Machine Learning

4. SVM

Yannick Le Cacheux

CentraleSupélec - Université Paris Saclay

September 2024

Outline

Reminder: limits of K-means

- K-means can be considered "linear" in that cluster boundaries are linear.

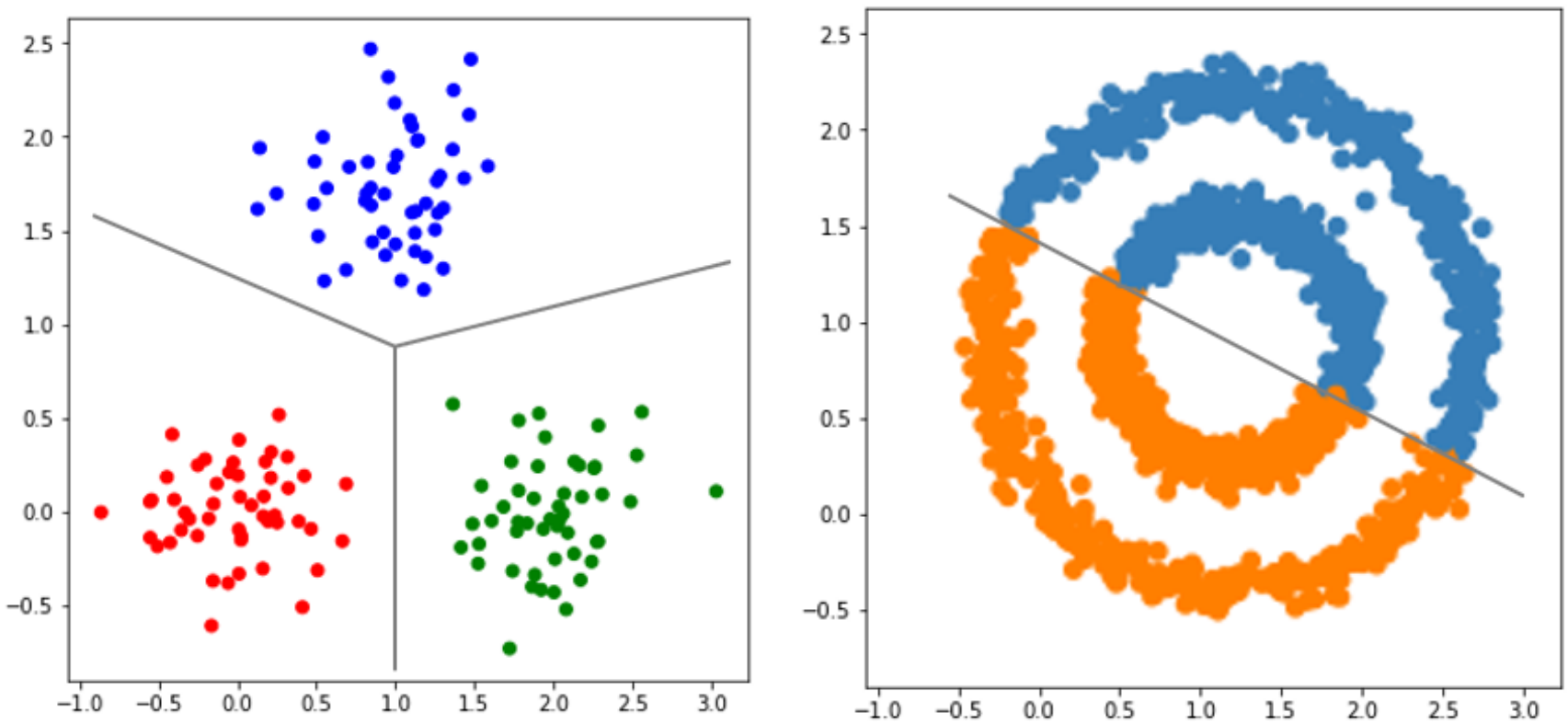
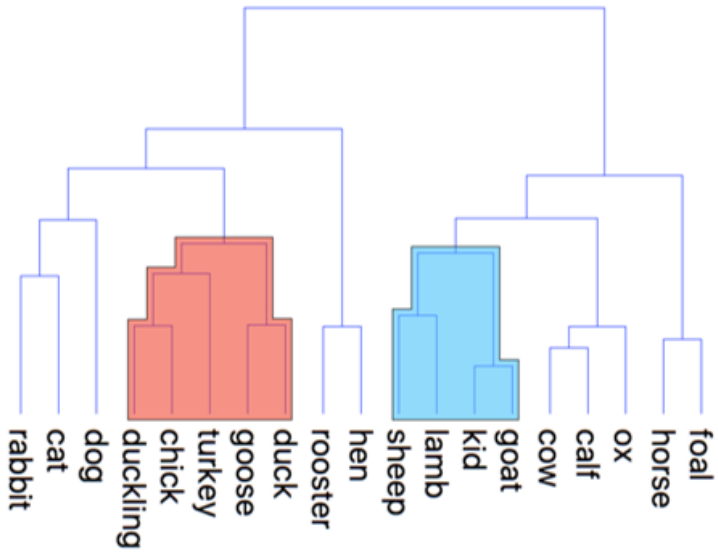


Figure: [1] Left: linearly separable clusters. Right: non-linearly separable clusters.

[1] From the scikit-learn documentation

Hierarchical clustering

- This is not a specific clustering method but more like a family of methods
  - Advantage: it is not necessary to define the number of clusters a priori (but is still has to be selected at some point)
  - We simply need two components: an inter-cluster distance (or dissimilarity) and an intra-cluster distance
  - If these two distances are properly defined, we can work with any type of object (for instance strings, bits...)



Hierarchical clustering

- Assign each point to its own cluster:

$$C_1 = \{\mathbf{x}_1\}, \ldots, C_N = \{\mathbf{x}_N\}$$

- Find the two clusters closest to each other:

$$C_i, C_j = argmin_{i,j} D(C_i, C_j)$$

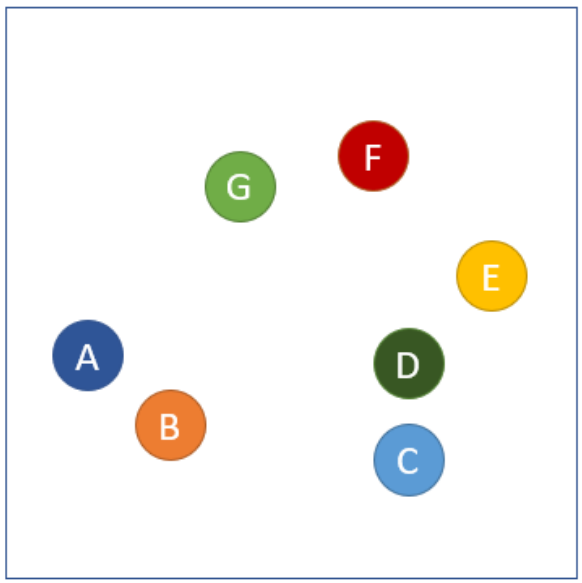
- Merge the two clusters, for instance
  - Update $C_i = C_i \cup C_j$
  - Remove C_j
  - (And keep track of these operations)

} Repeat

- until there is only one cluster $\mathcal{C}_1$

Example

- Intra-cluster distance: $d(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2$
- Inter-cluster distance: $D(C_i, C_j) = \min_{\mathbf{x}_i \in C_i, \mathbf{x}_j \in C_j} d(\mathbf{x}_i, \mathbf{x}_j)$



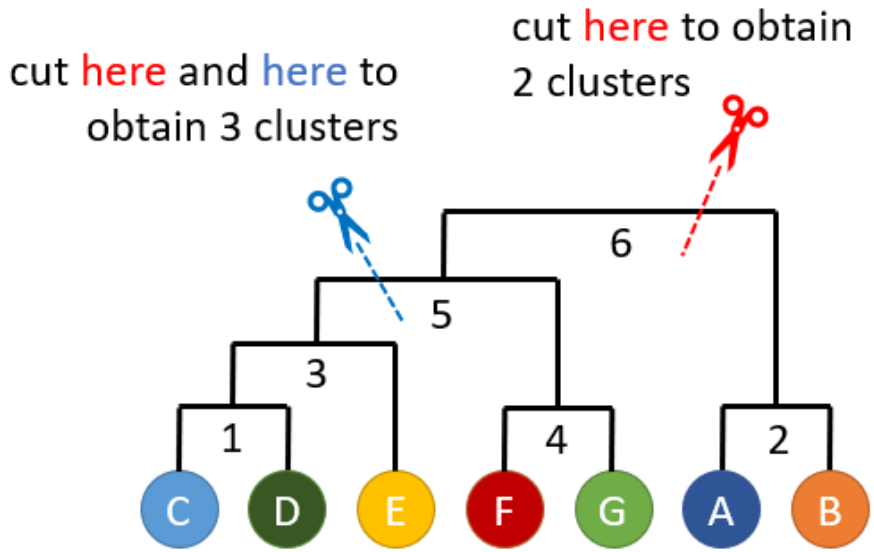


Figure: Clustering example with dendrogram

Possible group-wise distances

- Complete linkage: $D(C_i, C_j) = \max_{\mathbf{x}_i \in C_i, \mathbf{x}_j \in C_j} d(\mathbf{x}_i, \mathbf{x}_j)$
- Single linkage: $D(C_i, C_j) = \min_{\mathbf{x}_i \in C_i, \mathbf{x}_j \in C_j} d(\mathbf{x}_i, \mathbf{x}_j)$
- Average linkage: $\frac{1}{|C_i| \cdot |C_j|} \sum_{\mathbf{x}_i \in C_i} \sum_{\mathbf{x}_j \in C_j} d(\mathbf{x}_i, \mathbf{x}_j)$
- ...

Distances in general

- Desirable properties of distances / dissimilarities:
  - Symmetry: $d(\mathbf{a}, \mathbf{b}) = d(\mathbf{b}, \mathbf{a})$
  - Separability: $d(\mathbf{a}, \mathbf{b}) = 0$ iff $\mathbf{a} = \mathbf{b}$
  - Triangular inequality: $d(\mathbf{a}, \mathbf{c}) \leq d(\mathbf{a}, \mathbf{b}) + d(\mathbf{b}, \mathbf{c})$
- Some methods may still work if this is not the case, but you may get unwanted results

Illustration

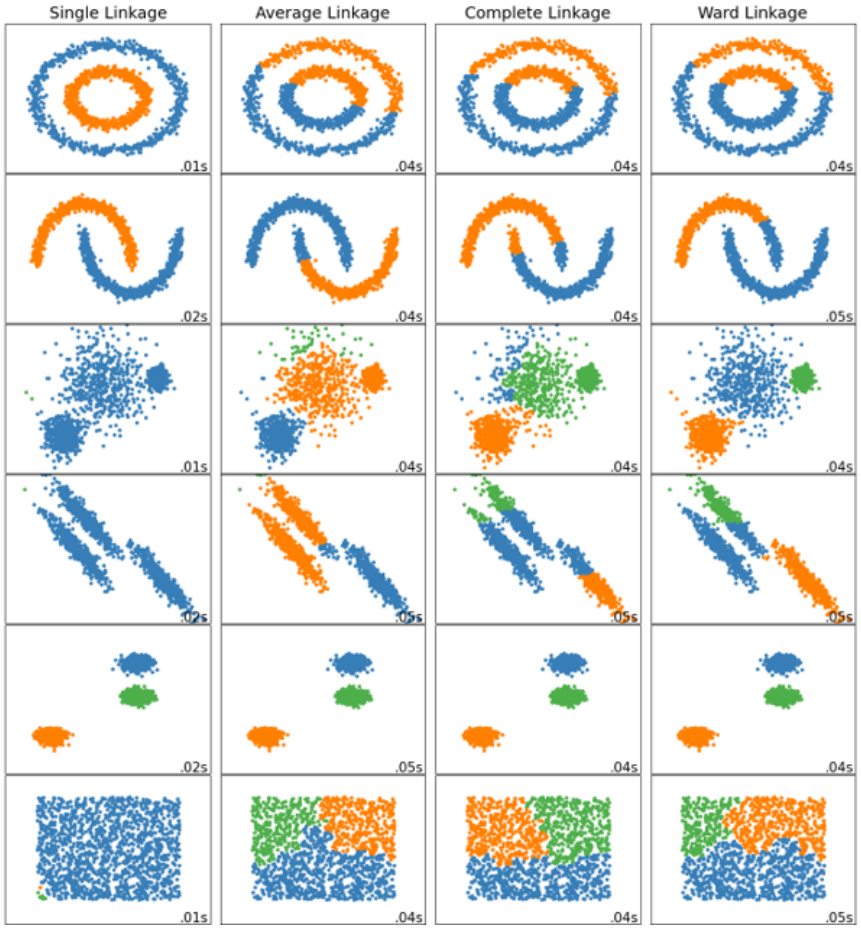


Figure: Illustration of different clustering methods (from sklearn doc)

DBSCAN

- Density Based Spatial Clustering of Applications with Noise (DBSCAN)
- Can automatically determine the number of clusters
- Can exclude some points from all clusters ("outliers")

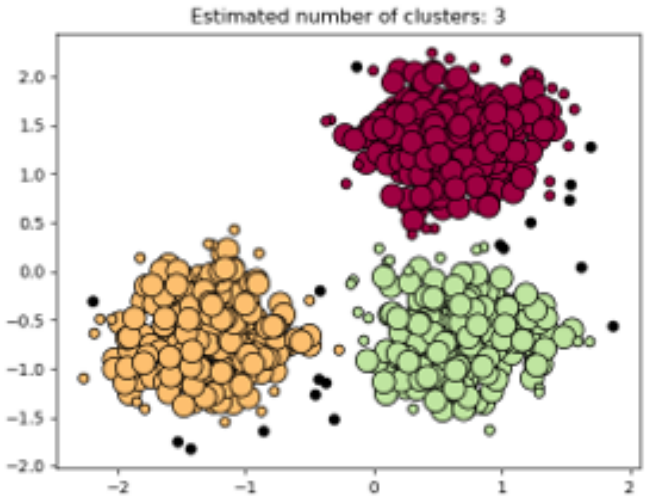


Figure: Result of DBSCAN

DBSCAN

- There are essentially two hyper-parameters: ϵ and P
- Points with at least P neighbors in a radius of ϵ are core points
- Points within a radius of ϵ of core points are neighbors
- Other points are outliers

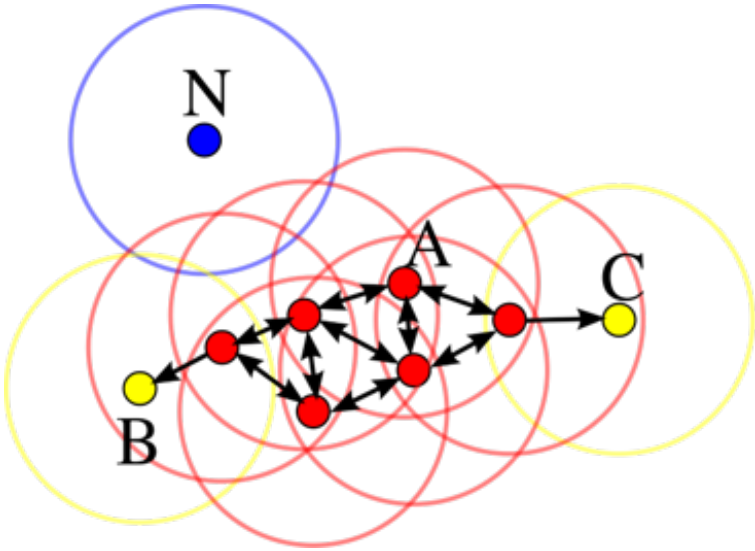


Figure: Core points, neighbors and outliers

Illustration of DBSCAN

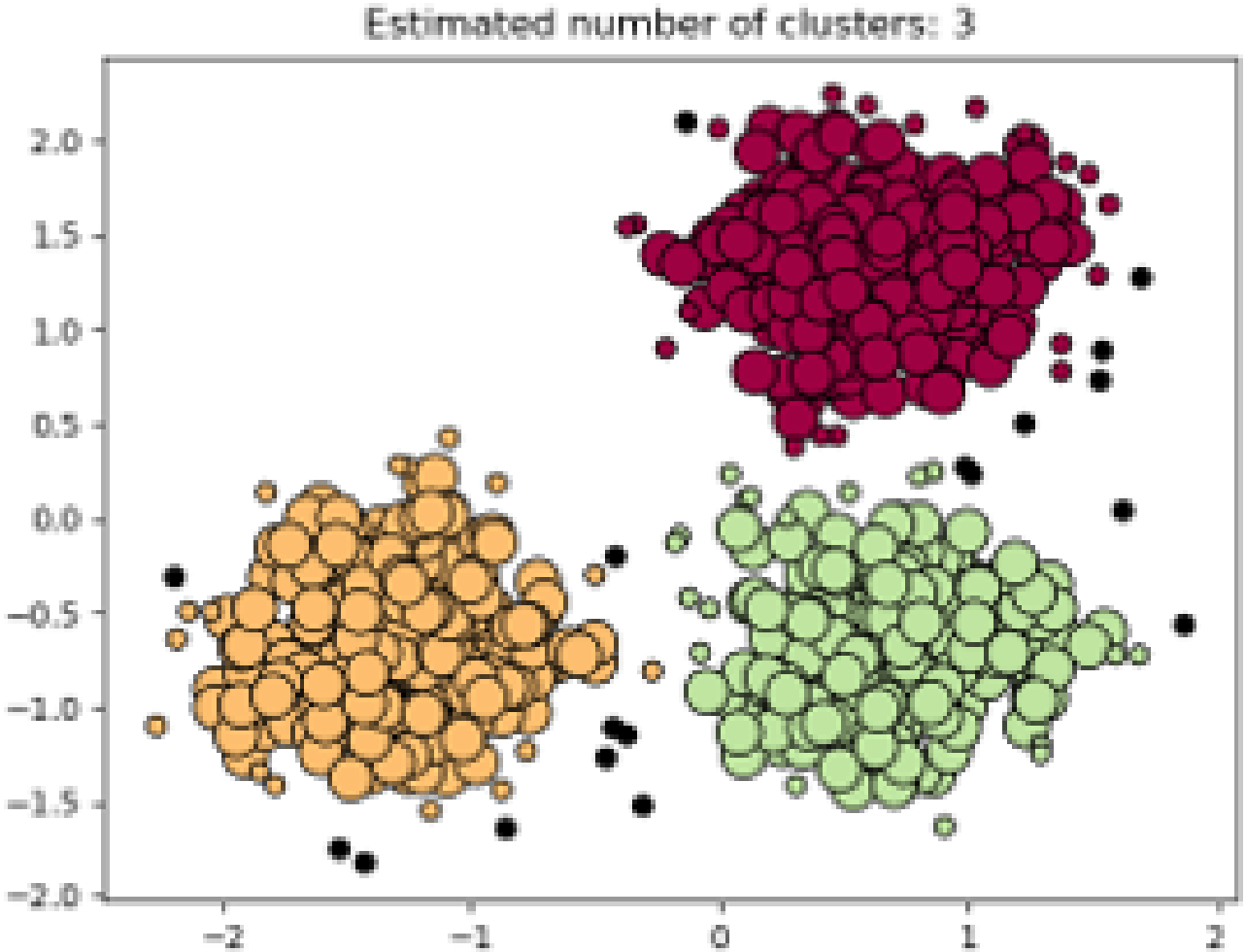
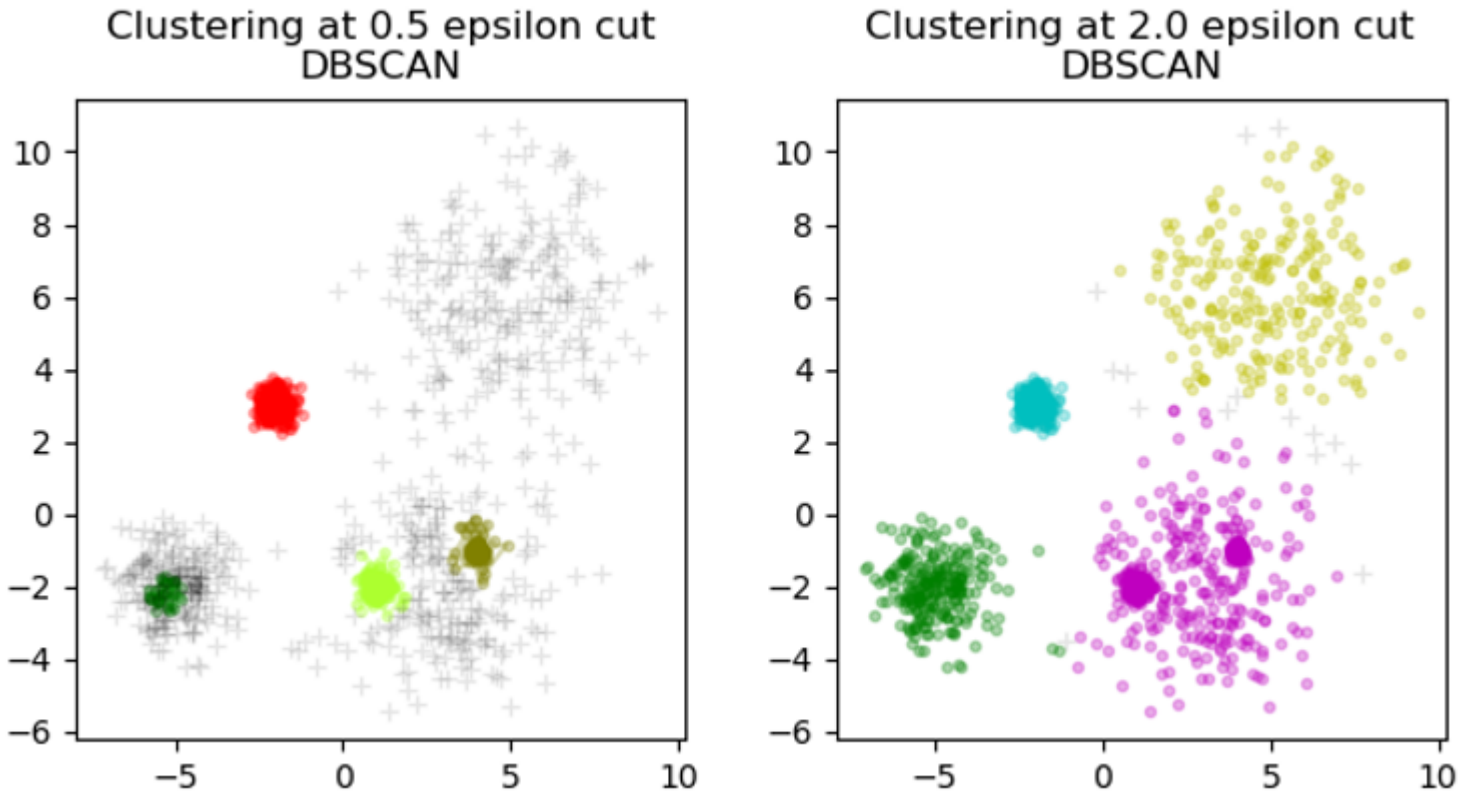


Figure: Illustration

Limits of DBSCAN

- May not work well if the density of points varies depending on areas



Other clustering algorithms

- **Expectation-Maximization**: can be seen as a generalization of K-Means allowing for "soft" (probabilistic) assignments to non isotropic clusters
- **OPTICS**: can be seen as a generalization of DBSCAN allowing for varying point densities
- **Spectral clustering**: performs dimensionality reduction on a pairwise affinity matrix, and performs clustering in lower dimension
- ...

Outline

Bayes theorem

- Formulation: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

This can be proven easily based on $P(A, B) = P(B, A)$ and $P(A, B) = P(A|B)P(B)$

- Application to machine learning:

$$P(\mathbf{w}|\mathcal{D}) = \frac{P(\mathcal{D}|\mathbf{w})P(\mathbf{w})}{P(\mathcal{D})}$$

- $\mathbf{w}$ corresponds to model parameters, $\mathcal{D}$ to training dataset

Posterior	Likelihood	Prior	Evidence
$P(\mathbf{w}\|\mathcal{D})$	$P(\mathcal{D}\|\mathbf{w})$	$P(\mathbf{w})$	$P(\mathcal{D})$

Naïve Bayes

- We are trying to predict the probability of label y based on input variables/features $x_1, \ldots, x_D$

$$P(y|x_1, \ldots, x_D) = \frac{P(y)P(x_1, \ldots, x_D|y)}{P(x_1, \ldots, x_D)}$$

- Assumption: all the x_i are (conditionally) independent:

$$P(x_i, x_j|y) = P(x_i|y)P(x_j|y)$$

- So $P(y|x_1, \ldots, x_D) = \frac{P(y)\prod_{i=1}^{D} P(x_i|y)}{P(x_1,\ldots,x_D)}$
- $P(x_1, \ldots, x_D)$ is constant for input $x_1, \ldots, x_D$ so

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{D} P(x_i|y)$$

Naïve Bayes with binary variables

$$\hat{y} = \arg\max_y P(y) \prod_{d=1}^{D} P(x_d|y)$$

- If we consider binary predictions $y \in \{0, 1\}$ and binary input variables $x_d \in \{0, 1\}$
  (where we write $P(y) = P(y = 1)$)
- We can estimate $P(y)$ and $P(x_d|y)$ with

$$P(y) = P(y = 1) = \frac{\sum_{n=1}^{N} y_n}{N}$$

$$P(x_d|y) = \frac{1}{N} \sum_{\substack{n \\ y_n=1}} x_{n,d}$$

$x_{n,d}$ is the d^{th} feature of the n^{th} training sample

Naïve Bayes

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{D} P(x_d|y)$$

- More generally, naïve Bayes is a category of methods.
- We can build different models with different probability distributions
- For instance, we can assume that x_i is continuous and a conditional gaussian:

$$P(x_d|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_d - \mu_y)^2}{2\sigma_y^2}\right)$$

Outline

Decision tree

- General idea: we create a decision tree by iteratively splitting the dataset based on one or several criteria

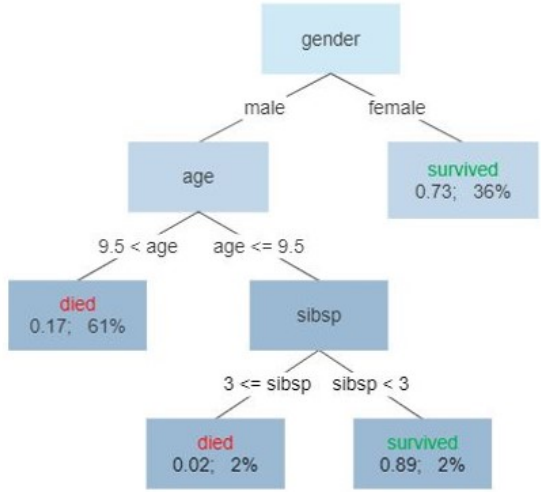


Figure: Example: survival of Titanic passenger (from Wikipedia)

Decision tree

- The decision tree is built recursively: we start with the whole dataset, split the dataset into two parts, and repeat the process for the two subparts as well as further subparts

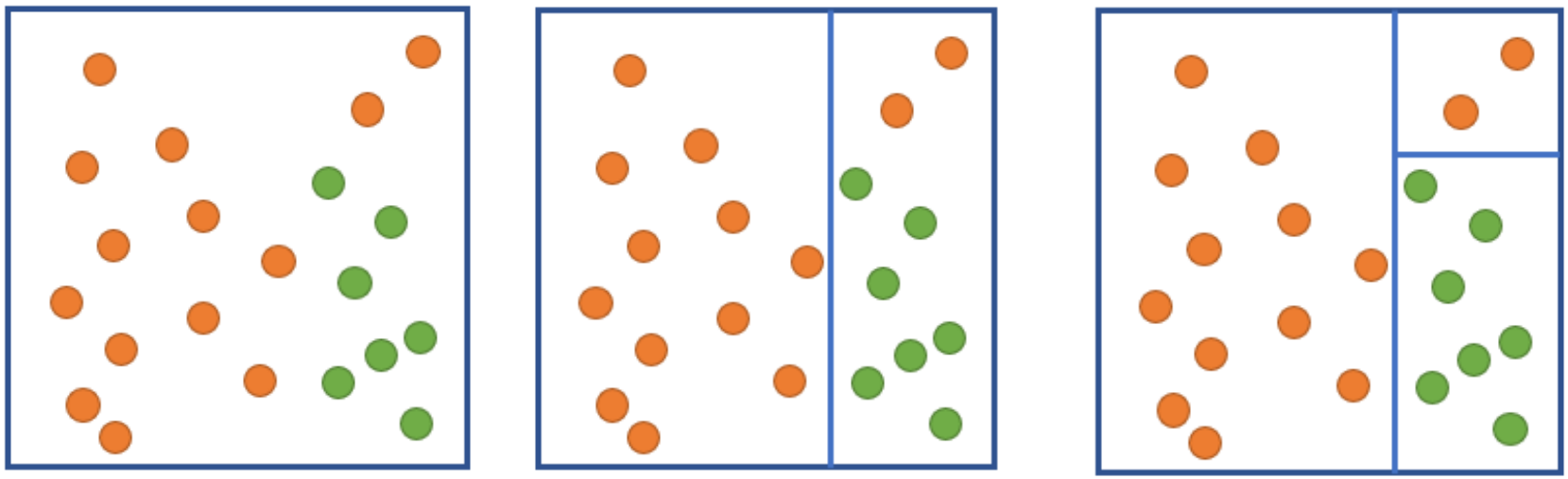


Figure: Iteratively building a tree

Decision tree

At each split, we need to decide:

- Which variable to consider (age, salary, gender...)
- How to split data with respect to this variable
  - Age > 10 ? Age > 15 ? Age > 70?...

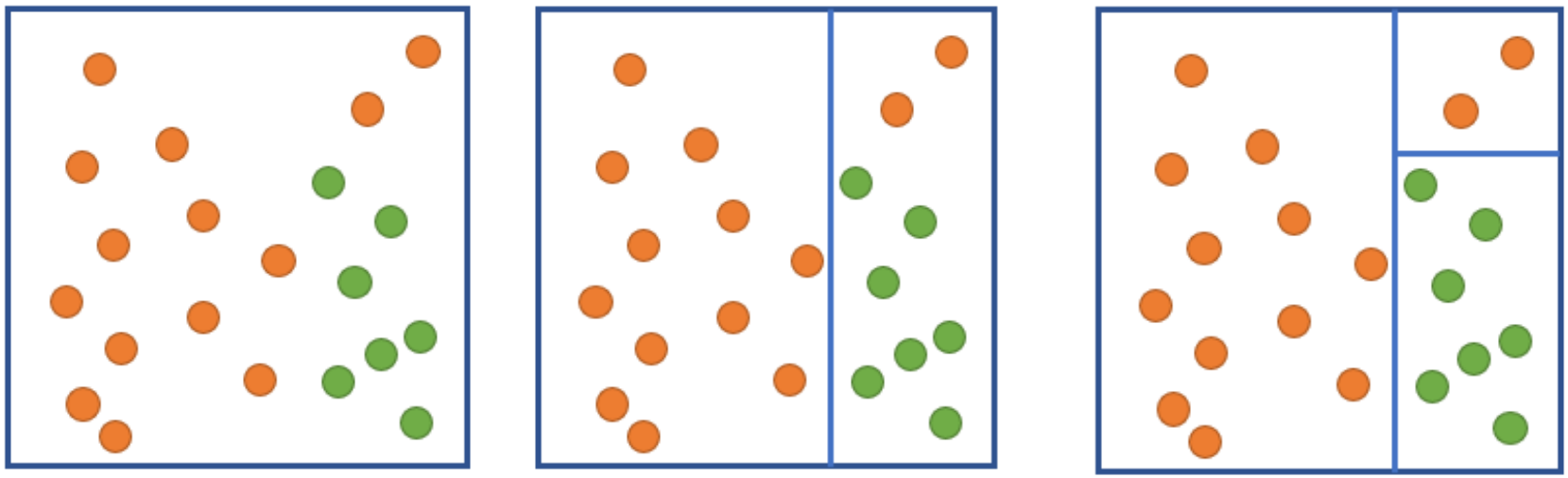


Figure: Iteratively building a tree

Decision tree

- We need a criterion to decide how "good" a split is
-
- One possibility: Shannon's entropy:

$$H[P] = -\sum_i P(i) \log_2 P(i)$$

H measures the "average amount of information" or uncertainty in a probability distribution P. We want low entropy in each split.

- Other possibility: Gini impurity

$$G[P] = \sum_i P(i)(1 - P(i))$$

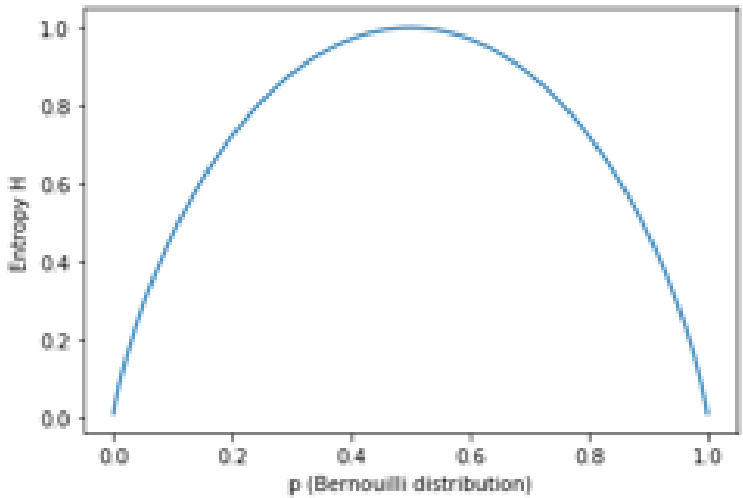


Figure: Entropy function

Tree example

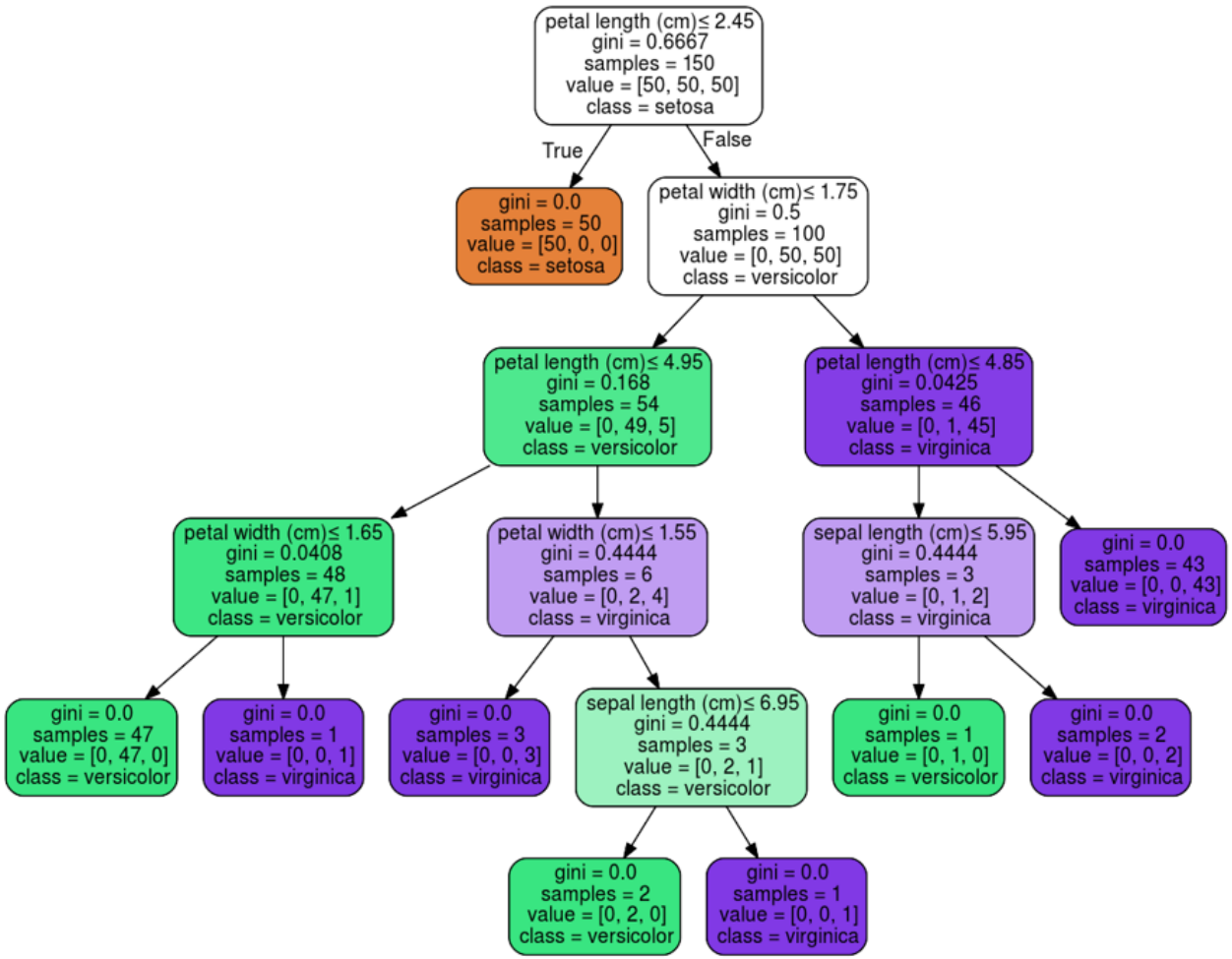
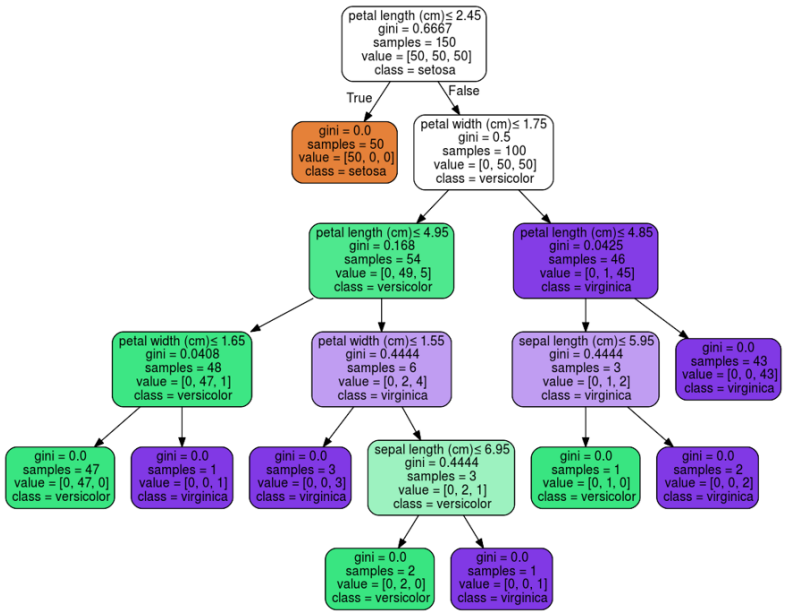


Figure: Example: decision tree for Iris dataset (from scikit-learn doc)

Hyper-parameters

Possible hyper-parameters

- Maximum recursive depth of the tree
- Minimum number of samples per leave
- ...



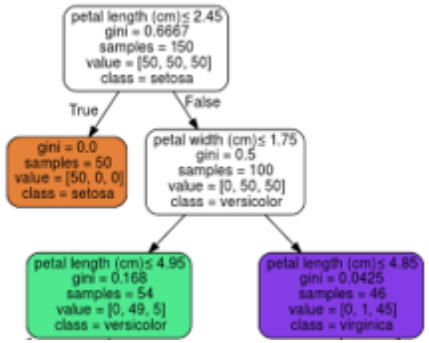


Figure: Trees with depth 5 (left) and 2 (right)

Advantages of decision tree

- The model is easy to understand and interpret in terms of overfitting
- It is simple to deal with categorical variables (for instance gender, eye color, town...)
- Interpretation of the model is intuitive
- Even though a single decision tree may not be very flexible, several trees can be combined to form very powerful learning algorithms (for example in a random forest, cf. future class on ensemble learning).

Outline

Best linear separation

- We assume we have two linearly separable classes
- Which classification boundary seems better?

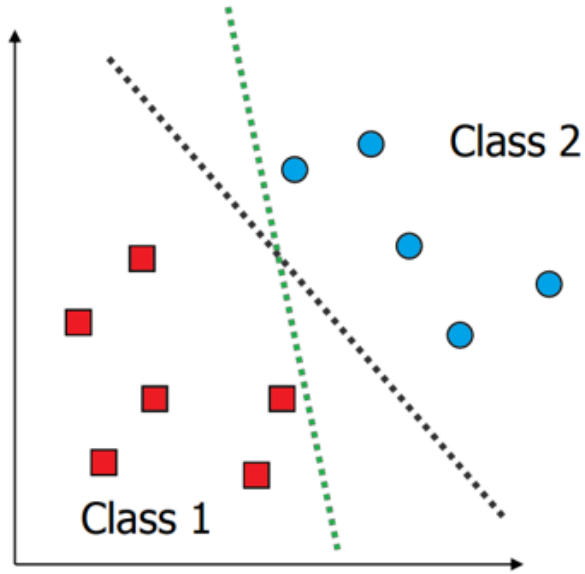


Figure: Linearly separable classes

Best linear separation

- We assume we have two linearly separable classes
- Which classification boundary seems better?
  - If we add some data or slightly move training points, which decision function is more likely to produce errors?
- We want to maximize the margin

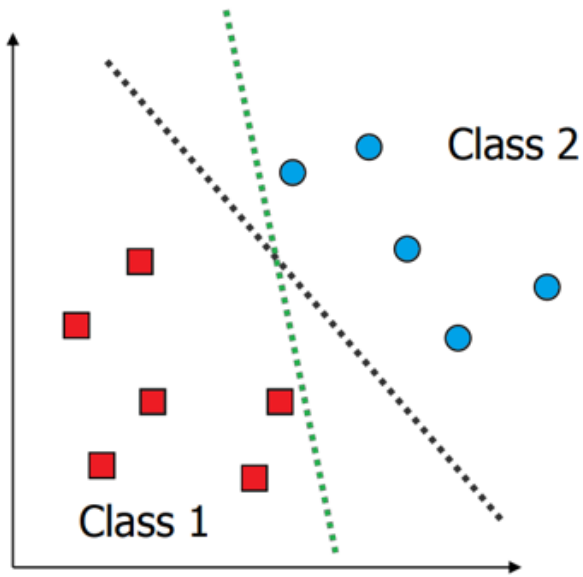


Figure: Linearly separable classes

Maximum margin

- The linear decision boundary can be written $\mathbf{w}^\top \mathbf{x} + w_0 = 0$
  - $\mathbf{w}$ is orthogonal to the decision boundary
- The dotted lines are the farthest (parallel) lines from the decision boundary s.t. no training points are inside this area

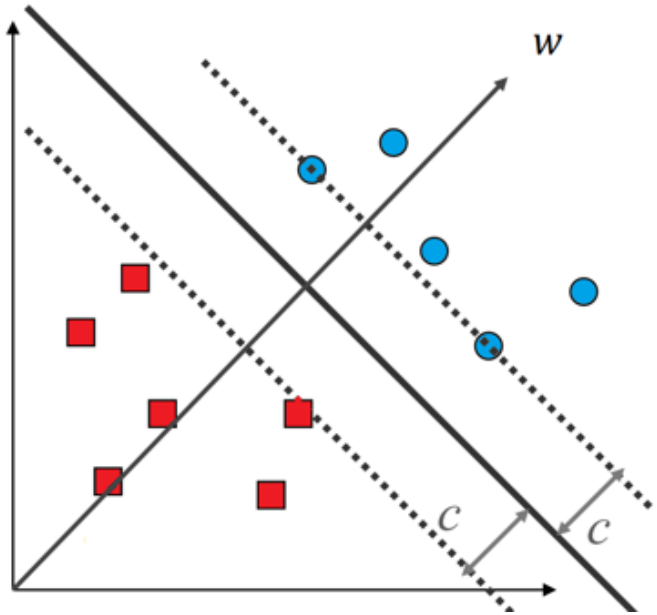


Figure: Maximum margin separation

Maximum margin

- We can write this in terms of constraints
- Assuming labels $y = +1$ and $y = -1$
  - For $\mathbf{x}$ in class 1: $\mathbf{w}^\top \mathbf{x} + w_0 \leq -c$
  - For $\mathbf{x}$ in class 2: $\mathbf{w}^\top \mathbf{x} + w_0 \geq c$
- This is equivalent to

$$(\mathbf{w}^\top \mathbf{x} + w_0) y \geq c$$

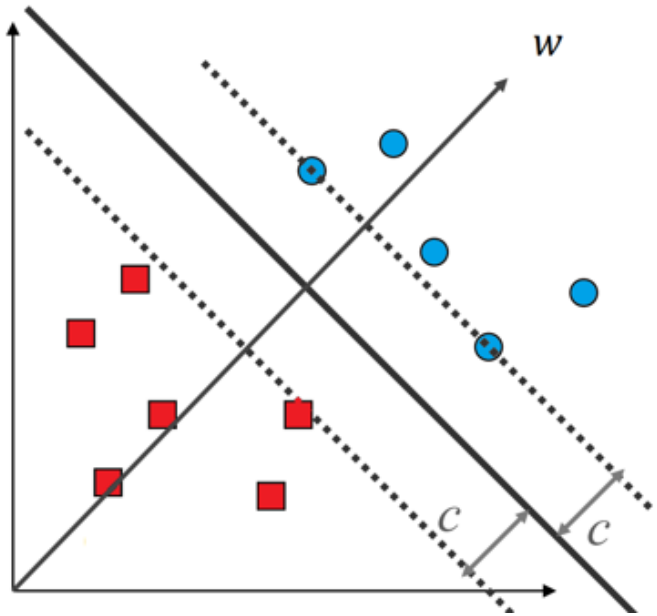


Figure: Maximum margin separation

Maximum margin

- If we take points $\mathbf{x}_i$ and $\mathbf{x}_j$ on the dotted lines, the margin is

$$m = \mathbf{w}^\top (\mathbf{x}_i - \mathbf{x}_j) = 2c$$

- Since $\mathbf{w}$ can be scaled arbitrarily, we consider the "normalized" margin

$$m = \frac{2c}{\|\mathbf{w}\|_2}$$

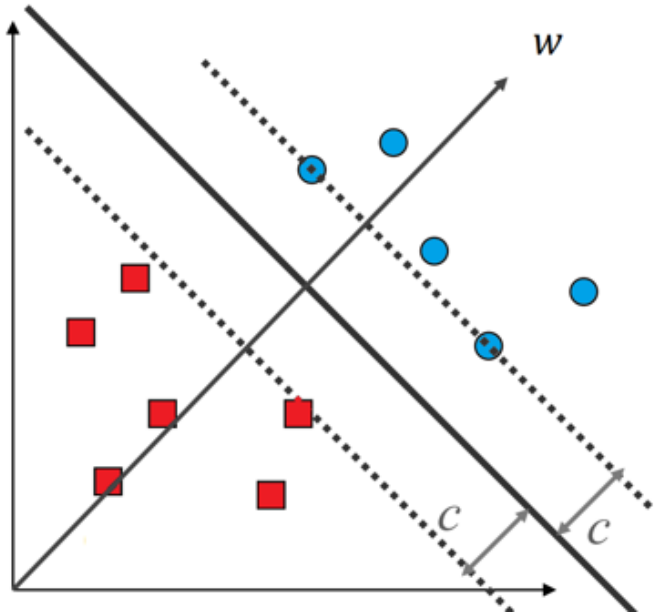


Figure: Maximum margin separation

Objective formulation

- We want to solve the following constrained optimization problem:

$$\max_{\mathbf{w},w_0} \frac{2c}{\|\mathbf{w}\|} \text{ such that } (\mathbf{w}^\top \mathbf{x}_n + w_0) y_n \geq c \quad \forall n$$

- c just scales $\mathbf{w}$ and w_0, so we set $c = 1$
- Our objective is thus equivalent to

$$\min_{\mathbf{w},w_0} \|\mathbf{w}\| \text{ such that } (\mathbf{w}^\top \mathbf{x}_n + w_0) y_n \geq 1$$

Minimization of objective

Objective

$$\min_{\mathbf{w},w_0} \|\mathbf{w}\| \quad \text{such that} \quad (\mathbf{w}^\top \mathbf{x}_n + w_0) y_n \geq 1$$

- Without going into details for now, we can introduce Lagrange multipliers to take the constraint into account
- We obtain a constrained quadratic programming formulation, which has a convex form and thus a global minimum
- This minimum can be found with gradient descent or similar methods

Predictions

- Predictions can be made with

$$\text{sign}(\mathbf{w}^\top \mathbf{x}_n + w_0)$$

- Only some vectors are actually important for the constraints
- These are called the support vectors (hence the name "support vector machine")
- What if the points are not linearly separable?

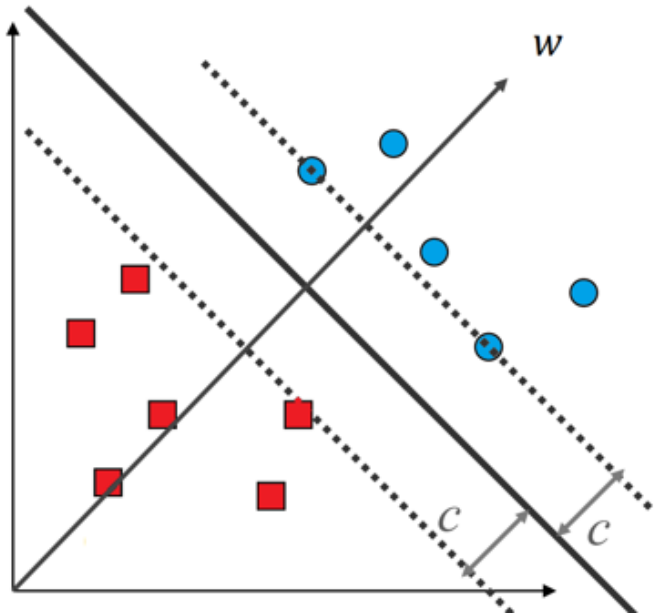


Figure: Making predictions

Hard vs. soft version

- We can relax the constraints to allow some points to violate the margin constraint

$$\min_{\mathbf{w},w_0,\boldsymbol{\zeta}} \|\mathbf{w}\| + C\sum_{n=1}^{N} \zeta_n$$

such that

$$(\mathbf{w}^\top \mathbf{x}_n + w_0) y_n \geq 1 - \zeta_n$$

$$\zeta_n \geq 0$$

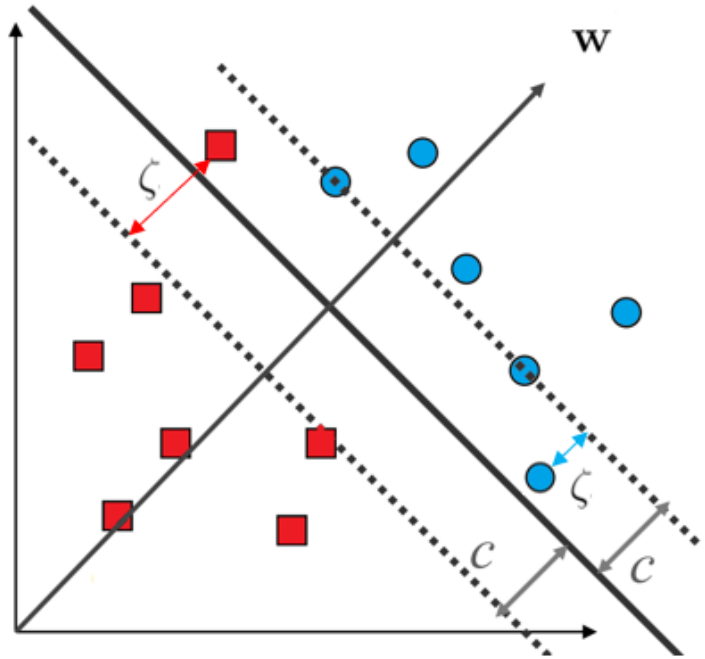


Figure: Allowing some points to violate the constraint

Non linear patterns

- SVM mostly makes sense if applied to an (approximately) linearly separable dataset
- What can we do if the decision boundary is not linear?
- We could introduce polynomial features etc
- The idea is to map training data into a higher dimensional space such that the decision boundary is linear
- Problem: this is not computationally efficient with SVM

The dual problem of SVM

- To understand why, we will need to briefly provide more details about the optimization of the cost function
- "Hard" margin objective: $\min_{\mathbf{w},w_0} \|\mathbf{w}\|$ s.t. $(\mathbf{w}^\top \mathbf{x}_n + w_0)y_n \geq 1$
- Introducing Lagrange multipliers:

$$\min_{\mathbf{w},w_0,\boldsymbol{\lambda}} \frac{1}{2}\mathbf{w}^\top \mathbf{w} + \sum_{n=1}^{N} \lambda_n (1 - (\mathbf{w}^\top \mathbf{x}_n + w_0)y_n)$$

- Setting the partial derivative with respect to $\mathbf{w}$ to 0, we could show that this is equivalent to solving the dual problem

$$\sum_{n=1}^{N} \lambda_n - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \text{ s.t. } \lambda_n \geq 0 \text{ and } \sum_n \lambda_n y_n = 0$$

- If we write $\phi(\mathbf{x})$ the projection of sample x in the high dimension, we need to compute many times the inner product $\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$

Kernel trick

- We can use what is called the "kernel trick": we can find suitable kernels $k(\cdot, \cdot)$ such that we don't need to explicitly map the points in higher dimension to compute the inner product $\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$
- For example:

$$\phi(\mathbf{x})^\top \phi(\mathbf{y}) = \begin{pmatrix} x_1^2 \\ x_1 x_2 \\ x_2^2 \\ x_2 x_1 \end{pmatrix}^\top \begin{pmatrix} y_1^2 \\ y_1 y_2 \\ y_2^2 \\ y_2 y_1 \end{pmatrix}$$

$$= x_1^2 y_1^2 + 2 x_1 x_2 y_1 y_2 + x_2^2 y_2^2$$

$$= (x_1 y_1 + x_2 y_2)^2 = (\mathbf{x}^\top \mathbf{y})^2$$

- More generally, for a "polynomial combinations" of degree d:

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\top \mathbf{y})^d = \phi(\mathbf{x})^\top \phi(\mathbf{y})$$

Kernel trick

- A function $k(\mathbf{x}_1, \mathbf{x}_2)$ is said to be a kernel if there is a function ϕ such that $k(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1)^\top \phi(\mathbf{x}_2)$
- We don't actually need to explicitly find the function ϕ: a function k is a kernel iff the Gram matrix whose elements are $k(\mathbf{x}_i, \mathbf{x}_j)$ is positive semidefinite.
- The use of kernel functions even enables to map input data to infinite dimension spaces
- We can also use the kernel trick with other learning algorithms, for example ridge regression or PCA

Choices of kernel

- Given these elements, we can build many different kernels:
  - Polynomial of degree d:

$$k(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1^\top \mathbf{x}_2)^d$$

  - Polynomial of degree up to d:

$$k(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1^\top \mathbf{x}_2 + c)^d$$

  - Exponential kernel (infinite degree polynomials):

$$k(\mathbf{x}_1, \mathbf{x}_2) = \exp(s \cdot \mathbf{x}_1^\top \mathbf{x}_2)$$

  - Gaussian Radial Basis Function (RBF):

$$k(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|^2}{2\sigma^2}\right)$$

  - ...

Kernel illustrations

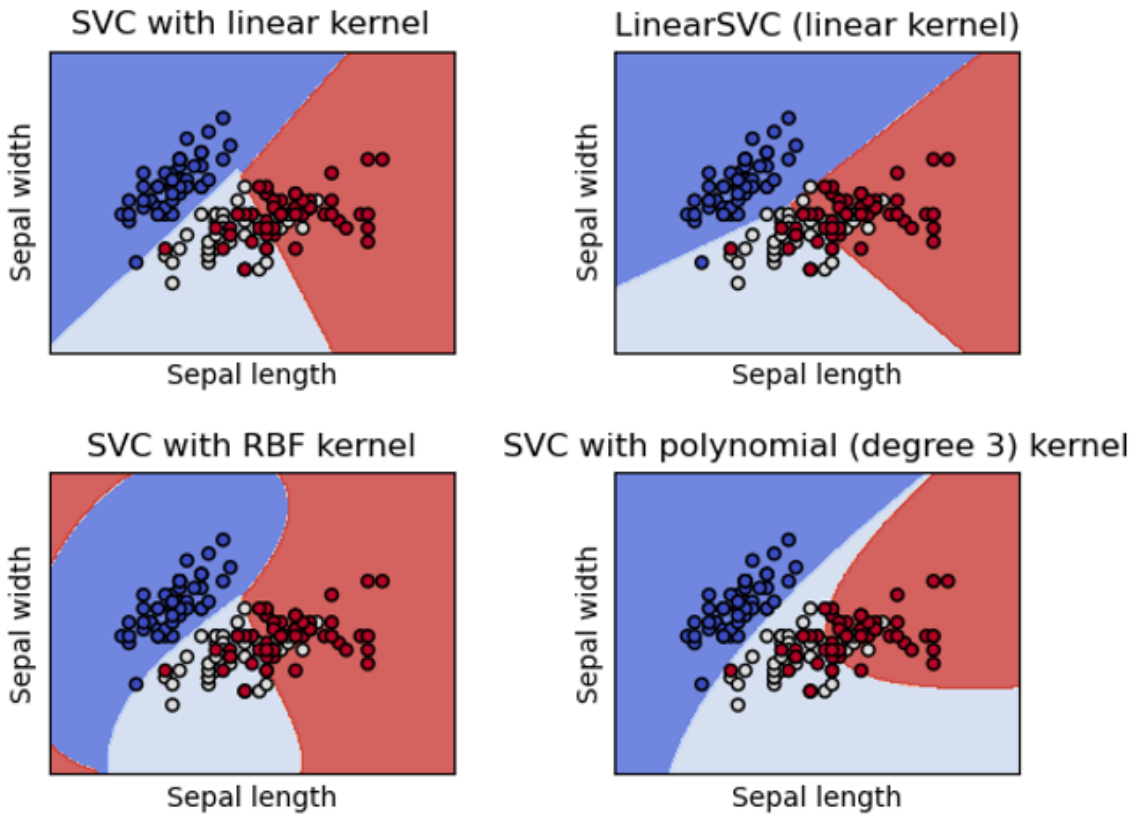


Figure: Decision boundaries with different kernels for 3-class classification on the Iris dataset (from Scikit-learn doc)

RBF illustration

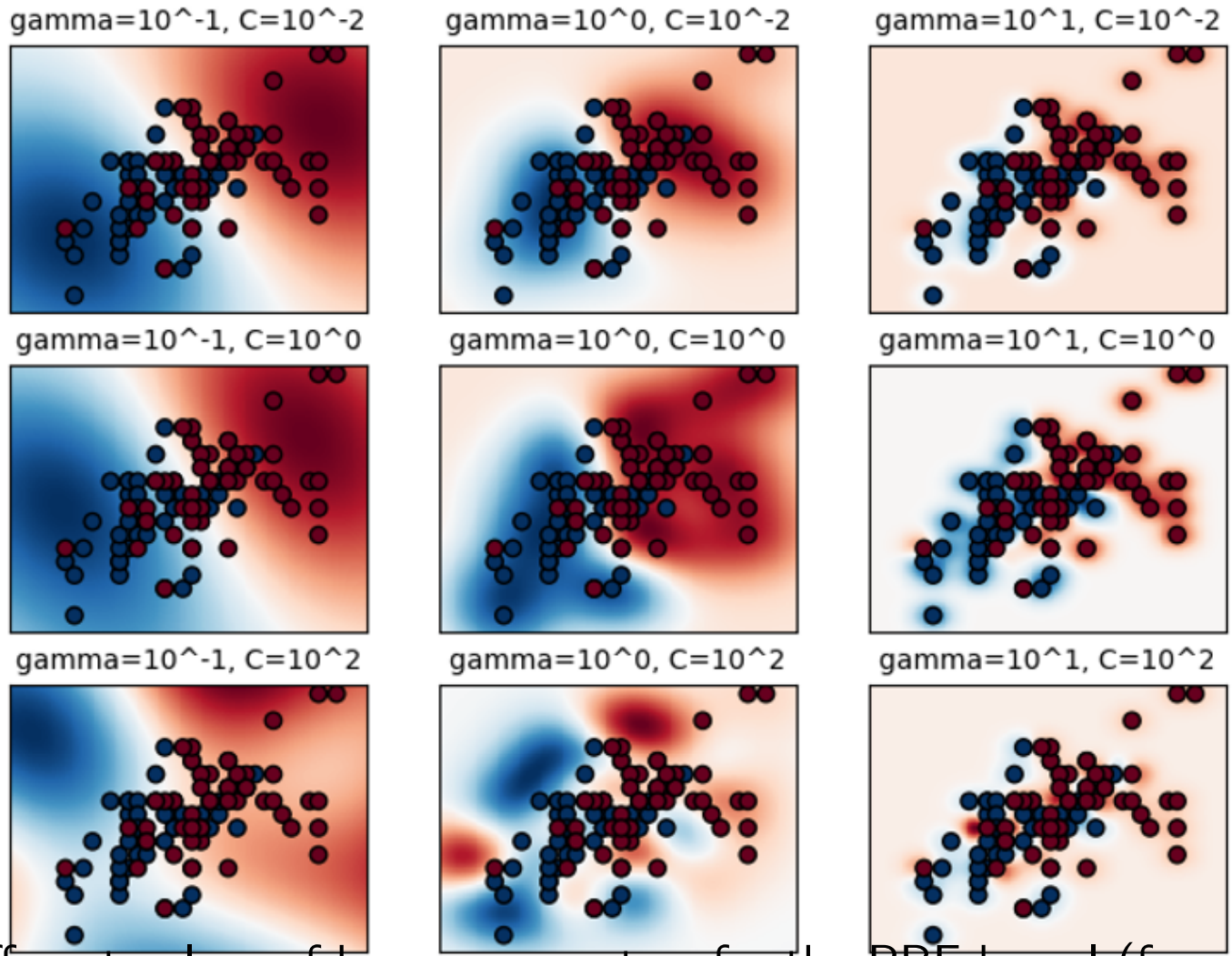


Figure: Different values of hyper-parameters for the RBF kernel (from scikit-learn doc)