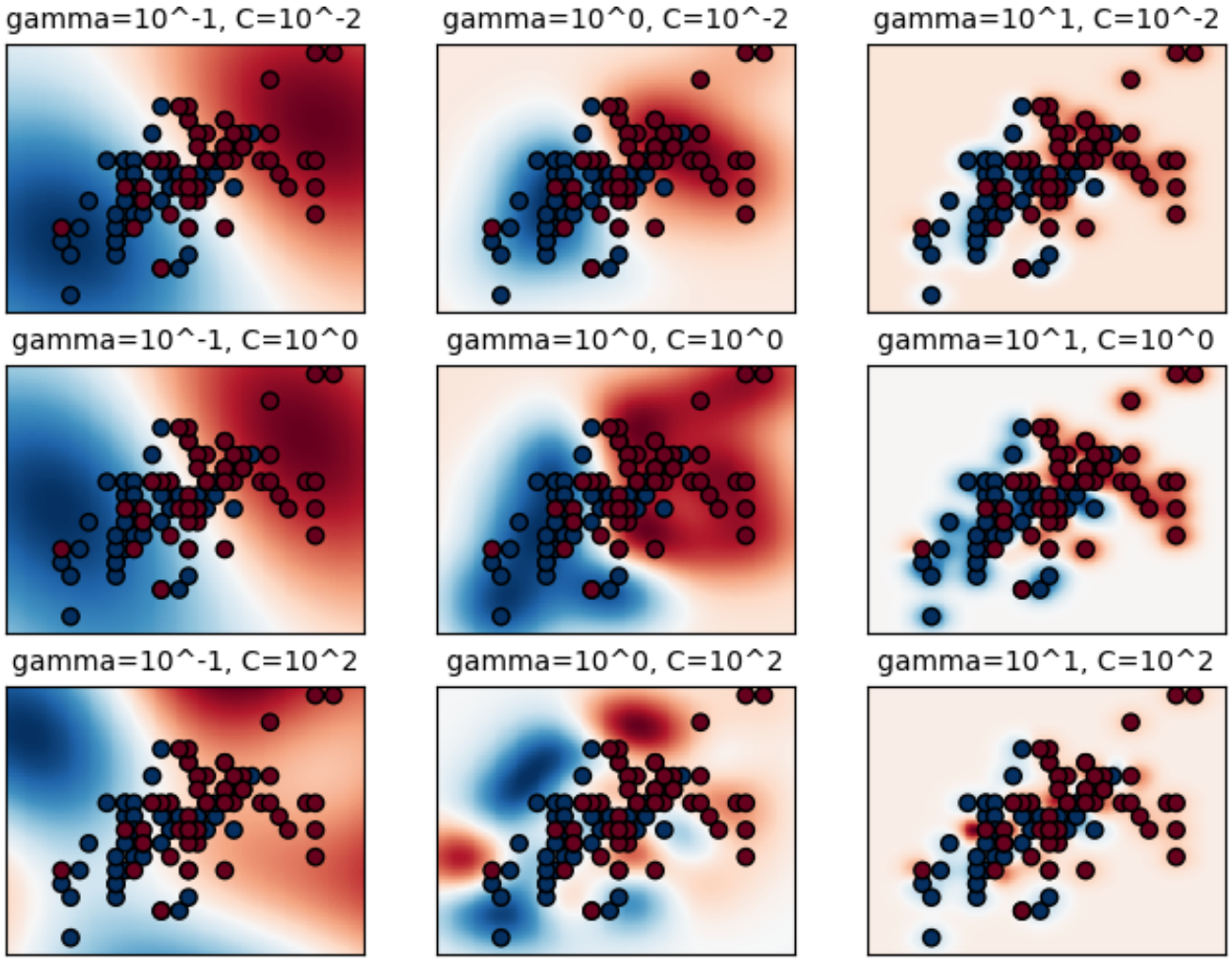
Laplace RBF

$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{||\mathbf{x} - \mathbf{y}||}{2\sigma^2}\right) \tag{12}$$

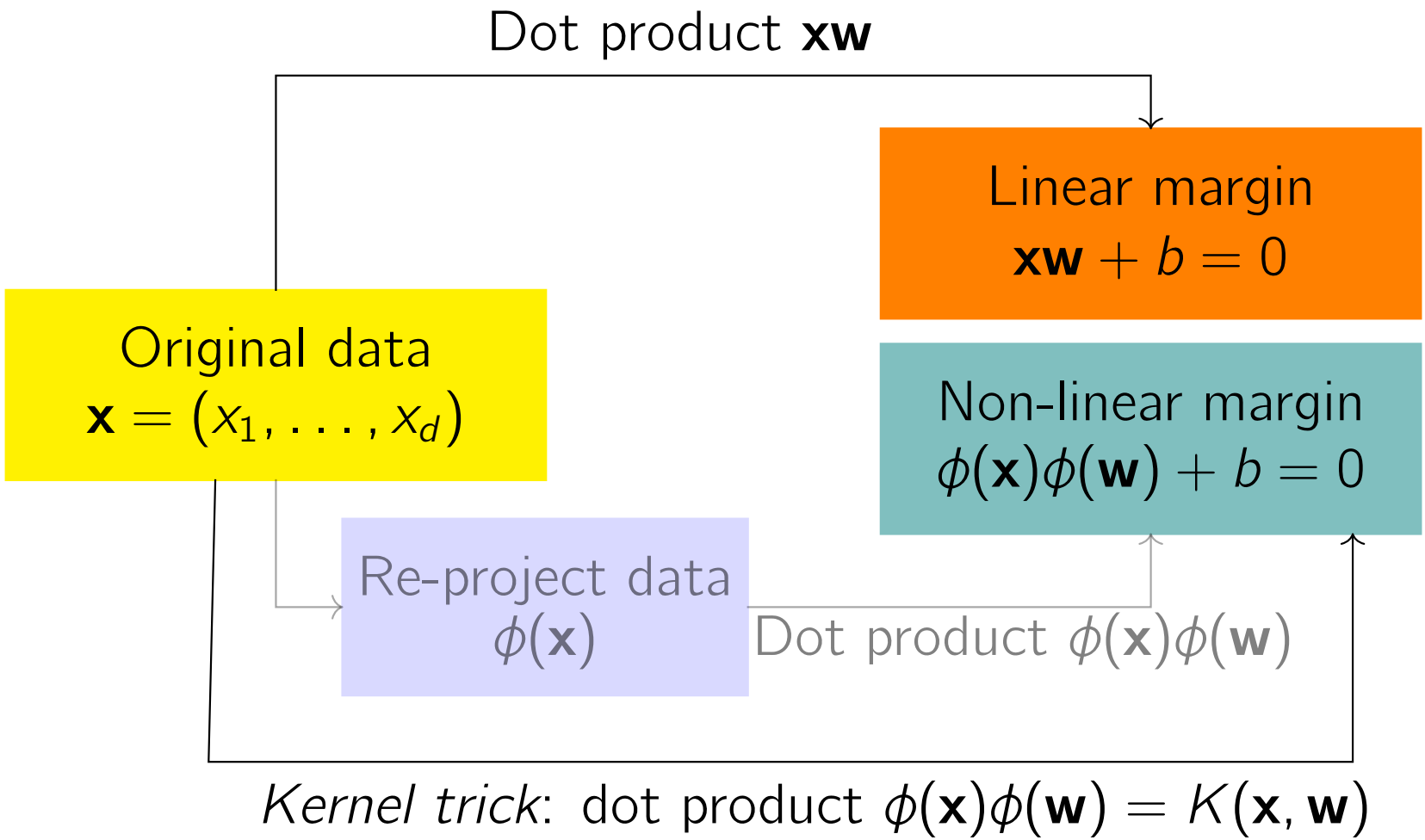
Hyperbolic Kernel

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\alpha \mathbf{x} \cdot \mathbf{y} + c) \tag{13}$$

Parameters of RBF Kernel



SVM - Scheme


Dot product **xw**
Linear margin
$\mathbf{xw} + b = 0$
Original data
$\mathbf{x} = (x_1, \ldots, x_d)$
Non-linear margin
$\phi(\mathbf{x})\phi(\mathbf{w}) + b = 0$
Re-project data
$\phi(\mathbf{x})$
Dot product $\phi(\mathbf{x})\phi(\mathbf{w})$
Kernel trick: dot product $\phi(\mathbf{x})\phi(\mathbf{w}) = K(\mathbf{x}, \mathbf{w})$

Outline

Measuring distances

Euclidean Distance

It's the distance you are familiar with, as intuitively measured with a ruler (also known as the *l*2 norm):

$$\text{2-norm distance} = \left(\sum_{i=1}^{n} |x_i - y_i|^2 \right)^{1/2} \tag{14}$$

Measuring distances

Euclidean Distance

It's the distance you are familiar with, as intuitively measured with a ruler (also known as the $l2$ norm):

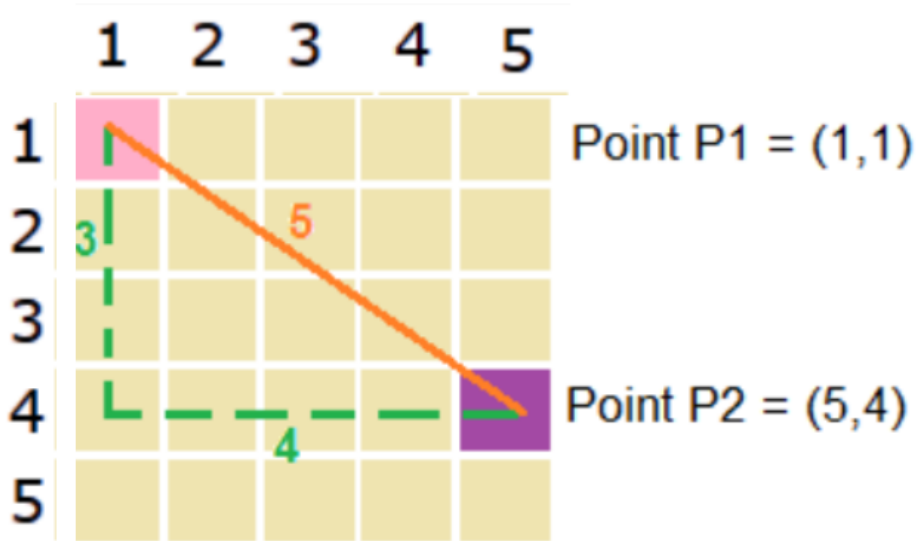
$$\text{2-norm distance} = \left(\sum_{i=1}^{n} |x_i - y_i|^2 \right)^{1/2} \tag{14}$$

Manhattan Distance

It measures the distance between two points in a city if you can only travel along orthogonal city blocks (also known as the $l1$ norm):

$$\text{1-norm distance} = \sum_{i=1}^{n} |x_i - y_i| \tag{15}$$

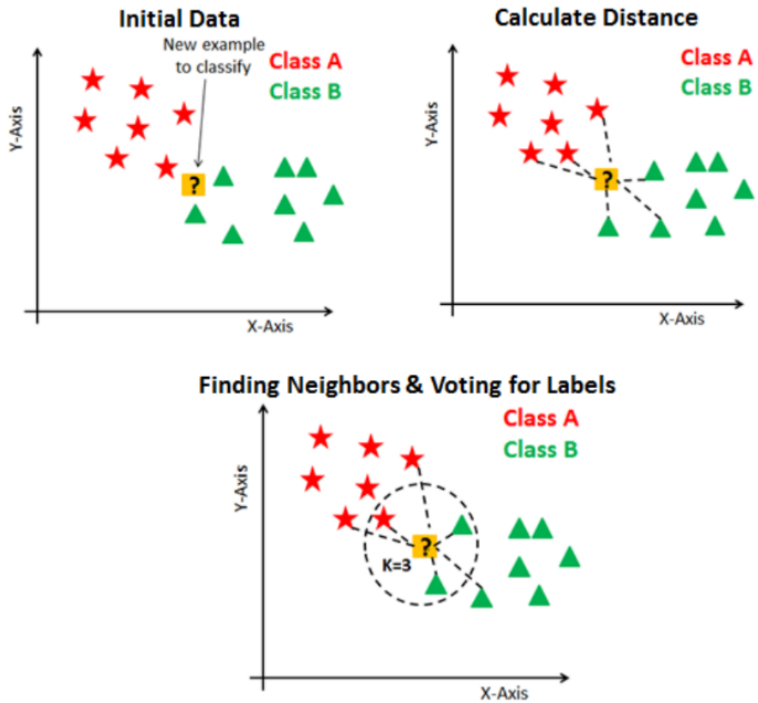
Measuring distances - Example



Euclidean distance $= \sqrt{(5-1)^2 + (4-1)^2} = 5$

Manhattan distance $= |5-1| + |4-1| = 7$

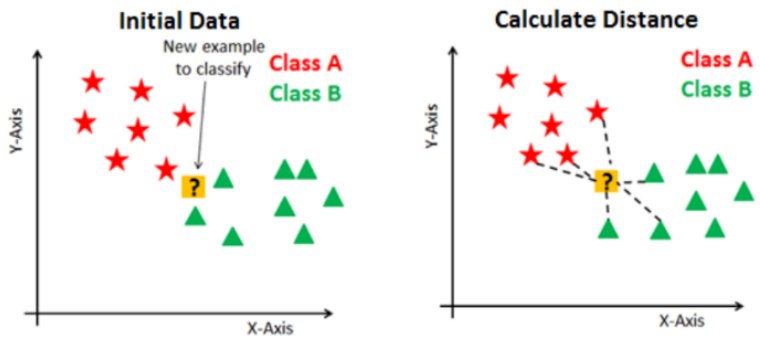
K-Nearest Neighbors



KNN is a non-parametric and lazy learning algorithm

- no assumption for underlying data distribution (the model structure is determined from the dataset)
- does not need any training data points for model generation (all training data will be used in the testing phase)
- KNN needs more time to scan all data points and scanning all data points will require more memory for storing training data.

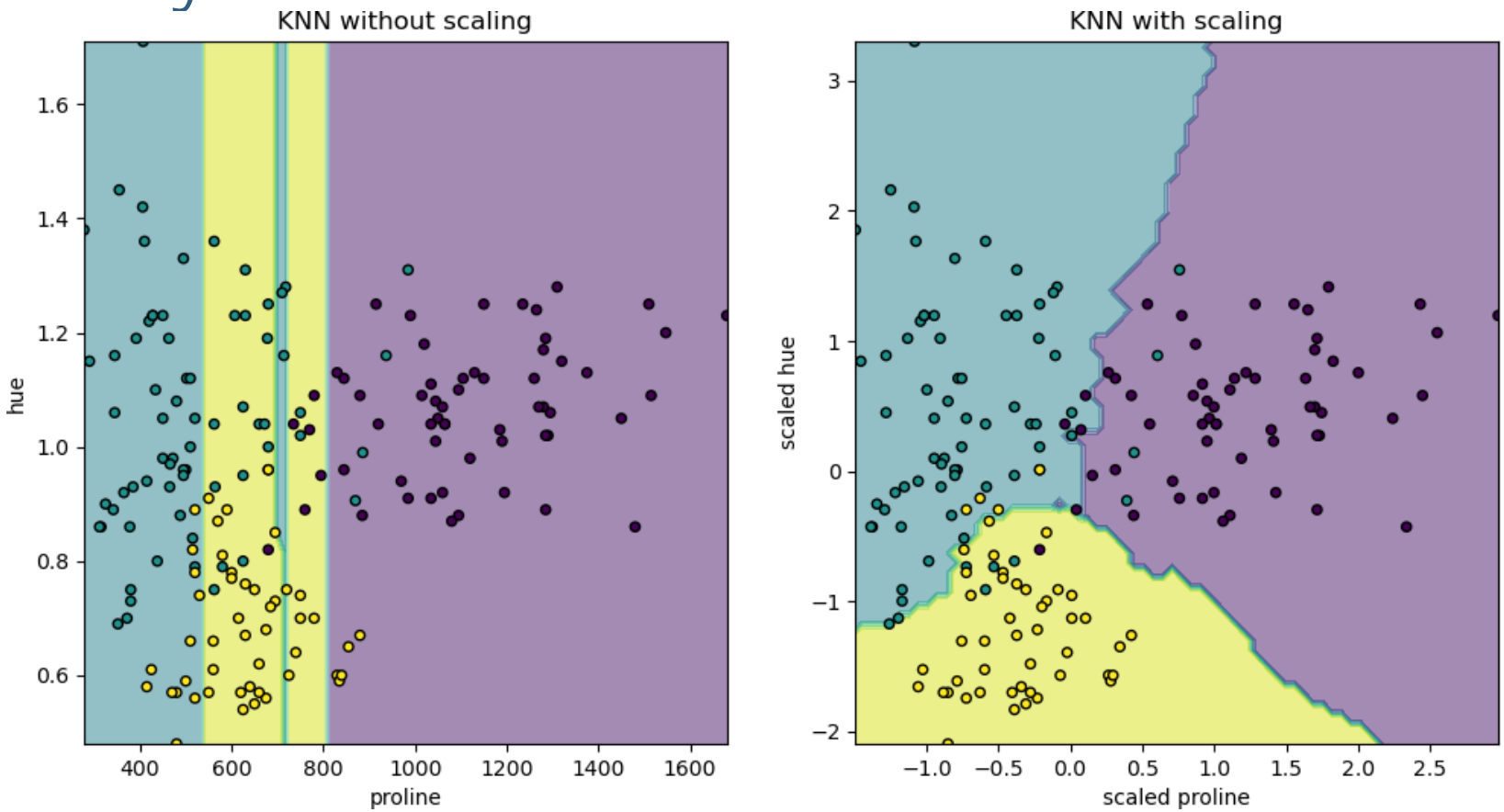
K-Nearest Neighbors



K is the number of nearest neighbors. The number of neighbors is the core deciding factor. K is generally an odd number if the number of classes is 2.

To choose the value of K, you can use the elbow method, covered in the Clustering part of the Unsupervised Learning I lecture.

K-Nearest Neighors Normalization and Decision Boundary



In this figure we show the effect of feature scaling on KNN (using z-score scaling), and also the decision boundary shapes.

Outline

Outline

Improving Machine Learning Models

Your model overfits or does not have the performance you need. What to do? What are your options?

In the real world, when we need to make a very important decision, we usually take the opinion of more than one person, usually multiple experts.

These are called ensemble models, and they basically combine the predictions of multiple models, and in general, these predictions are better than the individuals.

Ensembles

There are many ways to perform ensembling, and they can be divided into two large classes:

Homogeneous Ensembles

All models in the ensemble are similar in structure but different in parameters only. Basically the ensemble contains copies of the same model.

Heterogeneous Ensembles

Models in the ensemble are different in nature (linear models, neural networks, SVMs, etc) and do not share structure or model equations.

All ensemble methods use one hyper-parameter in common, the number of ensemble members M.

Basic Ensemble Methods

Voting Model

The most simple ensembling method is to train M copies of a model $f_i(x)$, which will obtain different weights due to random initialization. Then the predictions of each model are combined into a single prediction by taking the mean:

$$f_{ens}(x) = M^{-1} \sum_{i}^{M} f_i(x) \tag{16}$$

Each model f_i has parameters θ_i. This works for both classification and regression. For classification, the ensemble outputs the average probability vector, while for regression it reports the mean prediction.

Basic Ensemble Methods

Boosting

In a boosting method, weak learners are used, which are simple ML models (like one feature plus a threshold) that work barely better than random chance, but are combined in sequence.

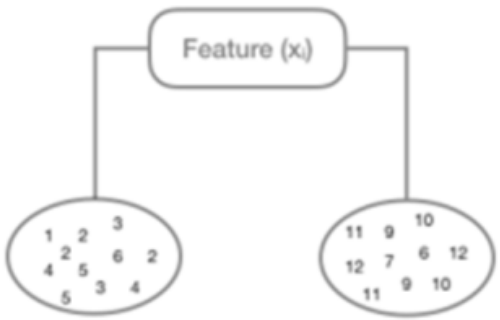
The idea is that at each weak learner, the data is weighted so the weak learner learns to correctly predict the samples that the previous weak learner predicted incorrectly.

Then at inference time, the weak learners are applied in sequence, each producing a class result or continuing in the chain of weak learners.

Outline

Regression vs Classification trees

Regression tree



Prediction:	3.36	9.7
SSE:	24.55	36.1

Classification tree

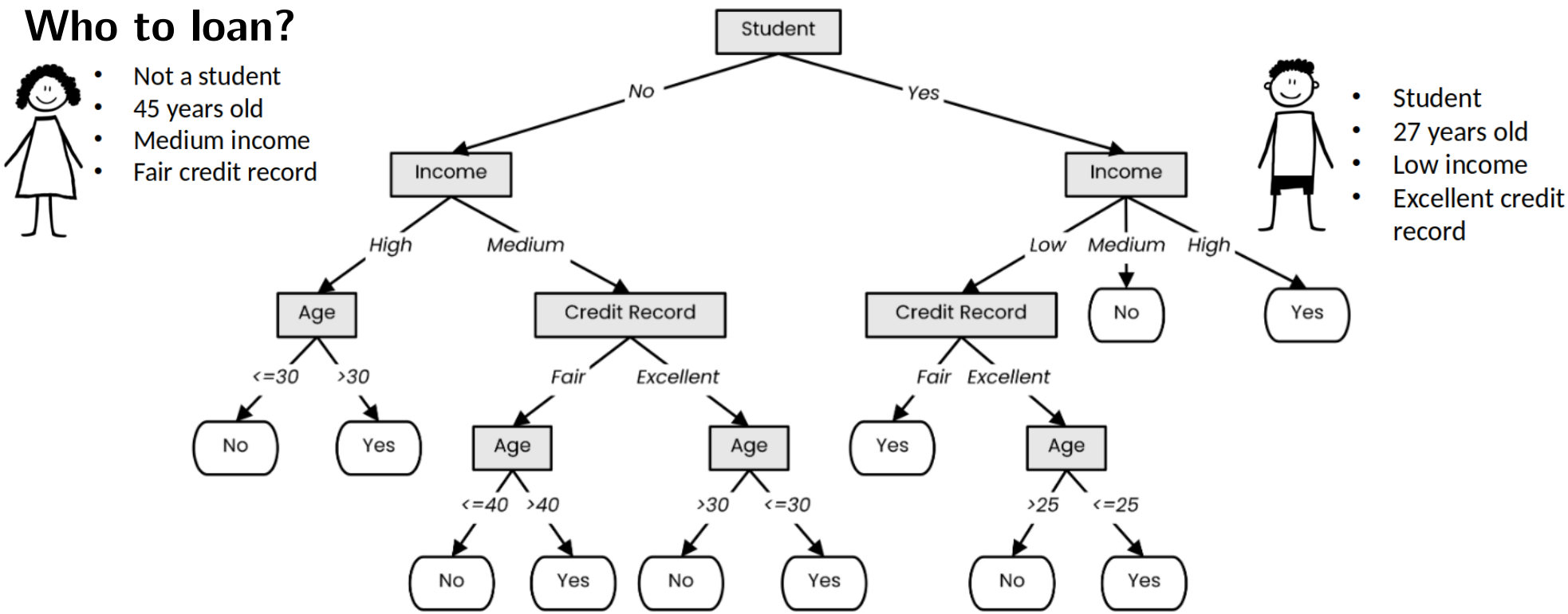
Prediction:	no	yes
Gini:	.16	0

Objective: Minimize disimilarity in terminal nodes

[1] Some slides are adapted from a presentation of Prof. Radu Ionescu (Univ. of Bucharest)

Definition

- A tree-like model that illustrates series of events leading to certain decisions
- Each node represents a test on an attribute and each branch is an outcome of that test



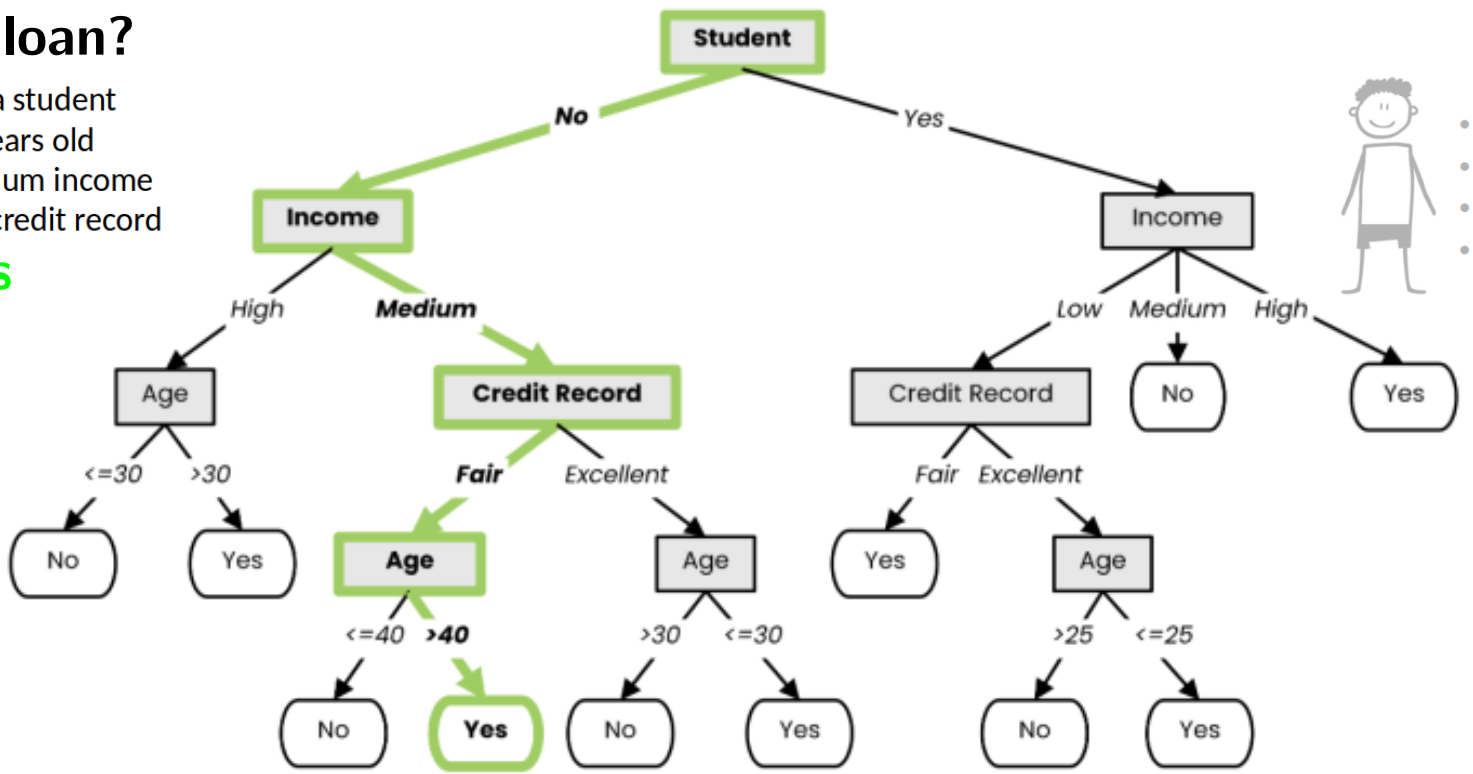
Definition

- A tree-like model that illustrates series of events leading to certain decisions
- Each node represents a test on an attribute and each branch is an outcome of that test

Who to loan?

- Not a student
- 45 years old
- Medium income
- Fair credit record

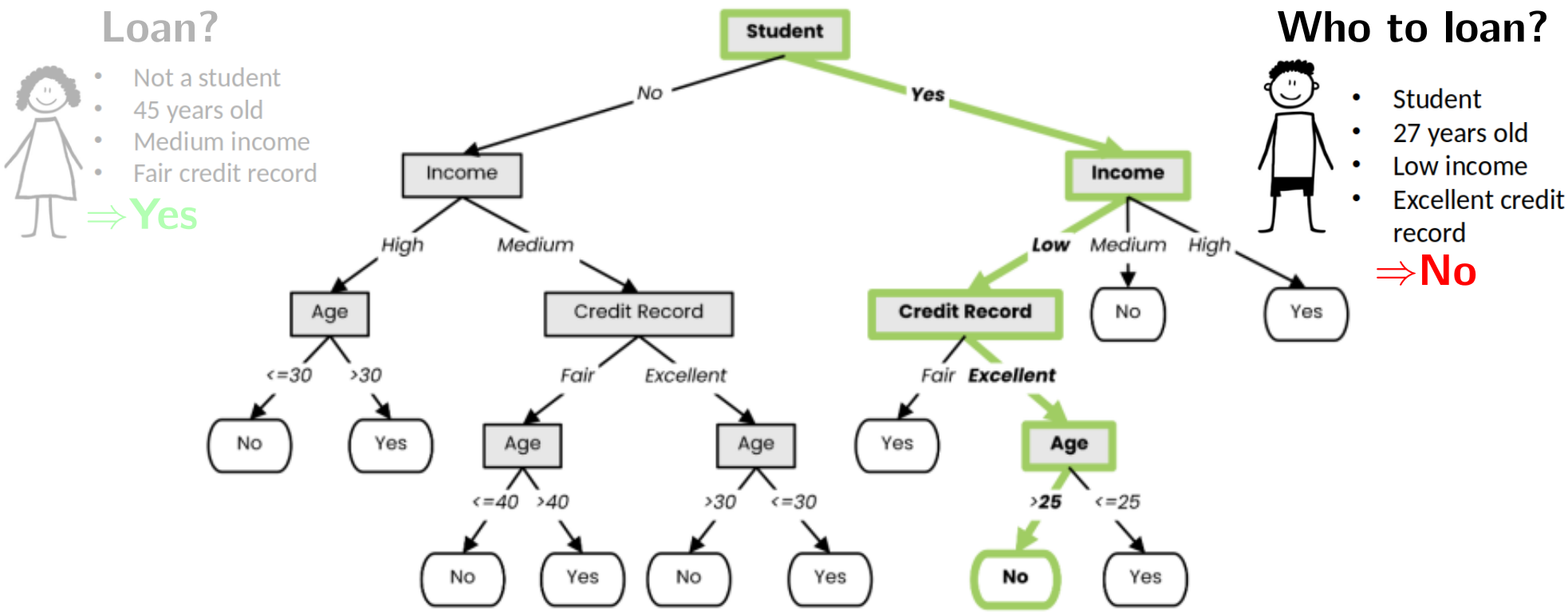
⇒**Yes**



- Student
- 27 years old
- Low income
- Excellent credit record

Definition

- A tree-like model that illustrates series of events leading to certain decisions
- Each node represents a test on an attribute and each branch is an outcome of that test



Decision Tree Learning

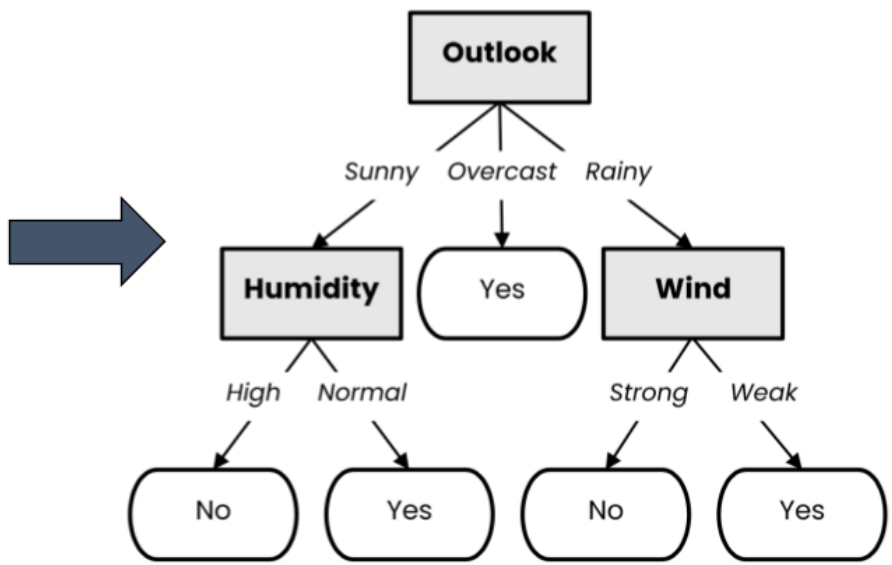
We use labeled data to obtain a suitable decision tree for future predictions.
We want a decision tree that works well on unseen data, while asking as few questions as possible.

Outlook	Temperature	Humidity	Wind	Play Tennis?
Sunny	Hot	High	Weak	**No**
Sunny	Hot	High	Strong	**No**
Overcast	Hot	High	Weak	**Yes**
Rainy	Mild	High	Weak	**Yes**
Rainy	Cool	Normal	Weak	**Yes**
Rainy	Cool	Normal	Strong	**No**
Overcast	Cool	Normal	Strong	**Yes**
Sunny	Mild	High	Weak	**No**
Sunny	Cool	Normal	Weak	**Yes**
Rainy	Mild	Normal	Weak	**Yes**
Sunny	Mild	Normal	Strong	**Yes**
Overcast	Mild	High	Strong	**Yes**
Overcast	Hot	Normal	Weak	**Yes**
Rainy	Mild	High	Strong	**No**

Decision Tree Learning

We use labeled data to obtain a suitable decision tree for future predictions.
We want a decision tree that works well on unseen data, while asking as few questions as possible.

Outlook	Temperature	Humidity	Wind	Play Tennis?
Sunny	Hot	High	Weak	**No**
Sunny	Hot	High	Strong	**No**
Overcast	Hot	High	Weak	**Yes**
Rainy	Mild	High	Weak	**Yes**
Rainy	Cool	Normal	Weak	**Yes**
Rainy	Cool	Normal	Strong	**No**
Overcast	Cool	Normal	Strong	**Yes**
Sunny	Mild	High	Weak	**No**
Sunny	Cool	Normal	Weak	**Yes**
Rainy	Mild	Normal	Weak	**Yes**
Sunny	Mild	Normal	Strong	**Yes**
Overcast	Mild	High	Strong	**Yes**
Overcast	Hot	Normal	Weak	**Yes**
Rainy	Mild	High	Strong	**No**

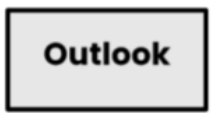


Decision Tree Learning

Basic step: choose an attribute and, based on its values, split the data into smaller sets

- Recursively repeat this step until we can surely decide the label

Outlook	Temperature	Humidity	Wind	Play Tennis?
Sunny	Hot	High	Weak	**No**
Sunny	Hot	High	Strong	**No**
Overcast	Hot	High	Weak	**Yes**
Rainy	Mild	High	Weak	**Yes**
Rainy	Cool	Normal	Weak	**Yes**
Rainy	Cool	Normal	Strong	**No**
Overcast	Cool	Normal	Strong	**Yes**
Sunny	Mild	High	Weak	**No**
Sunny	Cool	Normal	Weak	**Yes**
Rainy	Mild	Normal	Weak	**Yes**
Sunny	Mild	Normal	Strong	**Yes**
Overcast	Mild	High	Strong	**Yes**
Overcast	Hot	Normal	Weak	**Yes**
Rainy	Mild	High	Strong	**No**



Decision Tree Learning

Basic step: choose an attribute and, based on its values, split the data into smaller sets

- Recursively repeat this step until we can surely decide the label

Outlook = *Sunny*

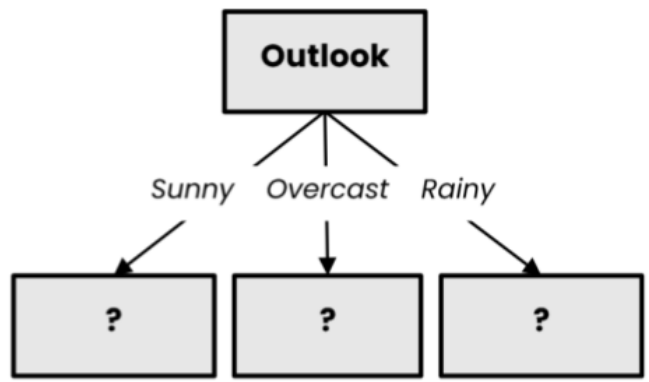
Temperature	Humidity	Wind	Play Tennis?
Hot	High	Weak	**No**
Hot	High	Strong	**No**
Mild	High	Weak	**No**
Cool	Normal	Weak	**Yes**
Mild	Normal	Strong	**Yes**

Outlook = *Overcast*

Temperature	Humidity	Wind	Play Tennis?
Hot	High	Weak	**Yes**
Cool	Normal	Strong	**Yes**
Mild	High	Strong	**Yes**
Hot	Normal	Weak	**Yes**

Outlook = *Rainy*

Temperature	Humidity	Wind	Play Tennis?
Mild	High	Weak	**Yes**
Cool	Normal	Weak	**Yes**
Cool	Normal	Strong	**No**
Mild	Normal	Weak	**Yes**
Mild	High	Strong	**No**



Decision Tree Learning

Basic step: choose an attribute and, based on its values, split the data into smaller sets

- Recursively repeat this step until we can surely decide the label

Outlook = *Sunny*

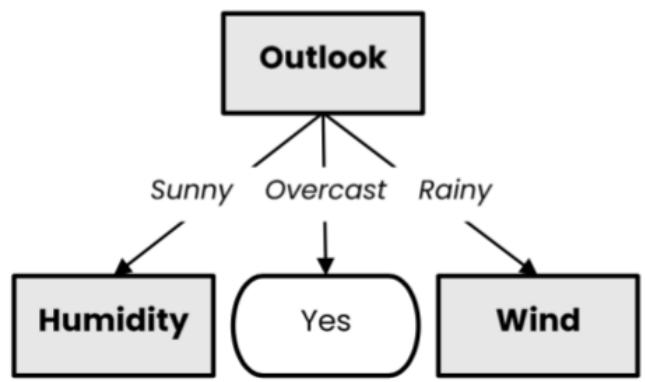
Temperature	Humidity	Wind	Play Tennis?
Hot	High	Weak	**No**
Hot	High	Strong	**No**
Mild	High	Weak	**No**
Cool	Normal	Weak	**Yes**
Mild	Normal	Strong	**Yes**

Outlook = *Overcast*

Temperature	Humidity	Wind	Play Tennis?
Hot	High	Weak	**Yes**
Cool	Normal	Strong	**Yes**
Mild	High	Strong	**Yes**
Hot	Normal	Weak	**Yes**

Outlook = *Rainy*

Temperature	Humidity	Wind	Play Tennis?
Mild	High	Weak	**Yes**
Cool	Normal	Weak	**Yes**
Cool	Normal	Strong	**No**
Mild	Normal	Weak	**Yes**
Mild	High	Strong	**No**



Decision Tree Learning

Basic step: choose an attribute and, based on its values, split the data into smaller sets

- Recursively repeat this step until we can surely decide the label

Outlook = *Sunny*

Humidity = High

Temperature	Wind	Play Tennis?
Hot	Weak	**No**
Hot	Strong	**No**
Mild	Weak	**No**

Humidity = Normal

Temperature	Wind	Play Tennis?
Cool	Weak	**Yes**
Mild	Strong	**Yes**

Outlook = *Overcast*

Temperature	Humidity	Wind	Play Tennis?
Hot	High	Weak	**Yes**
Cool	Normal	Strong	**Yes**
Mild	High	Strong	**Yes**
Hot	Normal	Weak	**Yes**

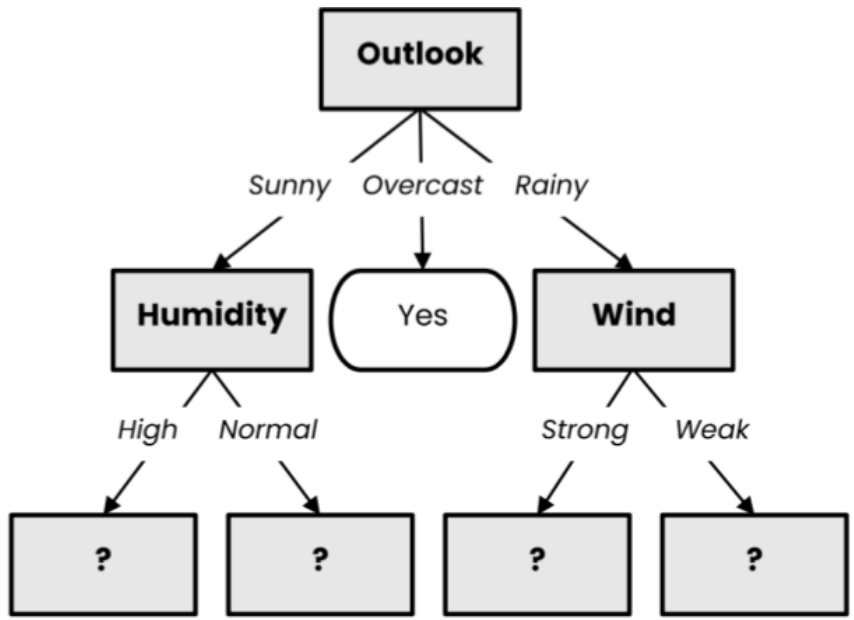
Outlook = *Rainy*

Wind = Strong

Temperature	Humidity	Play Tennis?
Cool	Normal	**No**
Mild	High	**No**

Wind = Weak

Temperature	Humidity	Play Tennis?
Mild	High	**Yes**
Cool	Normal	**Yes**
Mild	Normal	**Yes**



Decision Tree Learning

Basic step: choose an attribute and, based on its values, split the data into smaller sets

- Recursively repeat this step until we can surely decide the label

Outlook = *Sunny*

Humidity = High

Temperature	Wind	Play Tennis?
Hot	Weak	**No**
Hot	Strong	**No**
Mild	Weak	**No**

Humidity = Normal

Temperature	Wind	Play Tennis?
Cool	Weak	**Yes**
Mild	Strong	**Yes**

Outlook = *Overcast*

Temperature	Humidity	Wind	Play Tennis?
Hot	High	Weak	**Yes**
Cool	Normal	Strong	**Yes**
Mild	High	Strong	**Yes**
Hot	Normal	Weak	**Yes**

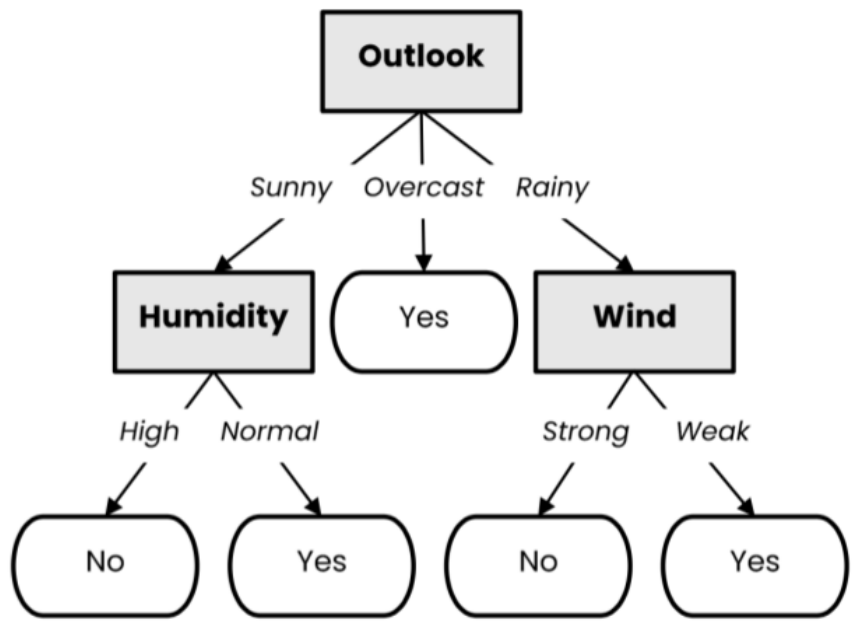
Outlook = *Rainy*

Wind = Strong

Temperature	Humidity	Play Tennis?
Cool	Normal	**No**
Mild	High	**No**

Wind = Weak

Temperature	Humidity	Play Tennis?
Mild	High	**Yes**
Cool	Normal	**Yes**
Mild	Normal	**Yes**



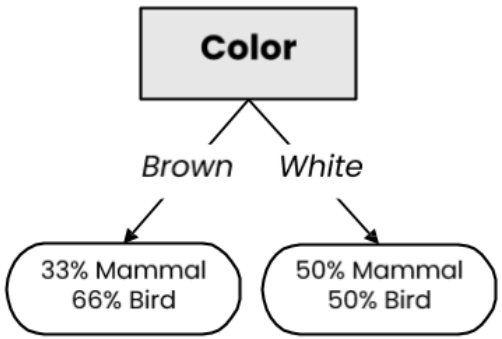
What is a good attribute?

Does it fly?	Color	Class
No	Brown	**Mammal**
No	White	**Mammal**
Yes	Brown	**Bird**
Yes	White	**Bird**
No	White	**Mammal**
No	Brown	**Bird**
Yes	White	**Bird**

- Which attribute provides better splitting?
- Why?
  - Because the resulting subsets are more pure
  - Knowing the value of this attribute gives us more information about the label (the entropy of the subsets is lower)$\rightarrow$ you will hear more about this in a future lecture

What is a good attribute?

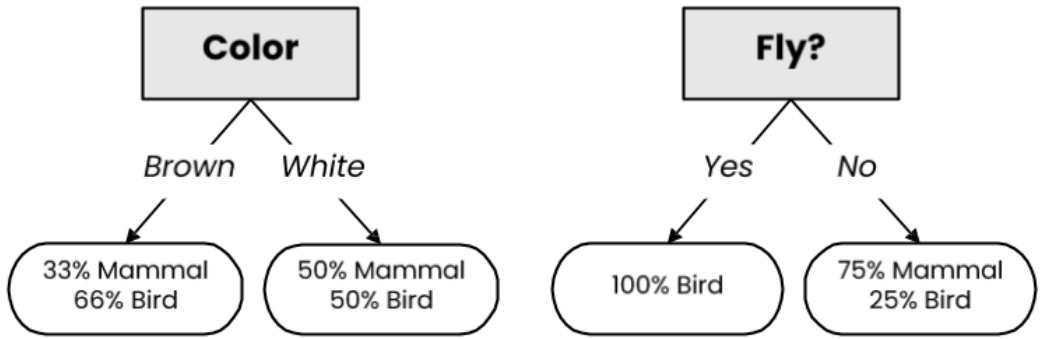
Does it fly?	Color	Class
No	Brown	**Mammal**
No	White	**Mammal**
Yes	Brown	**Bird**
Yes	White	**Bird**
No	White	**Mammal**
No	Brown	**Bird**
Yes	White	**Bird**



- Which attribute provides better splitting?
- Why?
  - Because the resulting subsets are more pure
  - Knowing the value of this attribute gives us more information about the label (the entropy of the subsets is lower)$\rightarrow$ you will hear more about this in a future lecture

What is a good attribute?

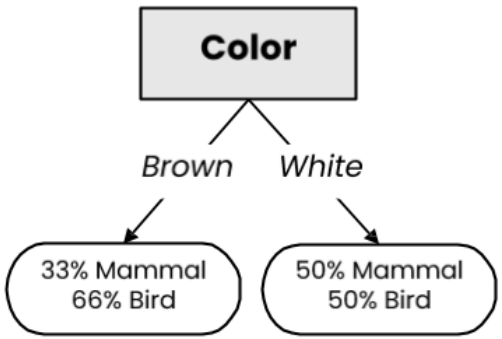
Does it fly?	Color	Class
No	Brown	**Mammal**
No	White	**Mammal**
Yes	Brown	**Bird**
Yes	White	**Bird**
No	White	**Mammal**
No	Brown	**Bird**
Yes	White	**Bird**

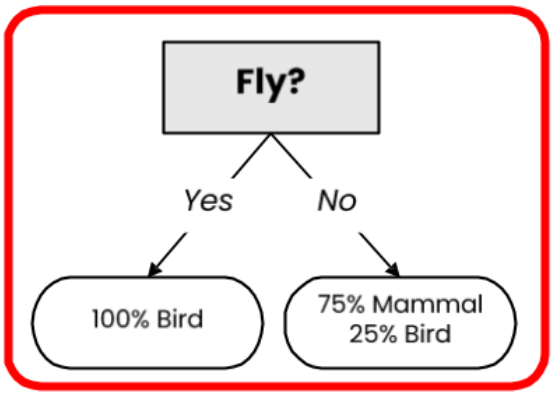


- Which attribute provides better splitting?
- Why?
  - Because the resulting subsets are more pure
  - Knowing the value of this attribute gives us more information about the label (the entropy of the subsets is lower)$\rightarrow$ you will hear more about this in a future lecture

What is a good attribute?

Does it fly?	Color	Class
No	Brown	**Mammal**
No	White	**Mammal**
Yes	Brown	**Bird**
Yes	White	**Bird**
No	White	**Mammal**
No	Brown	**Bird**
Yes	White	**Bird**





- Which attribute provides better splitting?
- Why?
  - Because the resulting subsets are more pure
  - Knowing the value of this attribute gives us more information about the label (the entropy of the subsets is lower)$\rightarrow$ you will hear more about this in a future lecture

Decision Tree - Strengths & Weaknesses

Strengths :D

- Small trees are easy to interpret
- Trees scale well to large N (fast!!)
- Can handle data of all types (i.e., requires little, if any, preprocessing)
- Automatic variable selection
- Can handle missing data
- Completely nonparametric

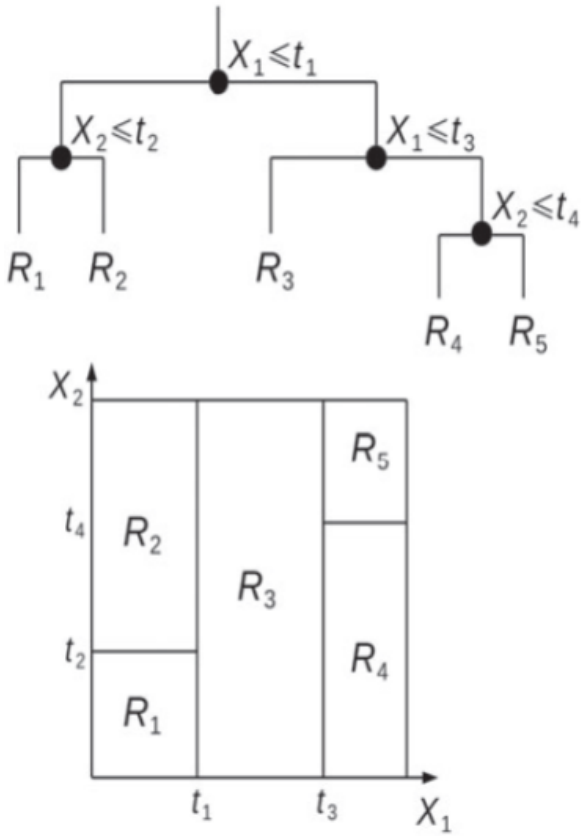
Weaknesses :'(

- Large trees can be difficult to interpret
- All splits depend on previous splits (i.e. capturing inteactions :D - additive models :'( )
- Trees are step functions (i.e., binary splits)
- Single trees typically have poor predictive accuracy
- Single trees have high variance (easy to overfit to training data)

Outline

Random Forests

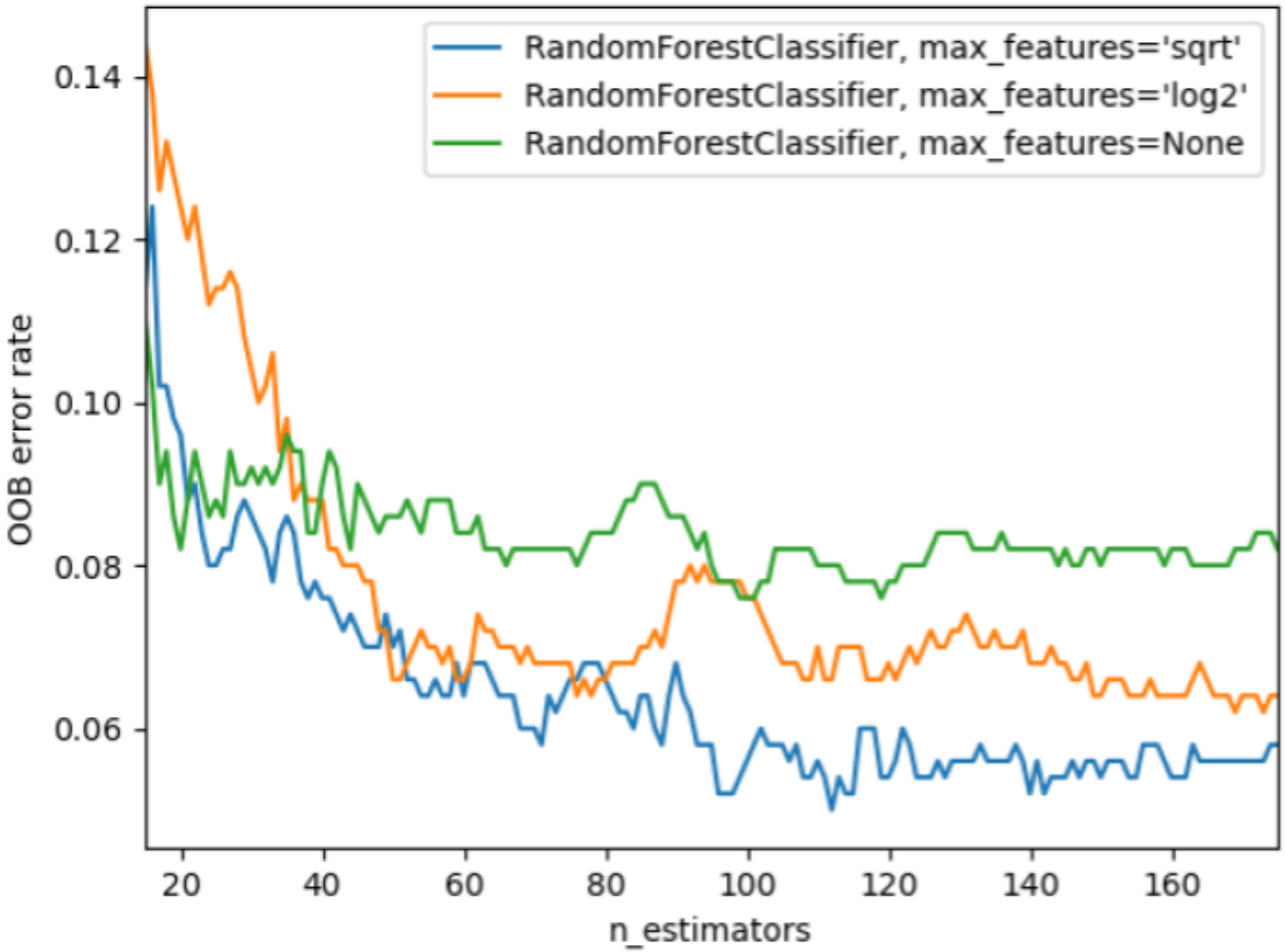
- Binary decision tree
  - Low bias, but high variance
- Bagging (bootstrap aggregation)
  - Many trees to reduce variance
  - Majority voting
  - However, trees should be uncorrelated. Use different trials and different features for each tree



Random Forests

- Random Forests:
  - Instead of building a single decision tree and use it to make predictions, build many slightly different trees and combine their predictions
- We have a single data set, so how do we obtain slightly different trees?
  1. Bagging (**B**ootstrap **Agg**regat**ing**):
     - Take random subsets of data points from the training set to create N smaller data sets
     - Fit a decision tree on each subset
  2. Random Subspace Method (also known as Feature Bagging):
     - Fit N different decision trees by constraining each one to operate on a random subset of features

Relation between number of estimators and error rate



Random Forest - Strengths & Weaknesses

Strengths :D

- Competitive performance
- Remarkably good "out-of-the-box" (very little tuning required)
- Built-in validation set (don't need to sacrifice data for extra validation)
- Typically does not overfit
- Robust to outliers
- Handles missing data (imputation not required)
- Provides automatic feature selection
- Minimal preprocessing required

Weaknesses :'(

- Although accurate, often cannot compete with the accuracy of advanced boosting algorithms
- Can become slow on large data sets
- Less interpretable (although this is easily addressed with various tools such as variance importance, partial dependence plots, LIME, etc.)

Questions to Think About

1. What is Maximum Likelihood estimation?
2. Explain the principle behind Naive Bayes.
3. What is a Prior in Bayesian Thinking?
4. Why is k-NN Non-Linear?
5. Why does an Ensemble improve performance?
6. Why do Random Forests often outperform Decision Trees?
7. Are Decision Trees Linear or Non-Linear methods?
8. How does the decision boundary of a Random Forest look like, considering the decision boundary of a Decision Tree?

Take Home Messages

- We covered a lot of methods, there are important underpinning concepts like linear models, linear separability, kernel tricks, etc.
- Linear methods are easy to understand and implement, and generally have good performance.
- Kernel methods allow to transform Linear methods into non-linear ones, with simple tricks.
- Non-linear methods can have better performance but they are difficult to understand and implement.
- In the end performance depends on the features and if they are linearly separable (classification) or fit a line (regression).

Questions?