Introduction to Artificial Intelligence (ENSIMAG) Intelligent Systems (MOSIG)

Some models for unsupervised and supervised learning

Original Slides by Clovis Galiez
Lecture: Sergi Pujades

2021-2022

Outline

- Unsupervised and supervised learning
- Unsupervised learning
  - EM
  - K-Means
  - PCA
  - t-SNE
- Supervised models
  - General setting
  - Logistic regression
  - SVM
  - Random forest

Supervised and unsupervised learning

Supervised and unsupervised learning

Make sense of the data

Example

Single nucleotide polymorphisms, frequently called SNPs (pronounced "snips"), are the most common type of genetic variation among people

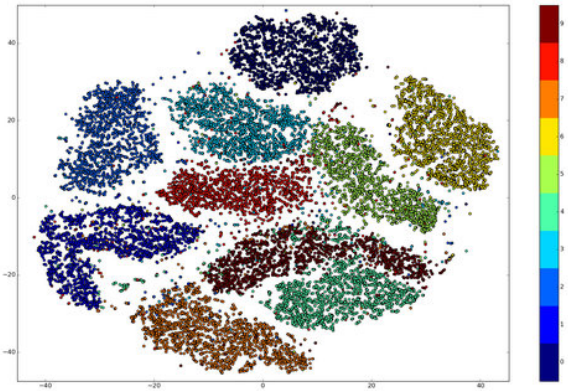
Patient	SNP1	SNP2	SNP3	...
A	0	0	1	...
B	1	0	1	...
C	1	0	0	...
...				

Example

Single nucleotide polymorphisms, frequently called SNPs (pronounced "snips"), are the most common type of genetic variation among people

Patient	SNP1	SNP2	SNP3	...
A	0	0	1	...
B	1	0	1	...
C	1	0	0	...
...				

$\rightarrow$

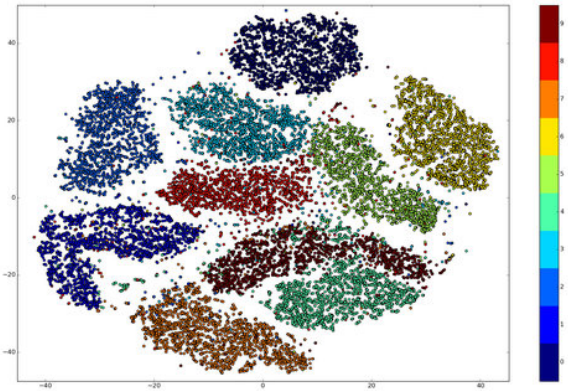


Example

Single nucleotide polymorphisms, frequently called SNPs (pronounced "snips"), are the most common type of genetic variation among people

Patient	SNP1	SNP2	SNP3	...
A	0	0	1	...
B	1	0	1	...
C	1	0	0	...
...				

$\rightarrow$



Techniques

Dimensionality reduction, clustering.

"Slightly" different data

From genotypes:

Patient	Status	SNP1	SNP2	SNP3	SNP4	...
A	**0**	0	0	0	1	...
B	**1**	1	0	0	1	...
C	**1**	1	0	0	0	...
...						

...

"Slightly" different data

From genotypes:

Patient	Status	SNP1	SNP2	SNP3	SNP4	...
A	**0**	0	0	0	1	...
B	**1**	1	0	0	1	...
C	**1**	1	0	0	0	...
...						

...

- discover a function $f(\text{SNP1}, \text{SNP2}, \text{SNP3}, \text{SNP4}, ...) = \text{Status}$ that predicts the status of a **(future)** patient,
- explain which SNP is important.

"Slightly" different data

From genotypes:

Patient	Status	SNP1	SNP2	SNP3	SNP4	...
A	**0**	0	0	0	1	...
B	**1**	1	0	0	1	...
C	**1**	1	0	0	0	...
...						

...

- discover a function $f(\text{SNP1}, \text{SNP2}, \text{SNP3}, \text{SNP4}, ...) = \text{Status}$ that predicts the status of a **(future)** patient,
- explain which SNP is important.

Techniques

Dimensionality reduction, regularization, supervised learning.

Learning: Make sense of data

Two cases:

- Data is only a set of points: $\mathcal{D} = (x_1, ...x_n)$ where $x_i \in \mathbb{E}^D$ is some (vector) space.

Learning: Make sense of data

Two cases:

- Data is only a set of points: $\mathcal{D} = (x_1, ...x_n)$ where $x_i \in \mathbb{E}^D$ is some (vector) space.
- Data is labelled: $\mathcal{D} = ((x_1, y_1), ...(x_n, y_n))$ where $y_i \in Y$ can be de discrete or continuous space.

Learning: Make sense of data

Two cases:

- Data is only a set of points: $\mathcal{D} = (x_1, ...x_n)$ where $x_i \in \mathbb{E}^D$ is some (vector) space. **Unsupervised learning**
- Data is labelled: $\mathcal{D} = ((x_1, y_1), ...(x_n, y_n))$ where $y_i \in Y$ can be de discrete or continuous space.**Supervised learning**

Technically, the inference methods algorithm are quite different.

Supervised learning

Goal

Exactly the same as the unsupervised case:

The goal as usual, is to make sense of the data. For this we define a model $\mathcal{M}(\theta)$ that have some parameters θ, and we try to get the model fit to the data by **minimizing a loss**.

Goal

Exactly the same as the unsupervised case:

The goal as usual, is to make sense of the data. For this we define a model $\mathcal{M}(\theta)$ that have some parameters θ, and we try to get the model fit to the data by **minimizing a loss**.

Which model? Which loss?

Classification

Let:

- X be an D-dimensional random variable,
- and Y binary (0/1) random variable.

X and Y are linked by some unknown *joint* distribution.

A predictor can be thought as a parametrized model $\mathcal{M}(\theta)$ of the conditional distribution $Y|X$.
The loss is usually chosen as the negative log-likelihood of the data:
$-\sum_i \log p_\theta(Y = y_i|X = x_i)$

$$\mathcal{M}(\theta) = P(Y = 1|X, \theta)$$

$$P(Y = 1|X, \theta) + P(Y = 0|X, \theta) = 1$$

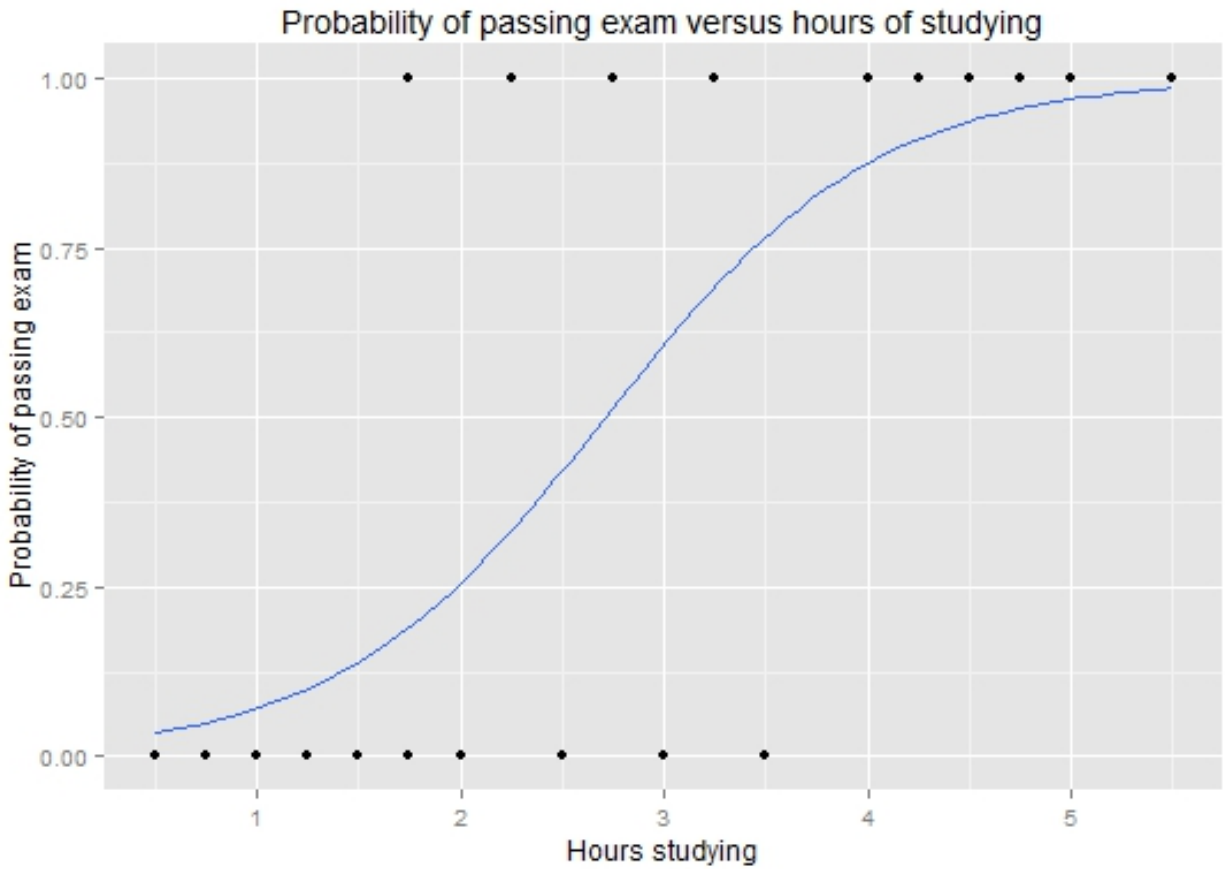
$$P(Y = 0|X, \theta) = 1 - P(Y = 1|X, \theta)$$

Logistic regression

Hours (x_i)	0.50	0.75	1.25	1.75	2.00	2.5	3.75	4.00	5.00	5.50
Pass (y_i)	0	0	0	1	0	1	0	1	1	1

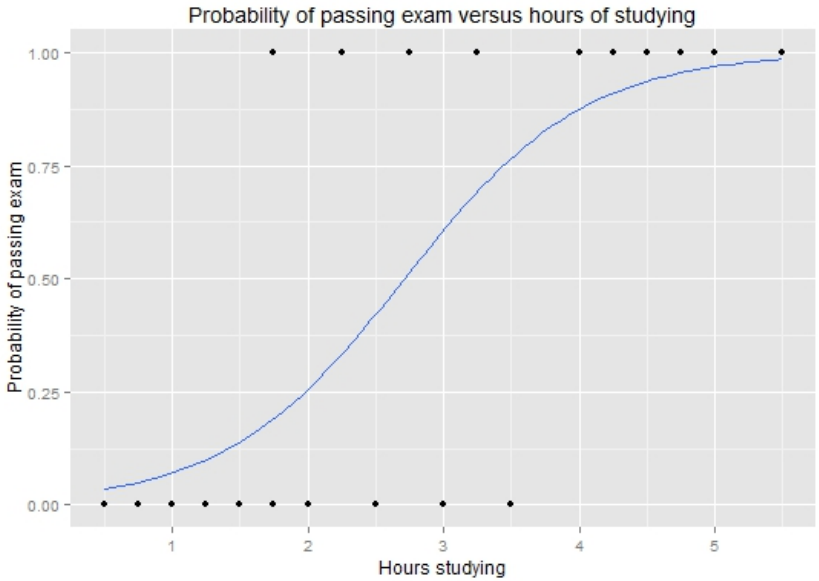
Logistic regression

Hours (x_i)	0.50	0.75	1.25	1.75	2.00	2.5	3.75	4.00	5.00	5.50
Pass (y_i)	0	0	0	1	0	1	0	1	1	1



[Source: Wikipedia]

Logistic regression



The probability to pass the exam can be modeled by

$$p(Y = 1|x) = \frac{1}{1 + e^{\frac{-(x-\mu)}{s}}}$$

can be rewritten as

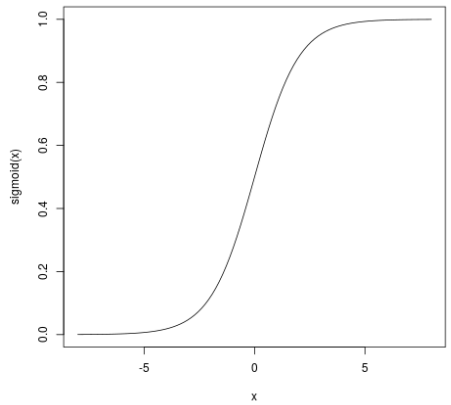
$$p(Y = 1|x) = \frac{1}{1 + e^{-(wx+b)}}$$

Logistic regression

So we write

$$p(Y = 1|x) = \sigma(w.x + b)$$

where the function σ is the logistic sigmoid $\sigma : x \mapsto \frac{1}{1+e^{-x}}$



Exercice

Let f be the predictor $f_{w,b}(x) = p(Y = 1|x) = \sigma(w.x + b)$. Consider the case where $x \in \mathbb{R}^{\mathbb{D}}$ and interpret geometrically the role of parameters w and b.

Conditional likelihood

To measure the goodness of a fit we use the likelihood function, given by the probability that the set is produced by a logistic function:

$$L = P(y_1, ..., y_N | x_1, ...x_N, w, b) = \prod_{i:y_i=1} p_i \prod_{i:y_i=0} (1 - p_i)$$

we want to find $\theta = (w, b)$ such that $\mathcal{M}(\theta) = p$ maximizes L for the observed data.

Conditional likelihood

Exercise

1. Let $f(x) = p(Y = 1|x) = \sigma(w.x + b)$. Show that the *conditional* log-likelihood $LL = \log P(y_1, ..., y_N|x_1, ..., x_N, w, b)$ can be written as:

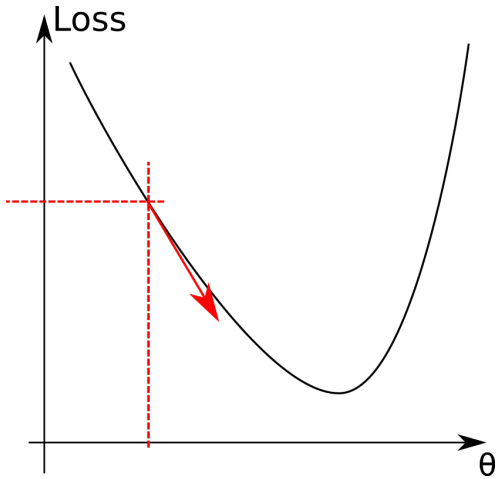
$$LL(w, b) = -\sum_{i=1}^{N} [y_i . \log f(x_i) + (1 - y_i) . \log(1 - f(x_i))]$$

The name of the loss $\mathcal{L}(w, b; x) = -LL(w, b)$ is called the logistic loss, or binary cross-entropy.

2. Show that if $X|Y = i \sim \mathcal{N}(\vec{\mu_i}, \Sigma)$, then $p(Y = 1|x)$ can be written as $\sigma(w.x + b)$. Determine w and b.
Hint: start by writing $p(Y = 1|x)$ using the Bayes rule.

Logistic regression algorithmics

The conditional negative log likelihood of the logistic regression is convex, having a unique minimum.



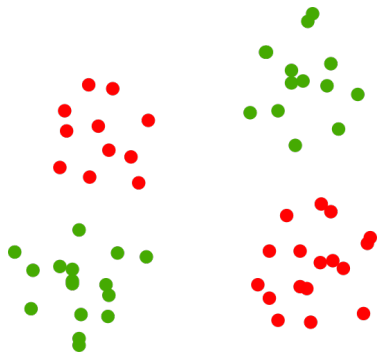
Can be optimized with gradient descent (first order), even speed up by a Newton-Raphson scheme (second order as we can compute the Hessian) $\rightarrow$ leads to an algorithm [Rubin, 83] called *Iterative Reweighted Least Squares*.

Issues with LR: linear separability

Other linear methods exist:

- Perceptron (lectures about neural networks)
- Fisher's Linear Discriminant

Most of the time, points are not linearly separable (thus, cannot be learnt with logistic regression):

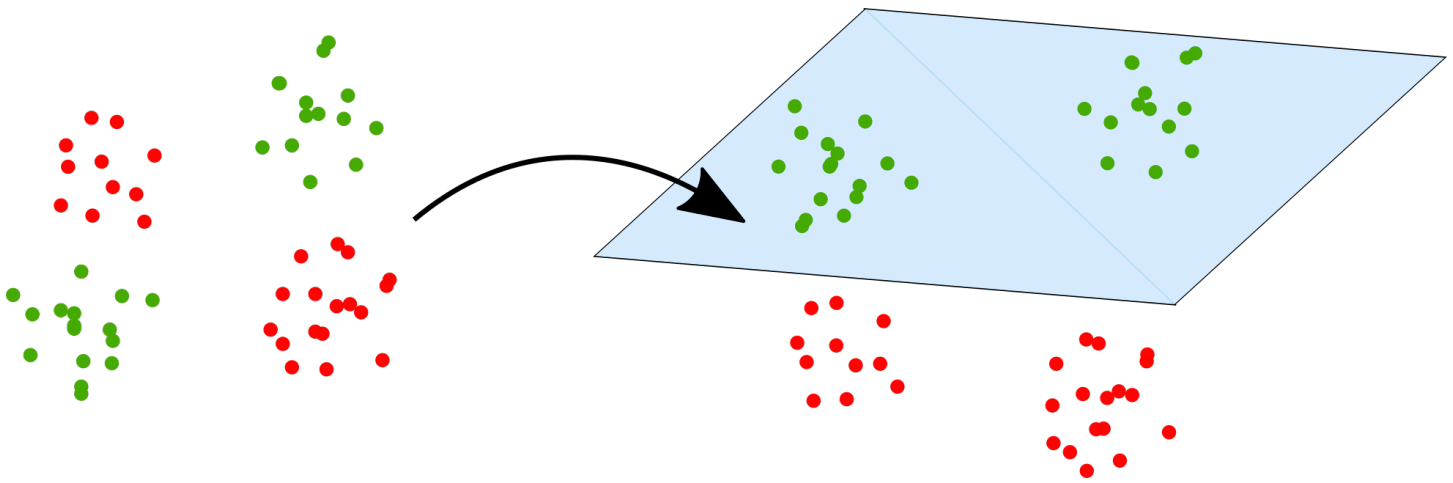


Toward kernel methods

One trick consists into projecting the points into a higher dimensional space where points are linearly separable:

Toward kernel methods

One trick consists into projecting the points into a higher dimensional space where points are linearly separable:



The idea is to replace the terms x_i by a projected version $\Phi(x_i)$ in a higher dimensional space (projection chosen so that hopefully the data is more linearly separable), and learn a linear classifier there.

Which projection?

We don't design the projection by hand.
Projections are usually chosen in families of projections known for:

- easing linear separation
- their computational tractability (see kernel trick just after)

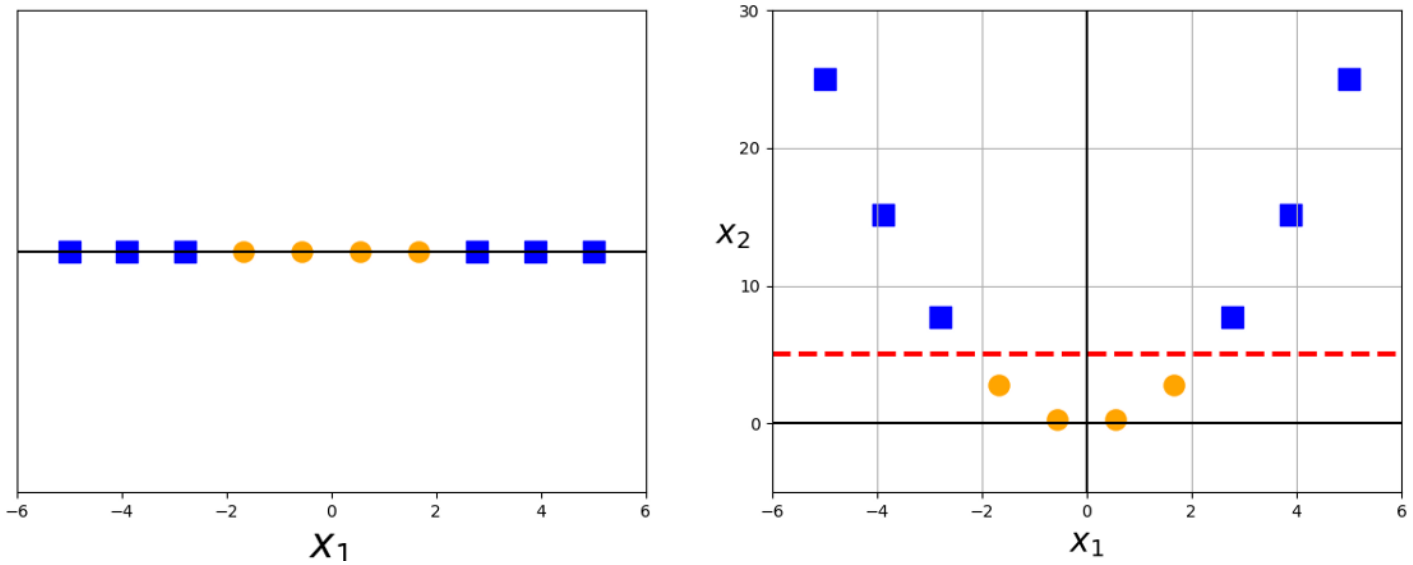
Which projection?

We don't design the projection by hand.
Projections are usually chosen in families of projections known for:

- easing linear separation
- their computational tractability (see kernel trick just after)

Indeed, Φ can project to a high (possibly infinity) dimensional space, that make the parameters and the scalar product $w.\Phi(x_i)$ costly/impossible to compute.

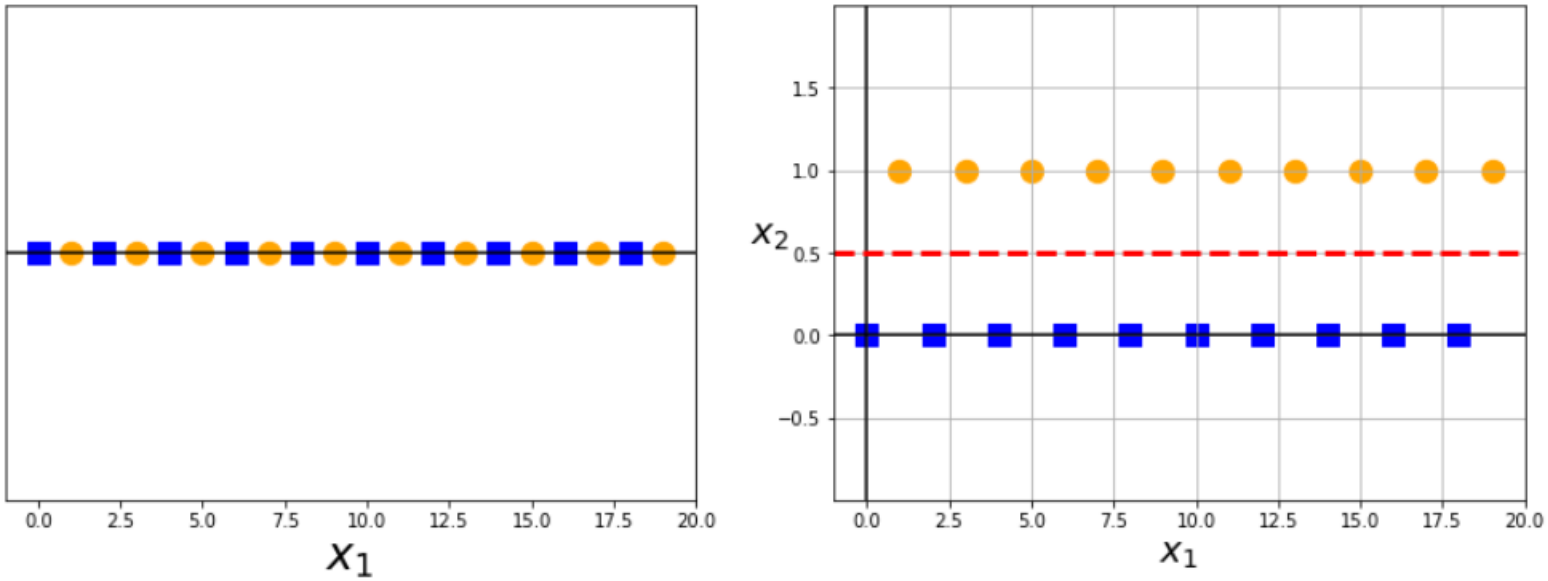
Which projection?



$$\Phi(x) = x^2$$

[Images from https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f]

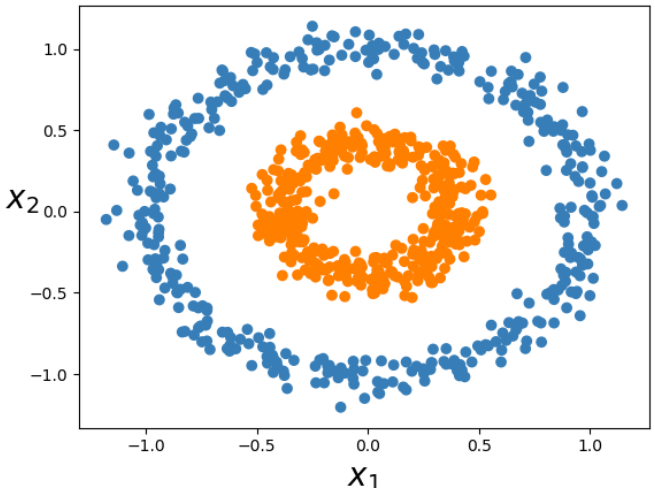
Which projection?

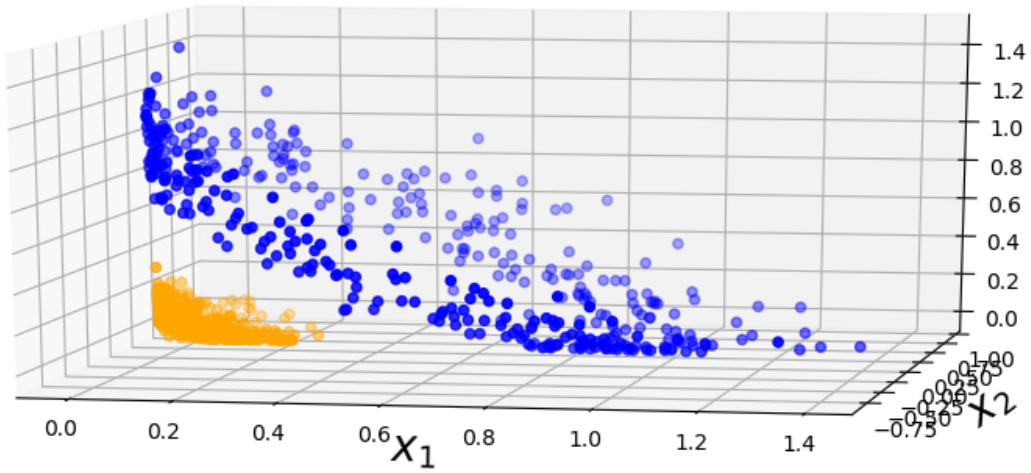


$$\Phi(x) = x \mod 2$$

[Images from https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f]

Which projection?





$$\Phi(x) = \Phi((x_1, x_2)) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

[Images from https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f]

Kernel trick

2 ingredients:

Kernel trick

2 ingredients:
1.Reformulate the loss (dual formulation) so that it involves only a **linear combination of terms** of the form $\Phi(x_i).\Phi(x_j)$ (no w, b anymore, but Φ comes with it's own parameters).

Kernel trick

2 ingredients:

1.Reformulate the loss (dual formulation) so that it involves only a **linear combination of terms** of the form $\Phi(x_i).\Phi(x_j)$ (no w, b anymore, but Φ comes with it's own parameters).

2. We can choose Φ so that $\Phi(x_i).\Phi(x_j)$ can be fast to compute.

Kernel trick

2 ingredients:
1.Reformulate the loss (dual formulation) so that it involves only a **linear combination of terms** of the form $\Phi(x_i).\Phi(x_j)$ (no w, b anymore, but Φ comes with it's own parameters).
2. We can choose Φ so that $\Phi(x_i).\Phi(x_j)$ can be fast to compute.

Kernel trick

Instead of choosing a projection, and computing the scalar product, we choose a *kernel* that computes from 2 low dimensional vectors their scalar product in high dimension **without explicitly** computing the projection.

Formally, we have data $\mathbf{x_i}, \mathbf{x_j} \in \mathbb{R}^D$ and a map $\Phi : \mathbb{R}^D \to \mathbb{R}^E$, then a **kernel function** is

$$k(\mathbf{x_i}, \mathbf{x_j}) = \langle \Phi(\mathbf{x_i}), \Phi(\mathbf{x_j}) \rangle$$

Kernel example

Kernel trick for a 2nd degree polynomial mapping:

$$k(x_i, x_j) = \langle \Phi(a), \Phi(b) \rangle = \begin{bmatrix} a_1^2, \\ \sqrt{2}a_1 a_2 \\ a_2^2 \end{bmatrix}^T \begin{bmatrix} b_1^2, \\ \sqrt{2}b_1 b_2 \\ b_2^2 \end{bmatrix} =$$

$$= a_1^2 b_1^2 + 2a_1 b_1 a_2 b_2 + a_2^2 b_2^2 =$$

$$= (a_1 b_1 + a_2 b_2)^2 = \left(\begin{bmatrix} a_1, \\ a_2 \end{bmatrix}^T \begin{bmatrix} b_1, \\ b_2 \end{bmatrix} \right)^2 =$$

$$= \langle a, b \rangle^2 = \langle x_i, x_j \rangle^2$$

Another common example is the Gaussian Kernel:

$$k(x_i, x_j) = \Phi(x_i).\Phi(x_j) = exp(-\gamma.||x_i - x_j||^2)$$

Kernel example

Kernel trick for a 2nd degree polynomial mapping:

$$k(x_i, x_j) = \langle \Phi(a), \Phi(b) \rangle = \begin{bmatrix} a_1^2, \\ \sqrt{2}a_1a_2 \\ a_2^2 \end{bmatrix}^T \begin{bmatrix} b_1^2, \\ \sqrt{2}b_1b_2 \\ b_2^2 \end{bmatrix} =$$

$$= a_1^2b_1^2 + 2a_1b_1a_2b_2 + a_2^2b_2^2 =$$

$$= (a_1b_1 + a_2b_2)^2 = \left(\begin{bmatrix} a_1, \\ a_2 \end{bmatrix}^T \begin{bmatrix} b_1, \\ b_2 \end{bmatrix} \right)^2 =$$

$$= \langle a, b \rangle^2 = \langle x_i, x_j \rangle^2$$

Another common example is the Gaussian Kernel:

$$k(x_i, x_j) = \Phi(x_i).\Phi(x_j) = exp(-\gamma.||x_i - x_j||^2)$$

There are not established, general rules to know what kernel will work best for your particular data.

Solving kernel methods

The solution of the dual problem (formulation omitted for this unit)

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x_i}$$

The decision boundary for a new point is

$$\mathbf{w}^T \mathbf{x} + w_0 = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x_i^T x} + w_0$$

The decision:

$$y = \text{sign} \left[\sum_{i=1}^{N} \alpha_i y_i \mathbf{x_i^T x} + w_0 \right]$$

Mapping to feature space we have the decision

$$y = \text{sign} \left[\sum_{i=1}^{N} \alpha_i y_i \langle \Phi(\mathbf{x}), \Phi(\mathbf{x_i}) \rangle + w_0 \right]$$

Toward SVM (support vector models)

So far we have:

- Supervised model for classification
- A way to train in the convex case (unique optimum + gradient-related algorithm)
- Extension to deal with the case of non-linear separability

Toward SVM (support vector models)

So far we have:

- Supervised model for classification
- A way to train in the convex case (unique optimum + gradient-related algorithm)
- Extension to deal with the case of non-linear separability

New issue:

Toward SVM (support vector models)

So far we have:

- Supervised model for classification
- A way to train in the convex case (unique optimum + gradient-related algorithm)
- Extension to deal with the case of non-linear separability

New issue:
Due to the kernel, the prediction of the class of a point x cannot be written $\sigma(\Phi(w).\Phi(x) + b)$, but it involves the computation of $\sum_{i=1}^{N} \alpha_i y_i k(x, x_i)$ where N is the size of the training set...

SVMs to the rescue

SVM solves this.

SVM: Support vectors

To avoid the computation of N terms when predicting: the loss is such that the model chooses few data points (called *support vectors*) that will play a role in the loss, the other are discarded.

SVM

SVM finds a linear separation between classes such that it maximizes the distance to the separation hyperplane (called the margin).
Instead of describing the hyperplane with a (potentially infinite) vector w, it writes it as a linear combination of support vectors (picked in the data).

SVM: Support vectors

To avoid the computation of N terms when predicting: the loss is such that the model chooses few data points (called *support vectors*) that will play a role in the loss, the other are discarded.

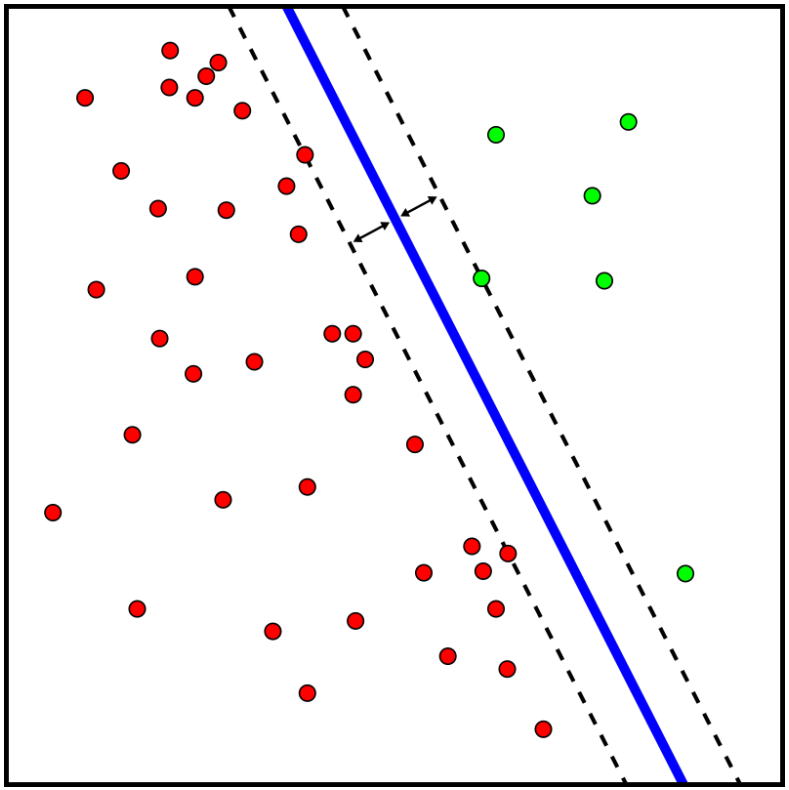
SVM

SVM finds a linear separation between classes such that it maximizes the distance to the separation hyperplane (called the margin).
Instead of describing the hyperplane with a (potentially infinite) vector w, it writes it as a linear combination of support vectors (picked in the data).

With a **Support Vector** set $\mathbf{SV} \subset \mathbf{X}$ we have

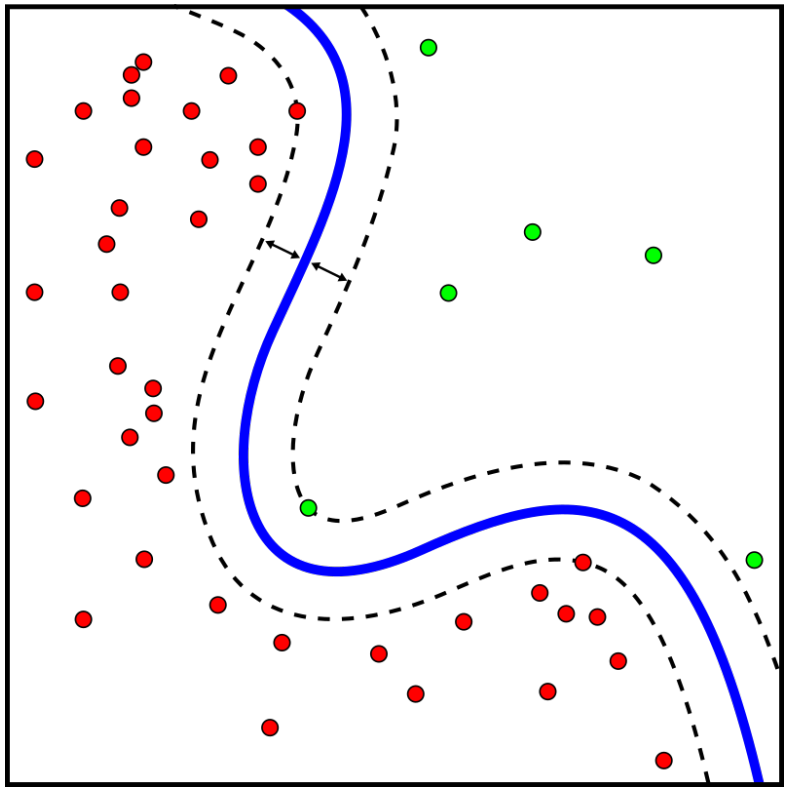
$$y = \text{sign}\left[\sum_{i \in \mathbf{SV}} \alpha_i y_i \langle \Phi(\mathbf{x}), \Phi(\mathbf{x_i}) \rangle + w_0\right]$$

SVM visually



Identity projection $\Phi(x) = x$ (linear kernel)

SVM visually

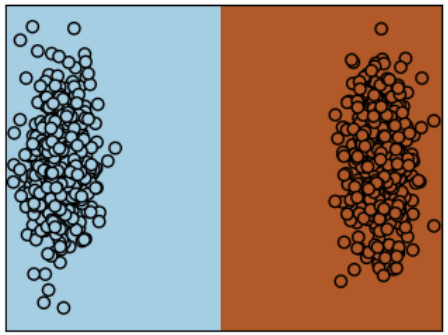


Gaussian kernel: $k(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}$

The blue line is a plane in higher dimensional space, projected in 2D.

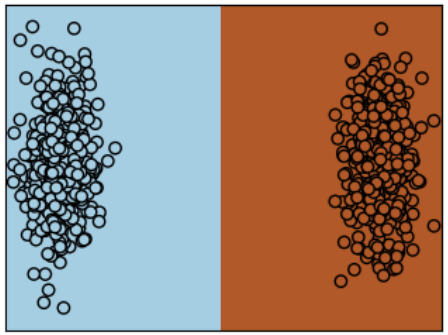
Influence of noise

"Robust" separation

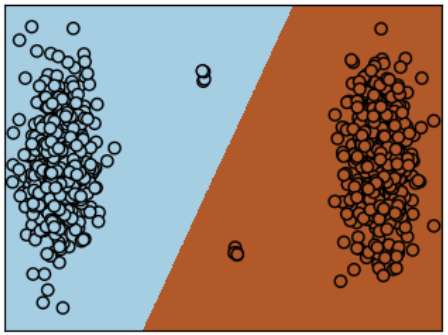


Influence of noise

"Robust" separation

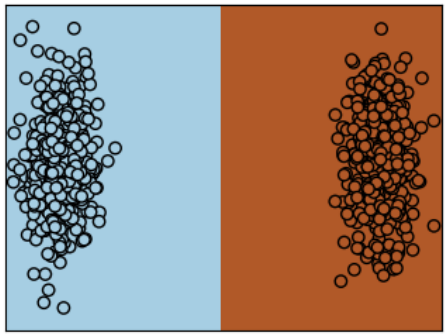


With few noisy points

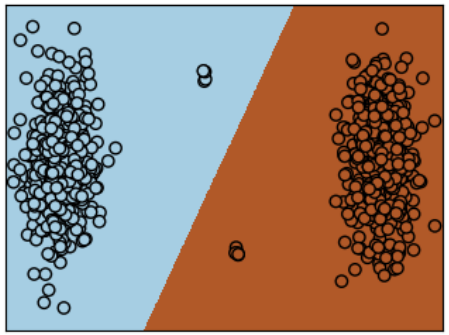


Influence of noise

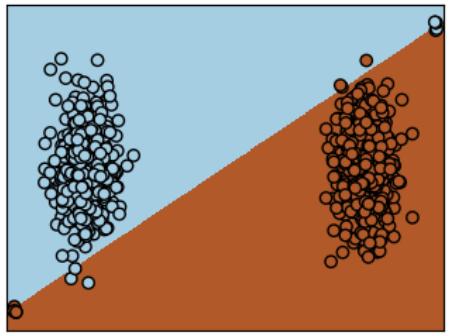
"Robust" separation



With few noisy points

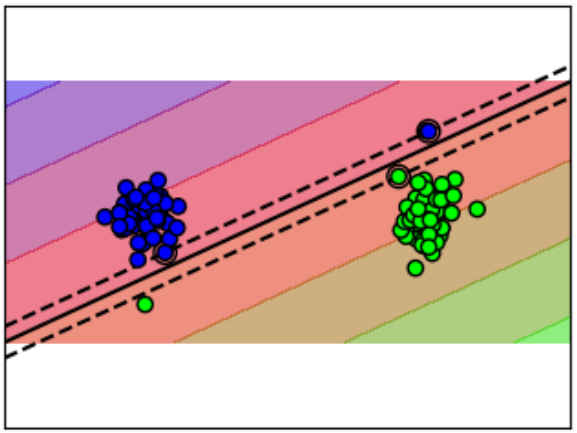


Even more



Soft margins

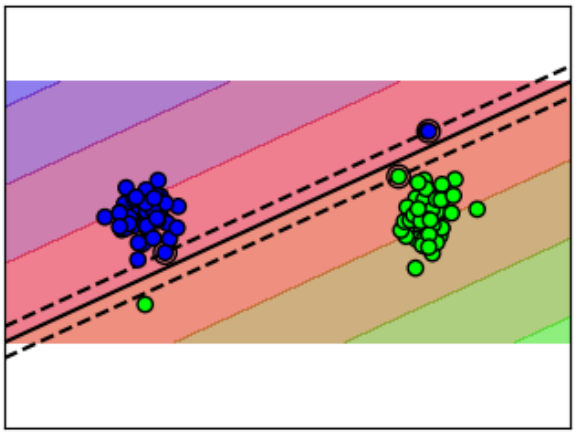
To solve the issue of robustness to points near the decision boundary, one can introduce an hyper-parameter that controls the tolerance to misclassification (during inference). Without entering into details, visually it amounts to:



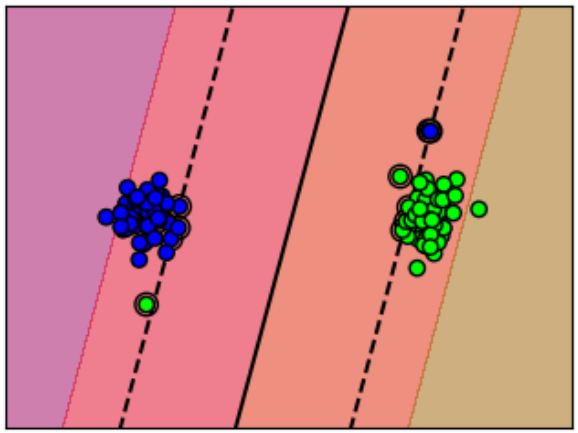
Hard margin

Soft margins

To solve the issue of robustness to points near the decision boundary, one can introduce an hyper-parameter that controls the tolerance to misclassification (during inference). Without entering into details, visually it amounts to:



Hard margin

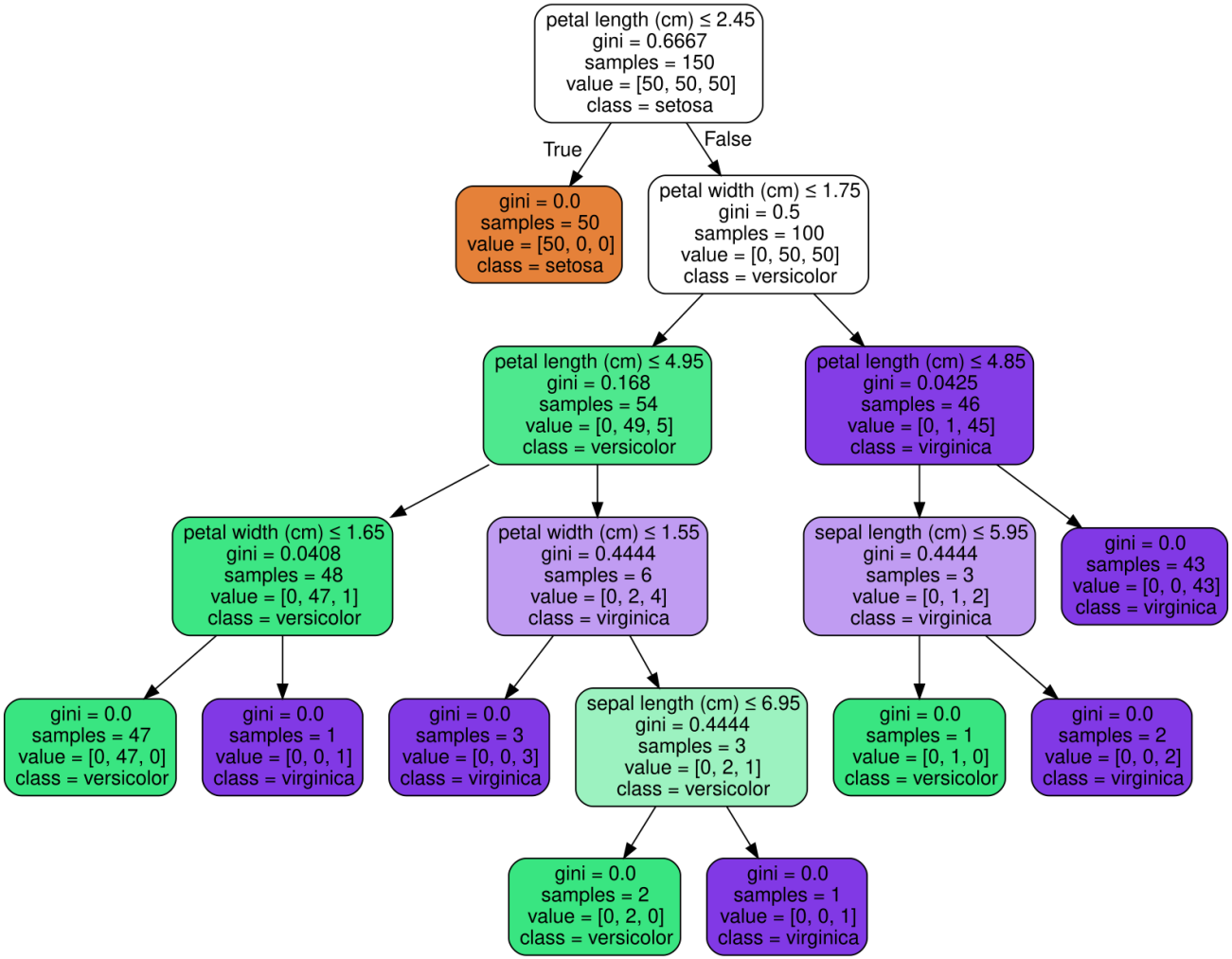


Soft margins

SVM summary

- Allows for kernels (linear, polynomial, Gaussian, etc.) $\rightarrow$ ideal for non-linearly separable data
- Can be tuned for "more robust inference" vs "more precise inference of boundary"
- Efficient when predicting: complexity proportional to number of support vectors.

Decision tree



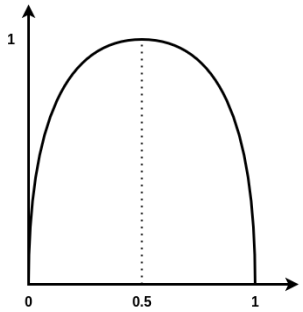
Decision tree

- Can be used for classification or regression
- Simple algorithm: recursively decide on a variable to split that minimizes the expectation of a loss in the subsequent leaves (regression: variance, classification: entropy of the outcome)

Decision tree example

Classification into two classes using entropy loss:

$$E = -P(\text{class 1}) \log(P(\text{class 1})) - P(\text{class 2})(\log P(\text{class 2}))$$



High entropy if "data is mixed".

Example

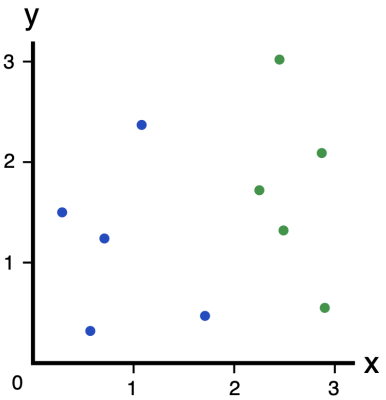
In a dataset with 20 elements, 14 are class 1 and 6 are class 2, the entropy can be computed as:

$$E = -\frac{14}{20} \log(\frac{14}{20}) - \frac{6}{20} \log(\frac{6}{20}) = 0.880$$

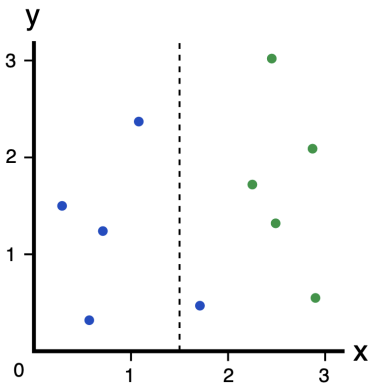
Decision tree example

Information Gain (IG) is the decrease in entropy after the dataset is split:

$$IG = E - E_{\text{split}}$$



The Dataset



An Imperfect Split

Before split (5 blue, 5 green): $E = -0.5\log(0.5) - 0.5\log(0.5) = 1.$

After the split: $E_{left} = 0$, $E_{right} = -\frac{1}{6}\log(\frac{1}{6}) - \frac{5}{6}\log(\frac{5}{6}) = 0.65$

$$E_{split} = 0.4 \cdot E_{left} + 0.6 \cdot E_{right} = 0.39$$

$$IG = 1 - 0.39 = 0.61$$

Decision tree

- Can be used for classification or regression
- Simple algorithm: recursively decide on a variable to split that minimizes the expectation of a loss in the subsequent leaves (regression: variance, classification: entropy / Gini impurity)

Issue

Imagine that the splits are partitioning the data in a half at each step of the inference algorithm. Can you foresee any issue?

Decision tree

- Can be used for classification or regression
- Simple algorithm: recursively decide on a variable to split that minimizes the expectation of a loss in the subsequent leaves (regression: variance, classification: entropy / Gini impurity)

Issue

Imagine that the splits are partitioning the data in a half at each step of the inference algorithm. Can you foresee any issue?
Overfitting (leaves are specialized for very few data points).

Decision tree

- Can be used for classification or regression
- Simple algorithm: recursively decide on a variable to split that minimizes the expectation of a loss in the subsequent leaves (regression: variance, classification: entropy / Gini impurity)

Issue

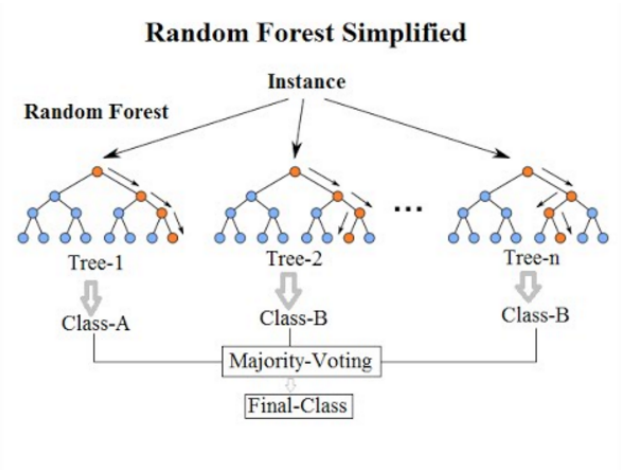
Imagine that the splits are partitioning the data in a half at each step of the inference algorithm. Can you foresee any issue?
Overfitting (leaves are specialized for very few data points).

Possible way out: random forests.

Random forest

Simple idea:

- bootstrap the training set and learn a tree on each bootstrapped set
- for a prediction, run all decision tree and aggregate with a majority vote



For free, we get also uncertainty measure by looking at the variance of the predictions in each decision tree.