Deep Learning for Image Analysis - Lecture 1: An introduction to Machine Learning

Thomas Walter, PhD

Centre for Computational Biology (CBIO)
MINES Paris-Tech, PSL Research University
Institut Curie, PSL Research University
INSERM U900

Overview

Overview

Definition of Artificial Intelligence and Machine Learning

- The definition of the term **intelligence** is highly controversial. Usually, one understands by intelligence the capacity of an individual to reason logically, to understand complexity, to learn more or less abstract concepts, to plan and to solve problems in varying conditions.
- **Artificial intelligence (AI)** is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans and other animals.
- AI effect: *"AI is whatever hasn't been done yet"*.
- In 1956, AI became a field of research. AI can be broken down into many subfields: knowledge representation, planning, natural language processing, object manipulation (robotics), machine learning, ...
- **Machine Learning** is concerned with the technology that enables computer programs to improve their performance at a certain task by experience.

A short history of artificial intelligence

- The dream to create machines that can think and act has been present in literature and mythology since antiquity (e.g. the myth of *"Talos"* or *"Rossum's Universal Robots"* by Karel Čapek, 1920).
- AI as a scientific discipline took its beginning with the publication of Alan Turing in 1950 (Turing test). The question *"Can machines think?"* was asked scientifically.
- Other theoretical bases were developed by Wiener (cybernetics) and Shannon (information theory).
- The term *"artificial intelligence"* was cornered in 1956, in the famous Dartmouth conference, where AI has been established as a field.

A short history of machine learning

Figure: The Mark I Perceptron

- The theoretical foundations go back to the early 19th century (e.g. Bayesian theory and Least Squares).
- Even before machines came into play, there was a lot of interest in finding methods to derive rules from data (*data fitting*). These methods are part of what we call Machine Learning today (linear regression, Bayesian theory, Logistic regression, Linear discriminant analysis, Markov chains).
- In 1958, Rosenblatt published his *Perceptron* (basically a linear classifier), which is seen as the predecessor of Neural Networks.

Machine Learning: basic definitions

- Machine Learning aims at predicting some output y from an input (or measurement) x:

$$y = f(x) \tag{1}$$

- In this formulation, Machine Learning aims at finding (learning) f from available data.
- The data that is used to learn f is called **training set**.
- In this general formulation, there is no particular limitation as to the mathematical nature of x and y. In many cases x is a P-dimensional vector and y a categorical or continuous output variable, but there are other settings, where x and / or y are more complicated objects, such as images or graphs.

Different settings in Machine Learning

	Supervised	Unsupervised
y discrete	Classification	Clustering
y continuous	Regression	Dimensionality reduction

- In **supervised learning**, the training data contains both measurements x_i and the corresponding output variables y_i. Together, they build the training set T:

$$T = \{(x_i, y_i)\}_{i=1,\ldots,N} \tag{2}$$

- In **unsupervised learning**, there are no annotations y_i. We aim at inferring **patterns** from the data (clusters, latent variables).

Classification vs. Clustering

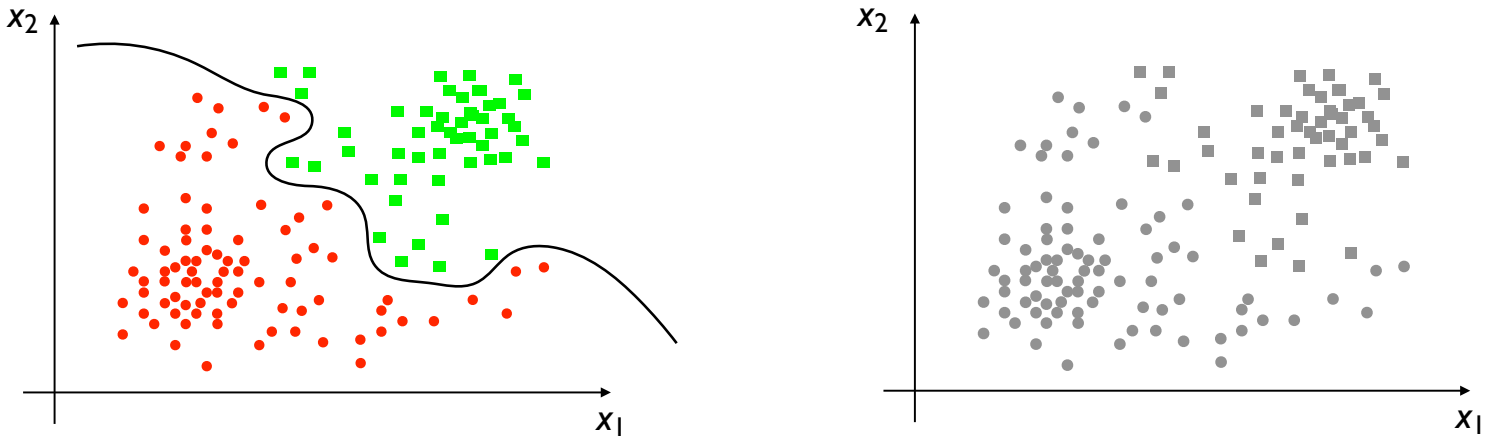
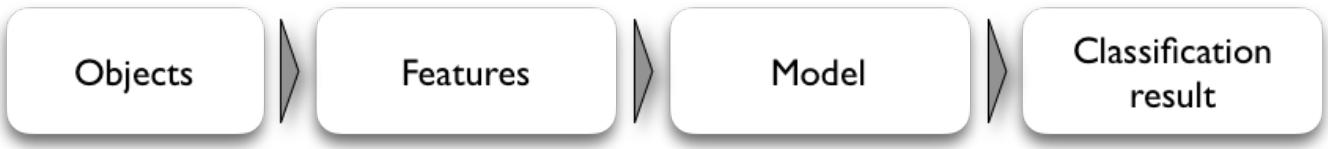


Figure: Supervised and unsupervised learning

- In **supervised learning**, we start from annotated data (here annotation is illustrated by the color), and we wish to learn a decision boundary that allows us to tell these classes apart.
- In **unsupervised learning**, we start with a point clouds and we wish to identify classes directly from their distribution.

Training and prediction

(a) Training a classifier



(b) Prediction

- Training: to learn a model from the training set. Depending on the method, this can take minutes, hours or days.
- Prediction: to apply the learned classifier to new data. This is usually computationally efficient.

Objects and features

- Machine learning typically deals with objects outside the mathematical world (emails, images, genomes, cars, ...).
- The first step is therefore to find a suitable representation of the objects.
  - **feature engineering**: finding descriptors according to existing domain knowledge
  - **representation learning**: learning the descriptors together with the classifier
- In many cases the objects can be represented by a P-dimensional vector of features (or descriptors): $\mathbf{x} \in \mathbb{R}^P$.
- It can be convenient to map a feature vector to a higher dimensional space:

$$\boldsymbol{\phi} : \mathbb{R}^P \rightarrow \mathbb{R}^Q \tag{3}$$

$$\mathbf{x} \rightarrow \boldsymbol{\phi}(\mathbf{x}) \tag{4}$$

Training set and design matrix

- In the frequent case that objects can be described by a feature vector $\mathbf{x} \in \mathbb{R}^P$, we can represent the training set $T = \{(x_i, y_i)\}_{i=1,\ldots,N}$ by a $N \times P$ **design matrix** $\mathbf{X}$ and an output vector y.
- In the design matrix, each row corresponds to one sample, each column corresponds to one feature:

	feature 1	feature 2	...	feature P
Sample 1	0.23	1.30	...	0.01
Sample 2	0.42	1.15	...	-0.23
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Sample N	0.31	1.53	...	0.33

Example: classification of flowers

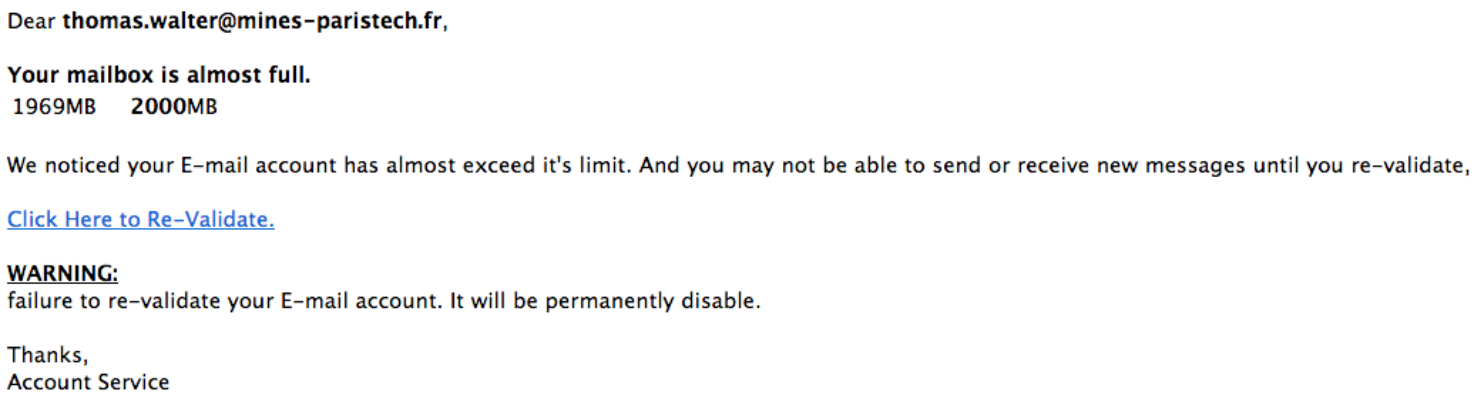
(a) Iris setosa

(b) Iris versicolor

(c) Iris virginica

- One of the oldest data sets in machine learning is the Iris data set, that was collected by the statistician and biologist Ronald Fisher in 1936 [Fisher, 1936].
- There are 3 classes (different types of the Iris flower). For each class, there were 50 samples collected. For each sample, 4 characteristics were measured (lengths and widths of different parts of the plants).
- The task is thus to learn a rule to predict the type of the flower (y) from a 4-dimensional vector x of geometric measurements.

Example: classification of SPAM emails



Dear **thomas.walter@mines-paristech.fr**,

Your mailbox is almost full.
1969MB **2000MB**

We noticed your E-mail account has almost exceed it's limit. And you may not be able to send or receive new messages until you re-validate,

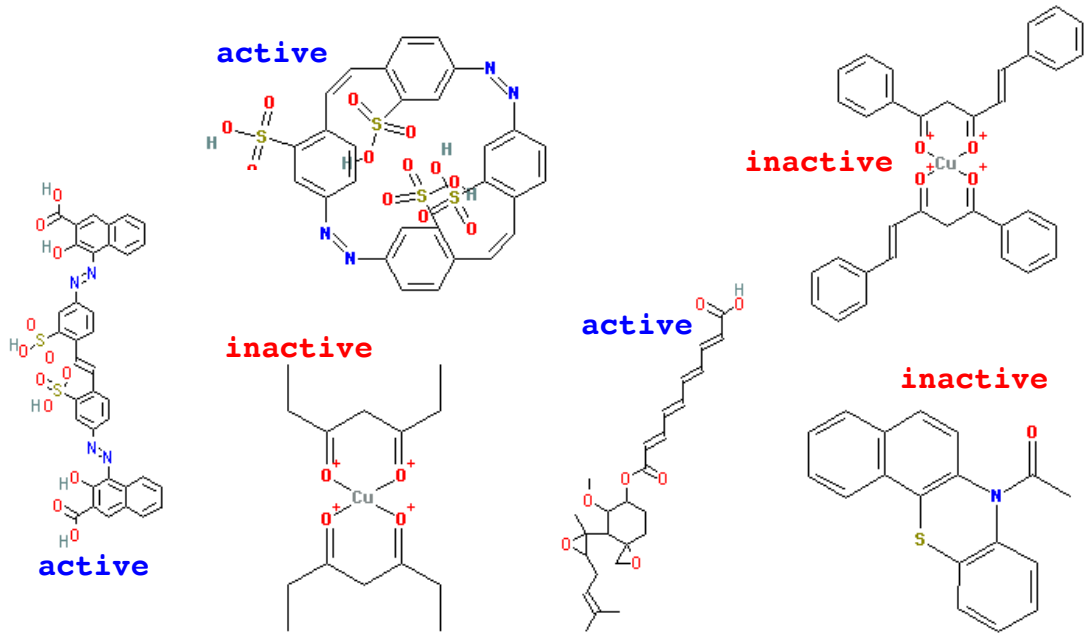
Click Here to Re-Validate.

WARNING:
failure to re-validate your E-mail account. It will be permanently disable.

Thanks,
Account Service

- This is a binary classification problem: $y \in \{0, 1\}$ (0: junk, 1: normal).
- The features can be constructed in the following way: for each email annotated by the user, the words are listed. An email is described as a vector of frequencies of these words.
- The system learns then a function that assigns to each vector of measured word frequencies the label y.

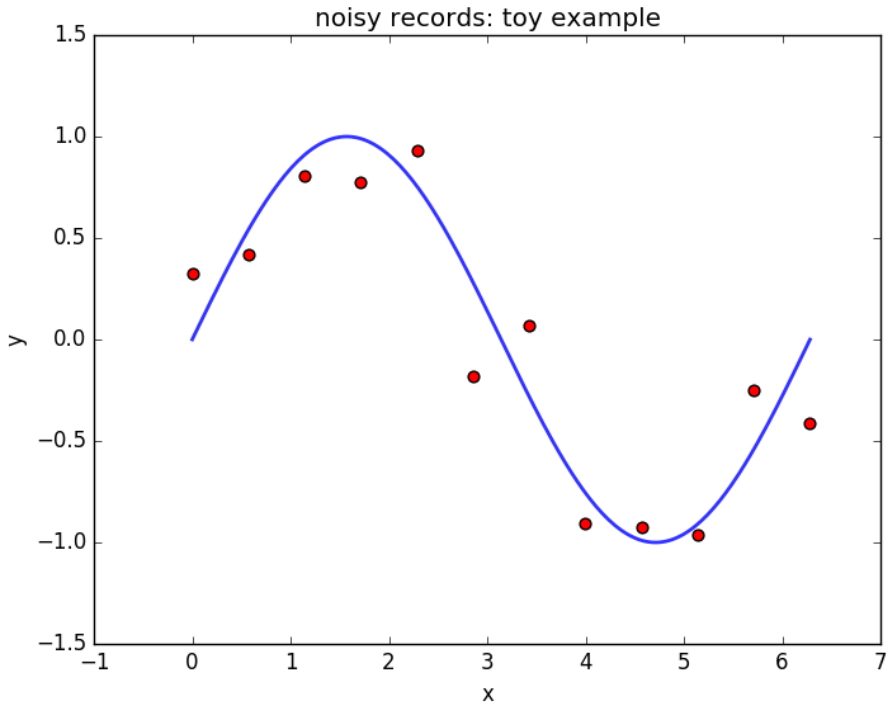
Example: classification of drugs



- Here, we want to classify molecules with respect to their efficiency against a disease (binary classification: a drug is efficient or not).
- An important question here is how to encode a molecule. One option is to define chemoinformatic features and obtain a vectorial representation of the molecule $\mathbf{x} \in \mathbb{R}^P$.

Overview

A simple example: polynomial curve fitting[1]



- From a set of measured points (x_i, y_i) (red), we would like to build a model to predict the value y for any given x.
- The true function is $g(x) = \sin(x)$ (displayed in blue).
- The measurements y_i are noisy outputs of that function, i.e.

$$y_i = \sin(x_i) + \epsilon \ , \qquad \epsilon \sim \mathcal{N}(0, 0.2) \tag{5}$$

[1] Example adapted from [Bishop, 2006]

A simple example: polynomial curve fitting

- We use the following polynomial model:

$$\begin{aligned} f(x) &= a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m \\ &= \boldsymbol{\theta}^T \boldsymbol{\phi}(x) \end{aligned} \tag{6}$$

- Parameter vector: $\boldsymbol{\theta} = (a_0, a_1, \ldots, a_m)^T$
- Here, the initial measurement x is a scalar. In our model, we map x to a higher dimensional space:

$$\begin{aligned} \boldsymbol{\phi} : \mathbb{R}^P &\rightarrow \mathbb{R}^Q \\ x &\rightarrow \boldsymbol{\phi}(x) = (1, x, x^2, \ldots, x^m)^T \end{aligned} \tag{7}$$

- The model is linear in the parameters θ and linear in $\boldsymbol{\phi}$, but for $m > 1$, the model is not linear in x.

A simple example: polynomial curve fitting

- One classical approach is to minimize the least squared error between measured and predicted values:

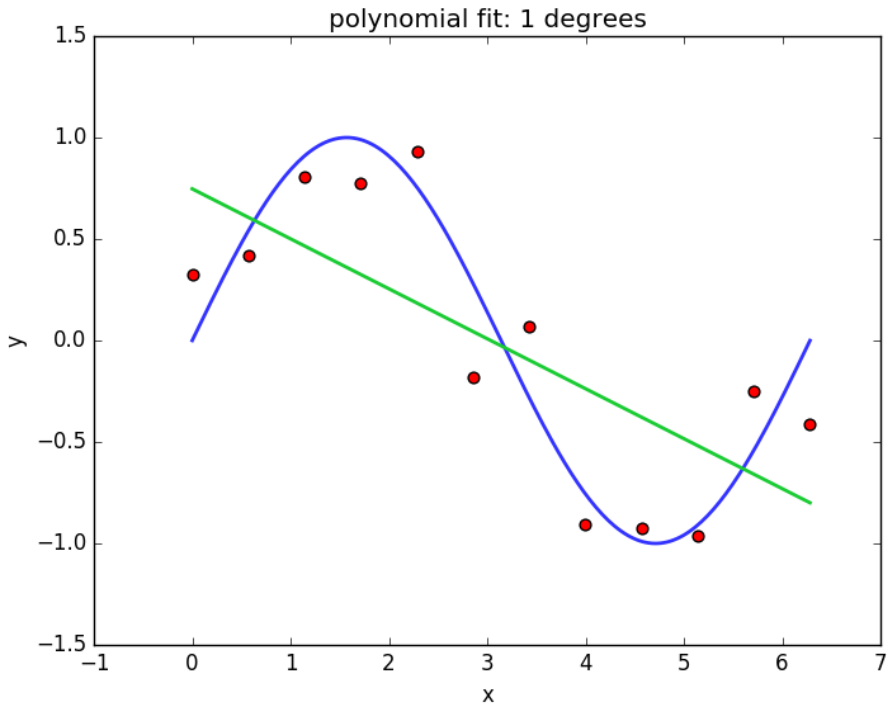
$$\begin{aligned} \min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) &= \min_{\boldsymbol{\theta}} \sum_{i=1}^{N} (y_i - f(x_i))^2 \\ &= \min_{\boldsymbol{\theta}} \sum_{i=1}^{N} (y_i - \boldsymbol{\theta}^T \boldsymbol{\phi}(x_i))^2 \end{aligned} \tag{8}$$

- This can be achieved by setting the gradient with respect to $\boldsymbol{\theta}$ to zero:

$$\nabla_{\boldsymbol{\theta}} L = (\frac{\partial L}{\partial a_0}, \frac{\partial L}{\partial a_1}, \ldots, \frac{\partial L}{\partial a_m})^T = 0 \tag{9}$$

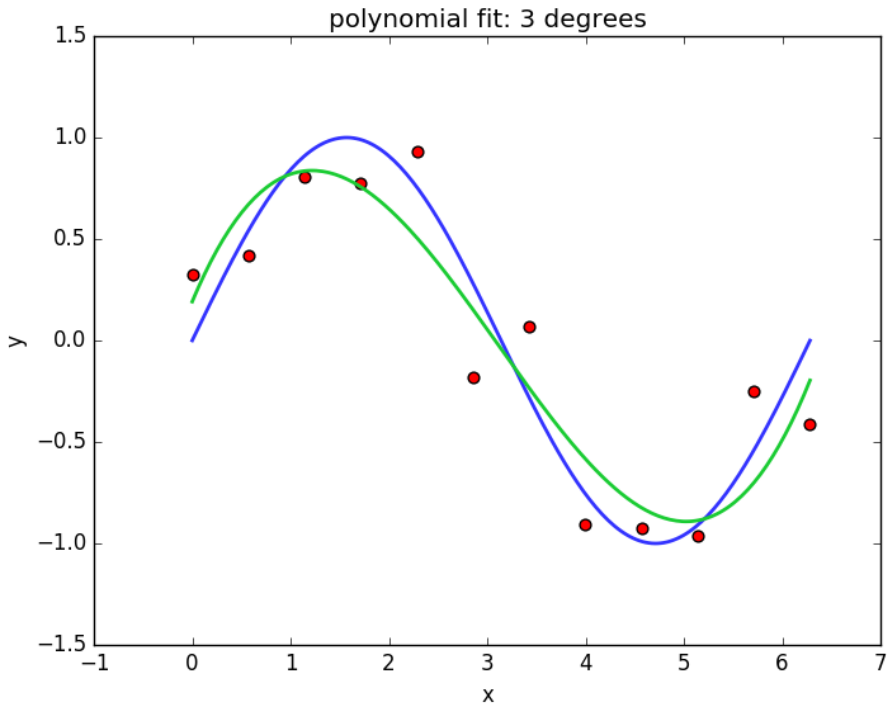
- Unlike for most optimization problems in this course, this leads to an analytical solution for $\boldsymbol{\theta}$. This is known as **linear regression**. For more details, we refer to [Hastie et al., 2009].

Overfitting and underfitting



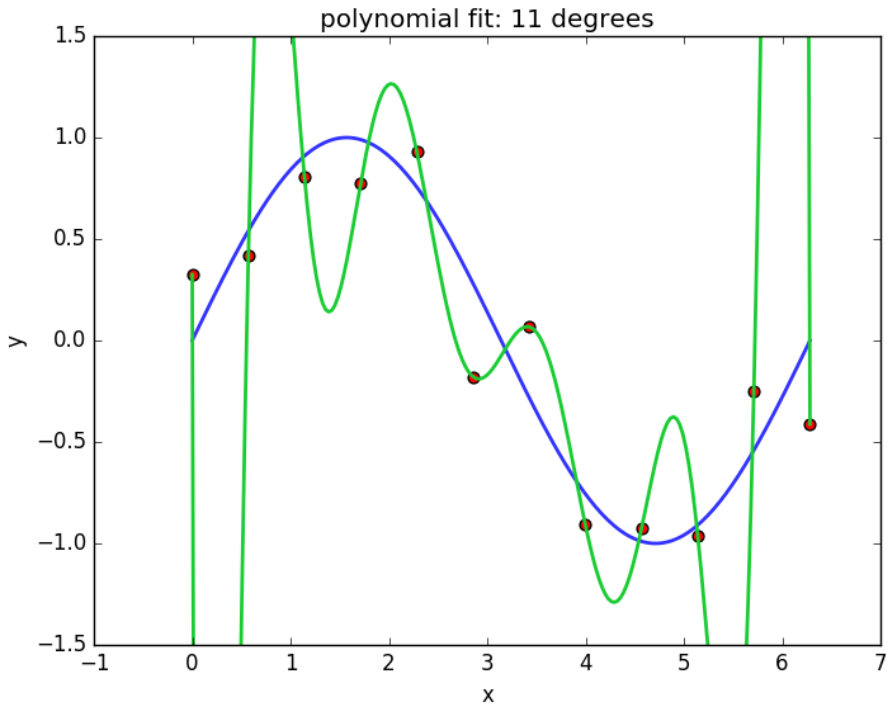
For $m = 1$, the model is linear in its inputs. The solution is not capable of modeling the measured data points; we get a poor approximation of the original function. The family of functions we have used was not complex enough to model the true data distribution. We also speak of **underfitting**.

Overfitting and underfitting



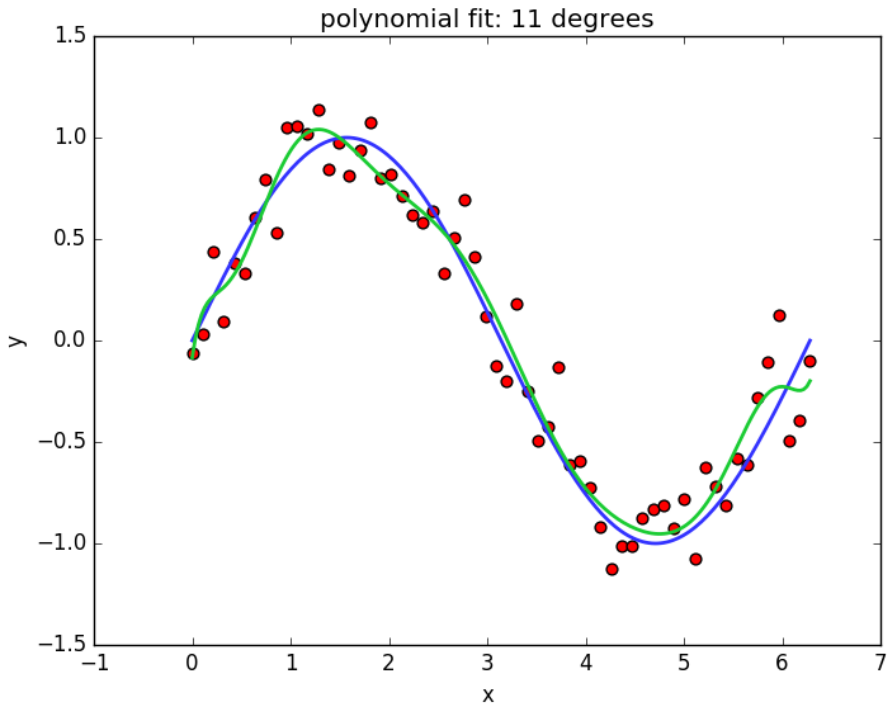
For $m = 3$, we obtain a solution that seems to be quite right: it is sufficiently complex to model the true data distribution, but not too complex to model the small variations which are due to noise.

Overfitting and underfitting



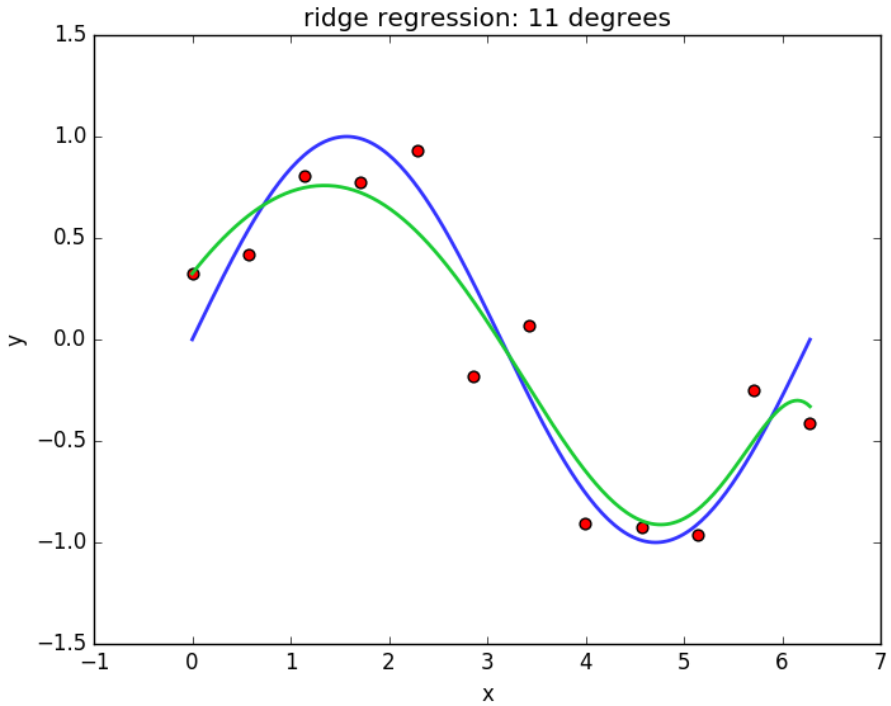
For $m = 11$, we obtain a solution that has zero error (the function passes through every point of the training set). But the coefficients with large absolute values that cancel each other precisely on the training points lead to a highly unstable function. We speak of **overfitting** and **poor generalization**.

Overfitting and underfitting



One way of reducing overfitting is to increase the number of samples. Even if the function is complex, it cannot be "too wild", as it has to find a compromise between many training samples. This however implies the annotation (or measurement) of more samples.

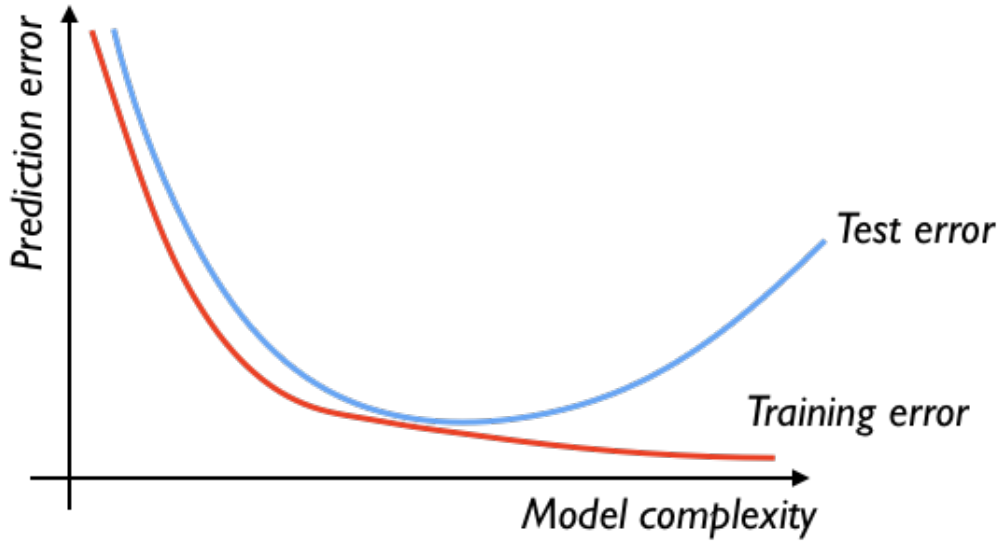
Overfitting and underfitting



Another way of preventing overfitting without increasing the number of samples, is to add a penalization term in the optimization procedure. This is also known as **regularization**:

$$L = \sum_{i=1}^{N} (y_i - \boldsymbol{\theta}^T \boldsymbol{\phi}(x_i))^2 + \lambda \|\boldsymbol{\theta}\|^2 \tag{10}$$

Generalization: training and test error



- Supervised Learning aims at finding a function f that predicts an output value y from a measurement x for unseen data, i.e. for data that has not been used to find f.
- Machine Learning is much concerned with avoiding f to **memorize** the training set, i.e. to perform well on a training set but poorly a test set.
- An important paradigm is that we must never evaluate the performance of our machine learning method on the data that has been used to train it.

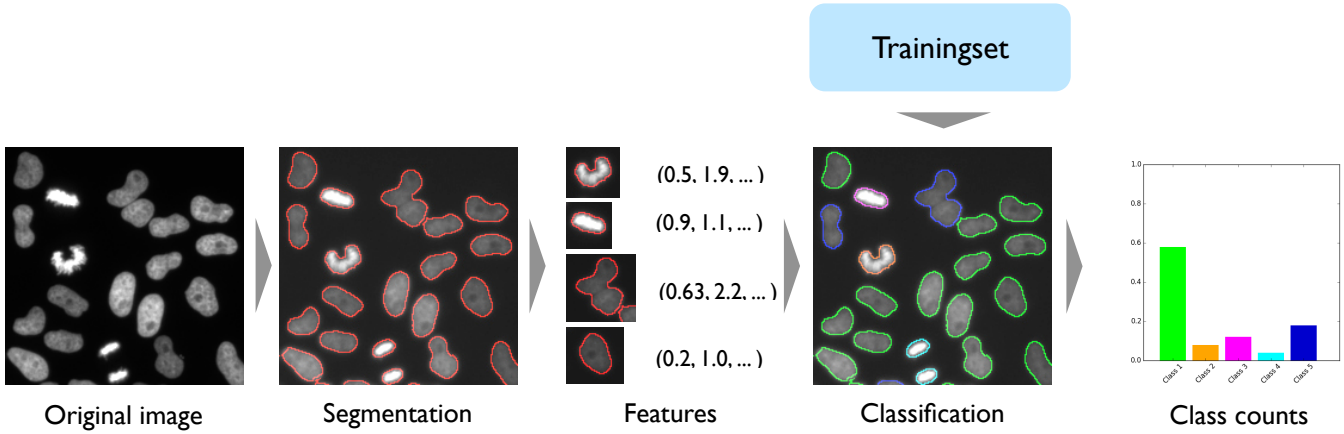
Generalization: strategies

- Many ML algorithms can be written as an optimization problem:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) + \mathcal{R}(\boldsymbol{\theta})$$

  Minimizing the loss $L(\boldsymbol{\theta})$ aims at finding the rule to reproduce the annotations in the training set, minimizing the regularization term $\mathcal{R}(\boldsymbol{\theta})$ aims at avoiding the model to adapt too much to the training data, leading to simpler models. We have seen the L_2 norm, but there are many other options for $\mathcal{R}$.
- Other regularization strategies include:
  - Model averaging (ensemble methods)
  - Artificial or actual increase of training data
  - Adversarial training

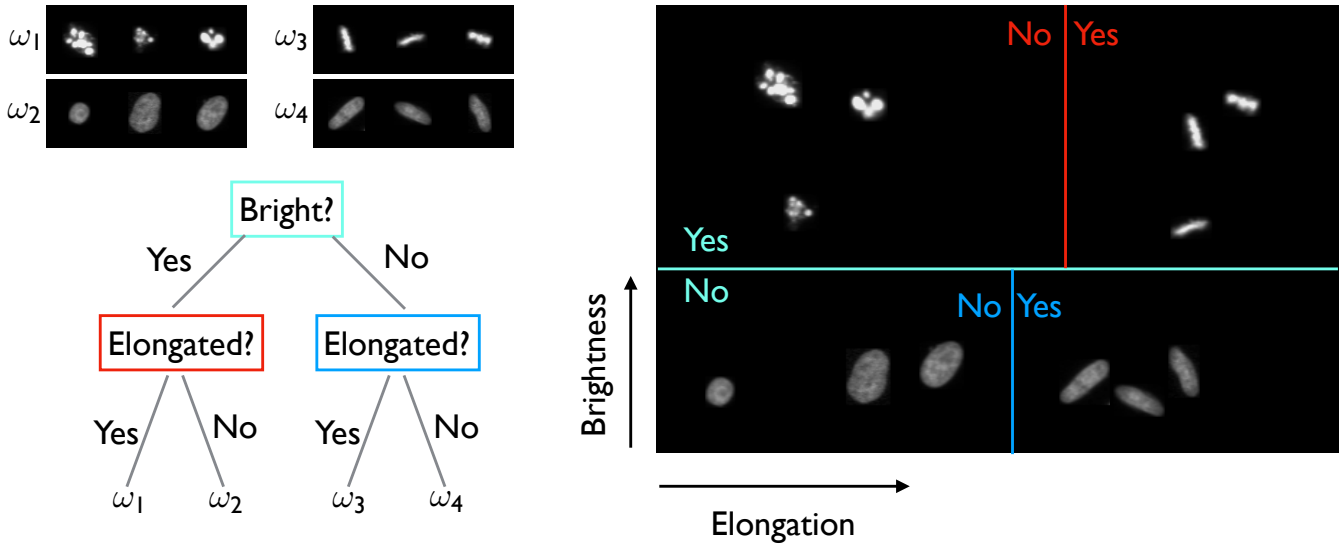
Example: classification of cells



- In this example, we wish to classify cell images.
- For this, cells are first segmented and then represented by a vector $x \in \mathbb{R}^P$ of features describing shape and texture for each segmented object.
- We are given a training set $T = \{(x_i, y_i)\}_{i=1,\ldots,N}$, i.e. for each cell in the training set $x_i \in \mathbb{R}^P$, we know its morphological class y_i.
- The task is to infer a function f from the training set that allows to assign one of these classes to new unseen objects.

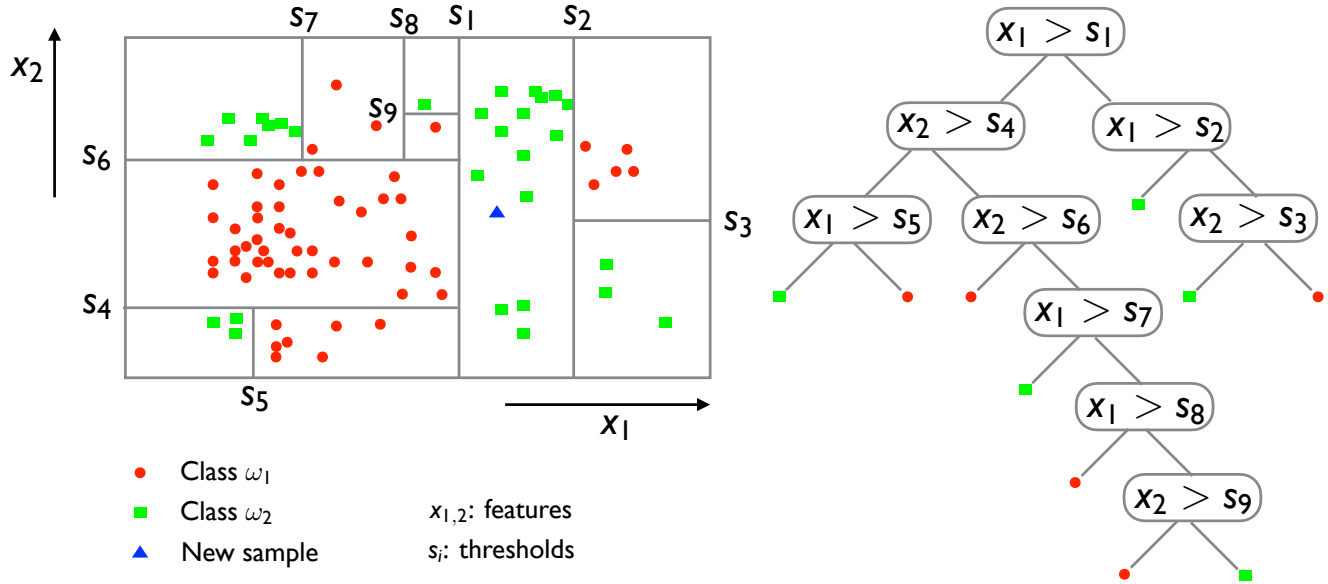
Random Forests

Random Forest: Decision trees



- Intuitive approach: applying a series of "rules" to the training data to recover the classes ω_i (e.g. "is the cell bright?", "is the cell elongated?", ...)
- Each rule (or decision) divides the set of objects into two subsets.
- The whole classifier is thus represented by a decision tree, i.e. a hierarchically organized set of binary decisions.

Random Forest: Decision trees



- The decision tree corresponds to a partition of the feature space.
- The corresponding decision boundaries can be very complex and adapt to the training set.
- Classification of a new object: application of the binary decision rules and assignment of the leaf label.

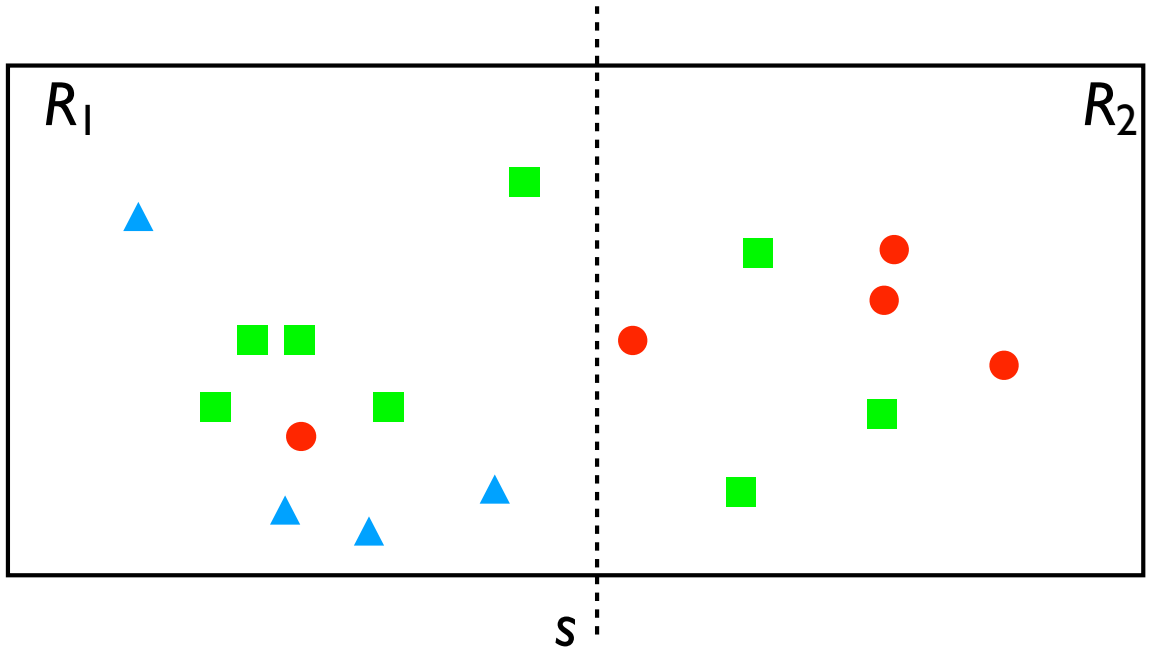
Random Forest: Decision trees

- Such decision trees can be built from data in an optimal way, by repeated division of the feature space.
- At every step, we split the data set into two, according to one feature.
- The feature and the threshold are chosen automatically in such a way that the resulting groups have best "purity".
- This can be achieved by minimizing the GINI impurity. For K classes, GINI impurity is defined as:

$$\begin{aligned} GI(R_m) &= \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \\ GI(s) &= \sum_{R_m} = \frac{|R_m|}{N} GI(R_m) \end{aligned}$$

where R_m are the sets resulting from the split s and $\hat{p}_{mk}$ is the probability of a sample in R_m to belong to class k.

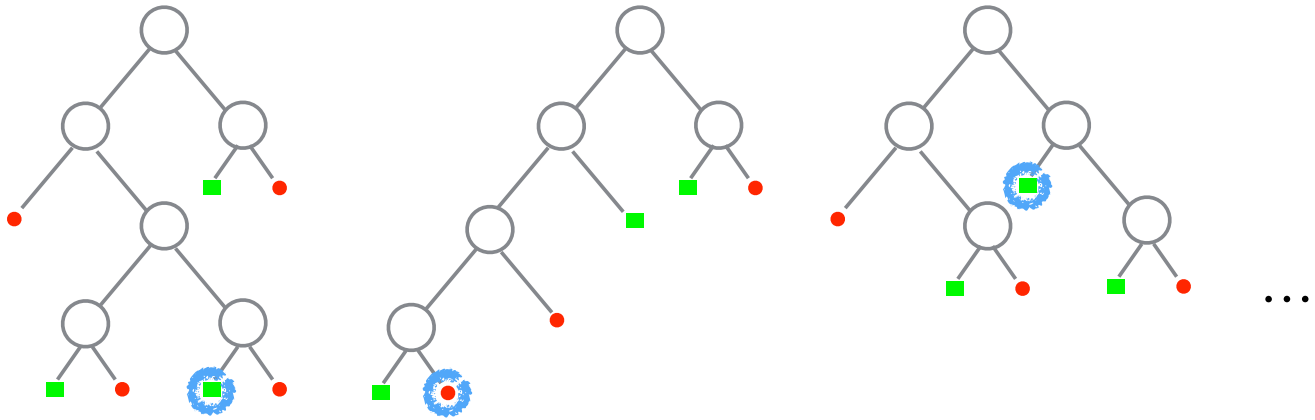
Random Forest: GINI impurity



Here, we obtain for the GINI impurity:

$$
\begin{aligned}
GI(R_1) &= \frac{1}{10} \cdot \frac{9}{10} + \frac{5}{10} \cdot \frac{5}{10} + \frac{4}{10} \cdot \frac{6}{10} = 0.58 \\
GI(R_2) &= \frac{0}{7} \cdot \frac{7}{7} + \frac{4}{7} \cdot \frac{3}{7} + \frac{3}{7} \cdot \frac{4}{7} = 0.49 \\
GI(s) &= \frac{10}{17} \cdot R_1 + \frac{7}{17} \cdot R_2 = 0.54
\end{aligned}
$$

Random Forests



- Decision trees can approximate very complicated decision boundaries, but they tend to fit to much to the data (overfitting).
- Random forests: set of decision trees, each learned on a different (randomly drawn) portion of the data and with different (randomly selected) features.
- Each tree gives a classification result.
- The final result is obtained by a majority vote.

Random Forests

- Because each tree is slightly different from the others, each tree learns a slightly different aspect of the training data.
- A classification method that exists in averaging the results of several classifiers (often weak classifiers), is called **ensemble method**.
- The strategy of averaging is a form of regularization.
- In practice, model averaging is very popular in the deep learning field. Often, one averages simply the output of several networks to obtain a more robust classifier.

Linear Discriminant Analysis

The Bayes rule of classification 1/3

Product and sum rules of probabilities

Let X and Y be random variables, and let $P(X)$ and $P(Y)$ be their marginal probabilities, $P(X, Y)$ their joint probability and $P(X \mid Y)$ the conditional probability of X given Y.

- Product rule:

$$P(X, Y) = P(X \mid Y)P(Y) \tag{11}$$

- Sum rule:

$$P(X) = \sum_{Y} P(X, Y)P(Y) \tag{12}$$

From the product rule and $P(X, Y) = P(Y, X)$ we get immediately the Bayes theorem:

$$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)} \tag{13}$$

The Bayes rule of classification 2/3

We assume that the measured features are continuous variables. Almost identical rules hold in this case, but we need to replace probabilities by probability densities.

Bayes rule of classification

Let $x \in \mathbb{R}^P$ with a probability density $p(x)$ and $Y \in \{\omega_1, \ldots, \omega_K\}$ a discrete set of class labels. The posterior probability $P(Y = \omega_k \,|\, x)$ can be written as:

$$P(Y = \omega_k \,|\, x) = \frac{p(x \,|\, Y = \omega_k) P(Y = \omega_k)}{p(x)} \tag{14}$$

The Bayes rule of classification: *choose the class that maximizes the posterior probability* $P(Y = \omega_k | x)$:

$$\omega^*(x) = \arg\max_k P(Y = \omega_k \,|\, x) = \arg\max_k p(x \,|\, Y = \omega_k) P(Y = \omega_k) \tag{15}$$

The Bayes rule of classification 3/3

- The Bayes rule of classification simply states that the best class to choose is the one with highest probability given the observation.
- Importantly, we can calculate this posterior probability from the class dependent feature distributions and the class probabilities.
- This rule is important because it gives the best possible classifier, given that the feature distributions $p(x \mid Y = \omega_k)$ and the class probabilities $P(Y = \omega_k)$ are known.
- However, this is normally not the case: these distributions need to be estimated from the training data.

The Bayes rule for classification in the binary case

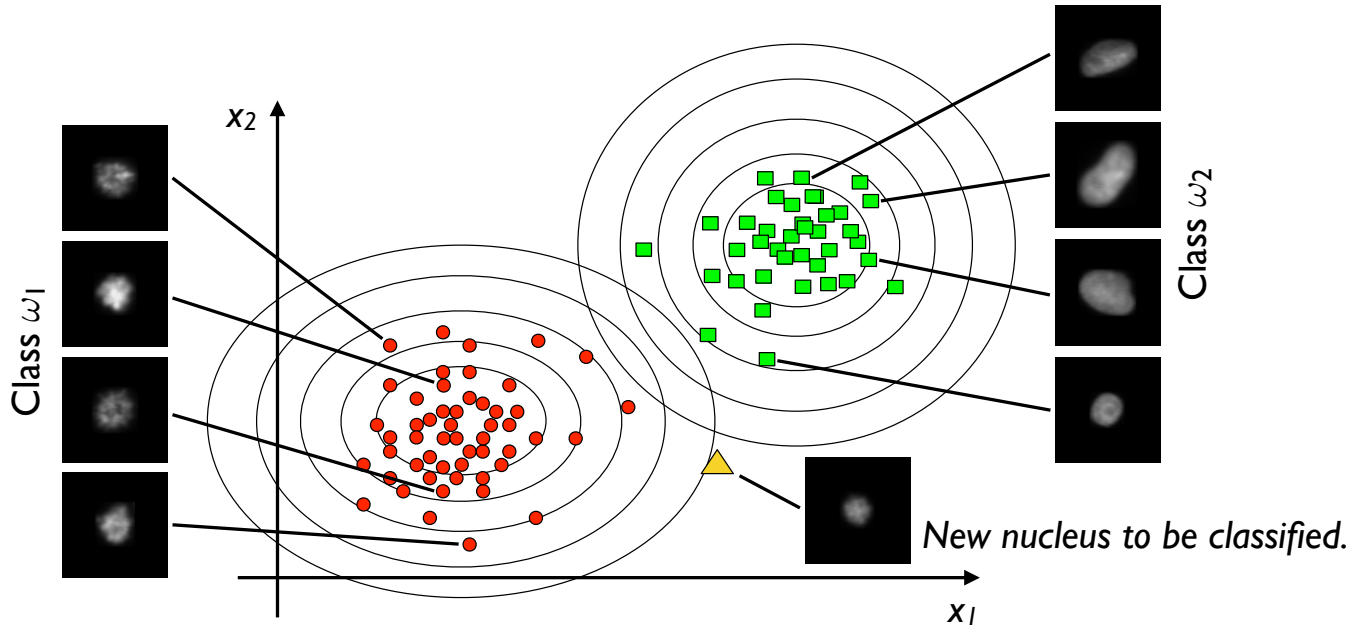
For simplicity, we assume the case of binary classification, i.e. $\Omega = \{\omega_1, \omega_2\}$. The Bayes rule for classification states that we should choose ω_1 if:

$$\frac{p(x|\omega_1)P(\omega_1)}{p(x|\omega_2)P(\omega_2)} > 1$$

We can apply the log to both sides of the equation. We obtain the following expression for the Bayes rule of classification:

$$\log \frac{p(x|\omega_1)P(\omega_1)}{p(x|\omega_2)P(\omega_2)} > 0 \tag{16}$$

Normality assumption



We assume that features are normally distributed for each of the classes:

$$p(x|\omega_k) = \frac{1}{(2\pi)^{\frac{p}{2}}|\Sigma_k|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)} \tag{17}$$

Linear Discriminant Analysis (LDA)

Plugging equation (17) into (16) leads to **Quadratic discriminant analysis (QDA)**:

$$\log \frac{P(\omega_1)}{P(\omega_2)} + \frac{1}{2} \log \frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2}(x - \mu_1)^T \Sigma_1^{-1}(x - \mu_1)$$
$$+ \quad \frac{1}{2}(x - \mu_2)^T \Sigma_2^{-1}(x - \mu_2) > 0$$

If we further assume that the covariance matrices of the classes are equal, i.e. $\Sigma_1 = \Sigma_2 = \Sigma$), we obtain **Linear Discriminant Analysis, LDA**:

$$\log \frac{P(\omega_1)}{P(\omega_2)} + x^T \Sigma^{-1}(\mu_1 - \mu_2) - \frac{1}{2}(\mu_1 - \mu_2)^T \Sigma^{-1}(\mu_1 + \mu_2) > 0$$

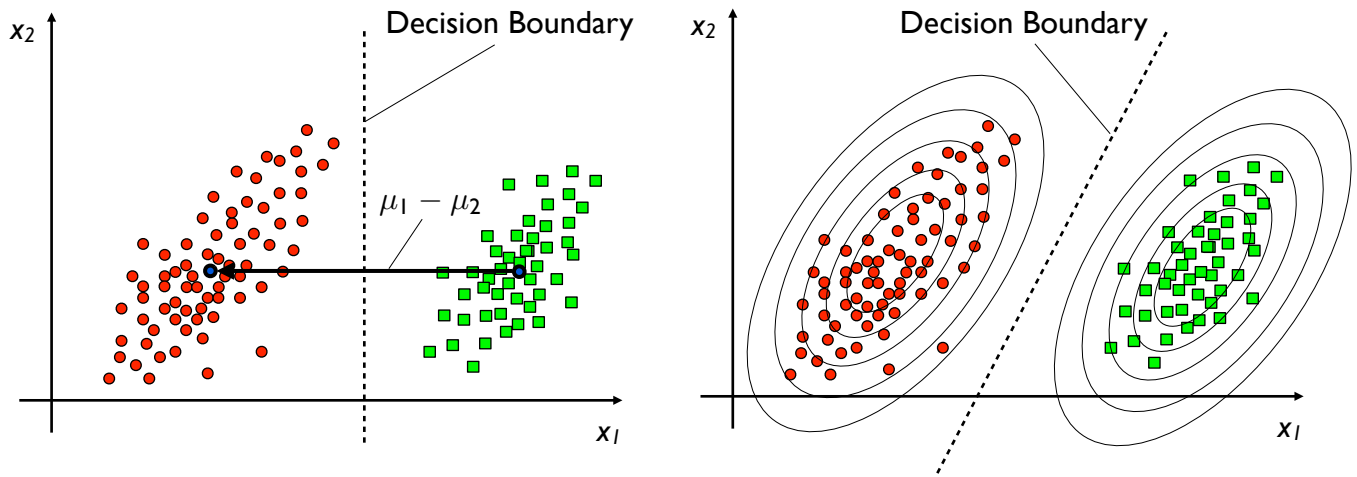
Comparison LDA vs. closest mean 1/2

- Closest Mean Classifier:
  - Calculate the mean value for each of the classes (from the training set).
  - Rule: for a new vector $x \in \mathbb{R}^P$ assign the class with the closest mean, i.e. choose class ω_1 if :

$$\begin{aligned} \|x-\mu_1\| &< \|x-\mu_2\| \\ (x-\mu_1)^T(x-\mu_1) - (x-\mu_2)^T(x-\mu_2) &< 0 \\ x^T(\mu_2-\mu_1) + \frac{1}{2}(\|\mu_1\|^2 - \|\mu_2\|^2) &< 0 \\ x^T(\mu_1-\mu_2) + b &> 0 \end{aligned}$$

- We see that the only term that depends on x is linear and corresponds to a projection of x onto the difference vector of the two class means μ_1 and μ_2.

Comparison LDA vs. closest mean 2/2



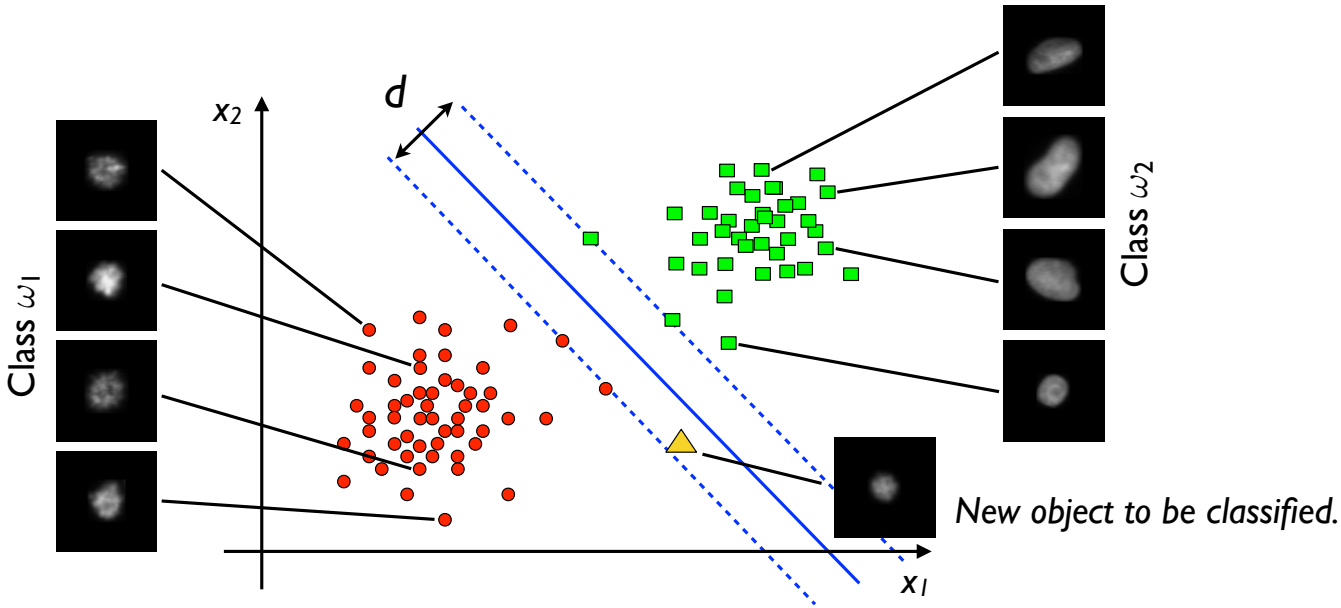
- The expression for LDA classification is:

$$x^T \Sigma^{-1}(\mu_1 - \mu_2) + b > 0$$

- We thus see that the only difference is that we take the covariance matrix Σ into account.
- The normal vector of the separating hyperplane is thus not $(\mu_1 - \mu_2)$ but $\Sigma^{-1}(\mu_1 - \mu_2)$.

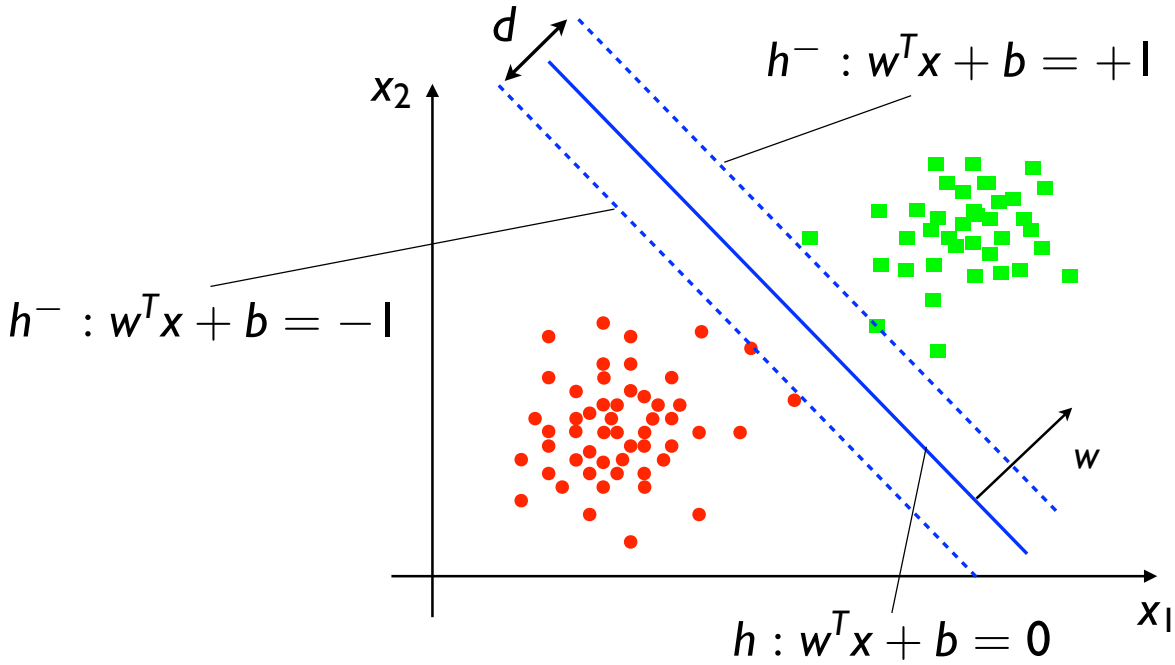
Support Vector Machines

Support Vector Machines: principle



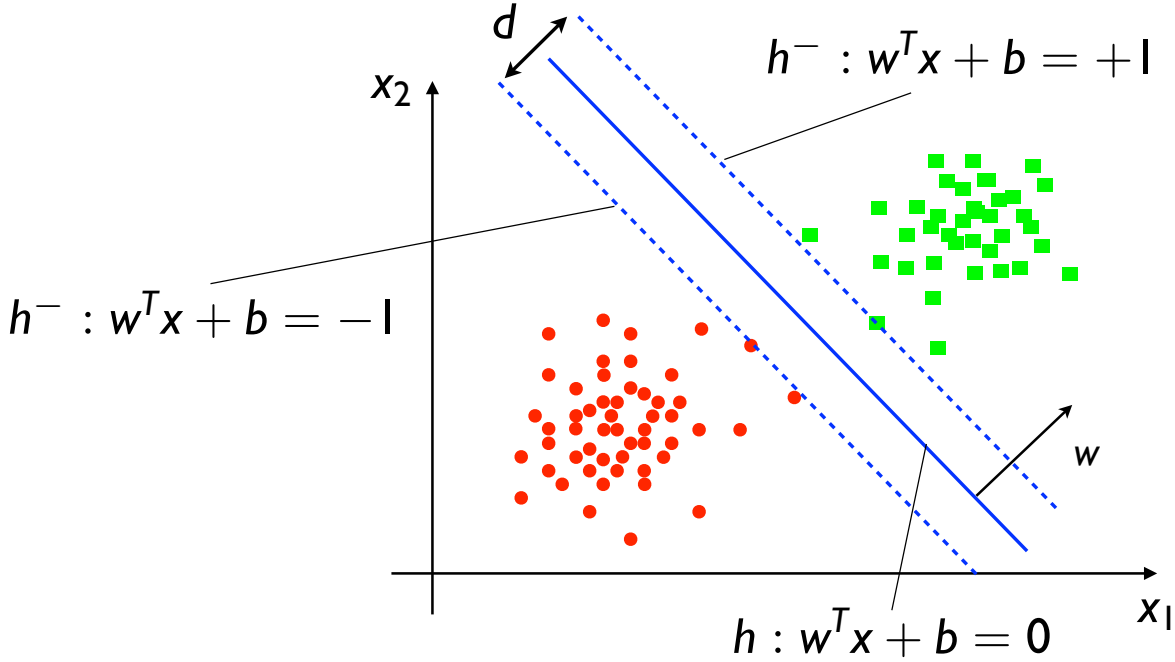
- Optimal placement of a linear decision boundary.
- Intuitive approach: place a "ribbon" instead of a single line, i.e. two parallel lines separated by a distance d.
- Maximization of the width d in order to push the separating hyperplane away from the two classes.

Support Vector Machines: linearly separable training set



- Equation of a linear decision boundary $h : w^T x + b = 0$.
- w: normal vector of the separating hyperplane.
- $w^T x + b > 0$: x is on the same side as the normal vector w points to.
- $w^T x + b < 0$: x is on the opposite side.

Support Vector Machines: linearly separable training set

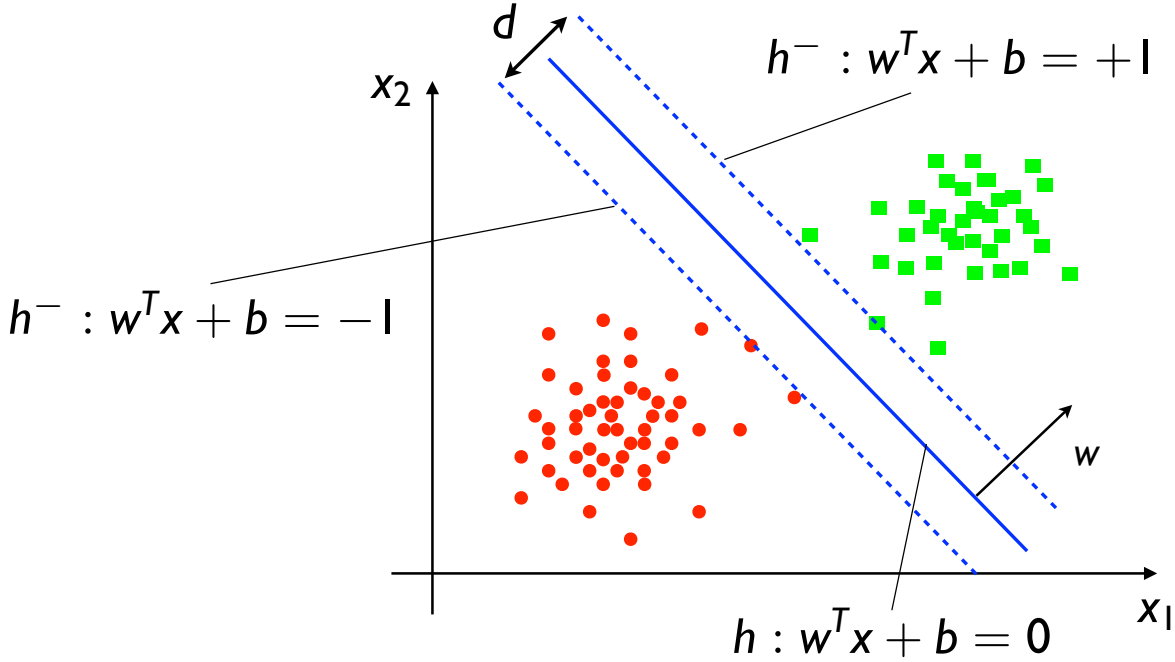


- With the encoding $y \in \{-1, +1\}$, a sample is correctly classified if $y_i(w^T x_i + b) > 0$.
- The distance between the two parallel lines $w^T x + b = \pm 1$ is given by

$$d = \frac{2}{\|w\|} \tag{18}$$

- Maximizing of d is equivalent to minimizing $\|w\|^2$.

Support Vector Machines: linearly separable training set

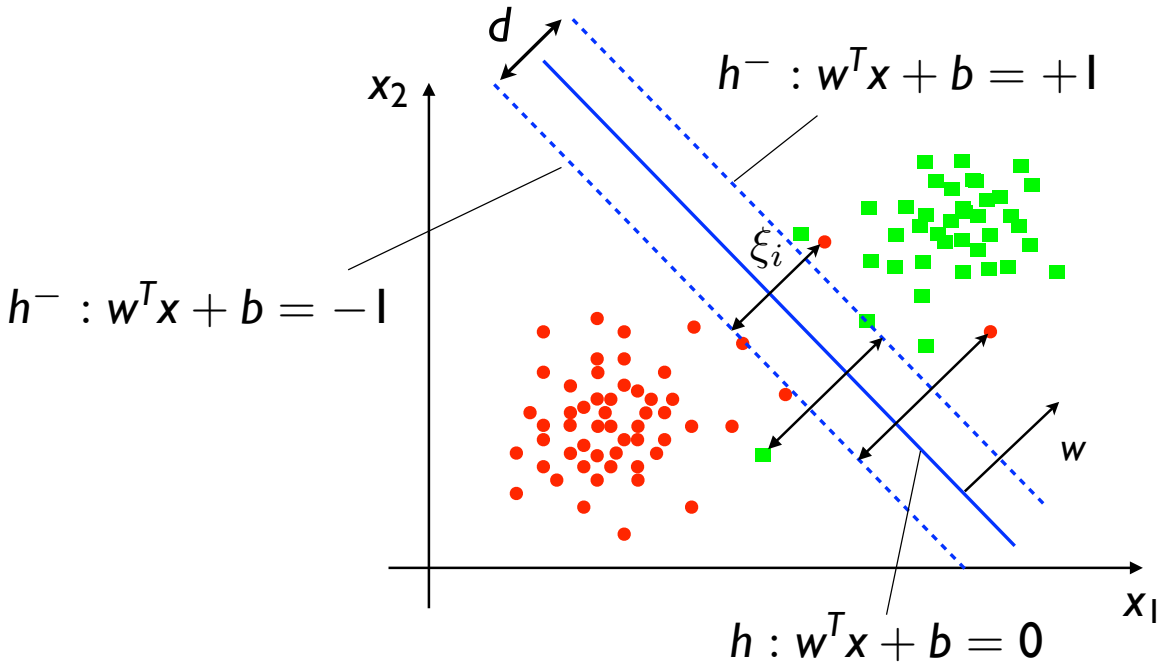


- With this, the problem can be written as a **convex optimization problem under constraints**:

$$\begin{aligned} \text{minimize} \quad & \|w\|^2 \\ \text{subject to} \quad & y_i(w^T x_i + b) \geq 1 \quad i = 1, \ldots, N \end{aligned}$$

- Convexity implies that there is no local minimum besides the global minimum.

SVM for not linearly separable data



- In reality, complex data is rarely linearly separable.
- In this case, we need to find a compromise between error term $\sum_{i=1}^{N} \xi_i$ and regularization $\|w\|^2$.

SVM for not linearly separable data

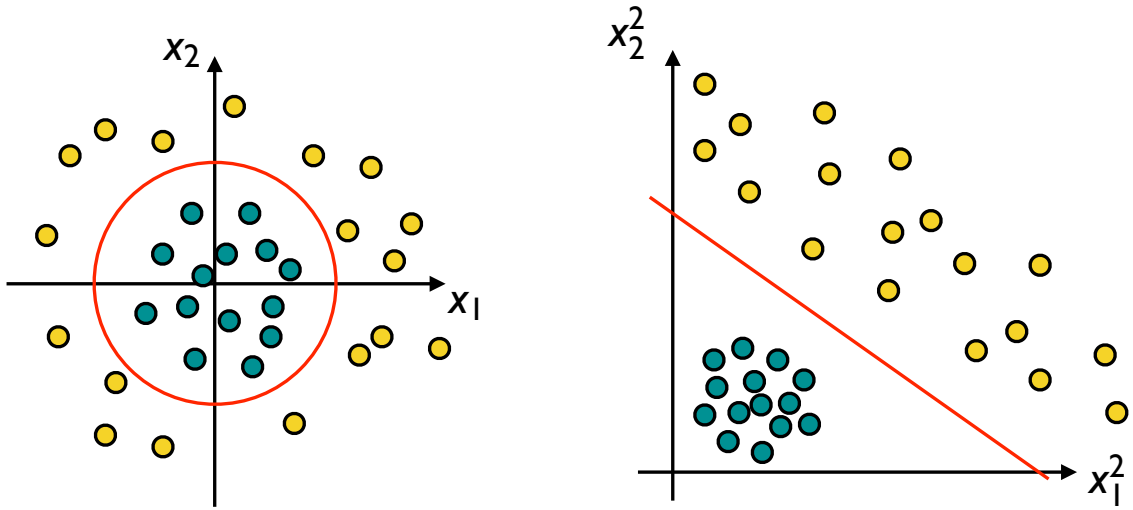
- SVM correspond to the following constrained optimization problem:

$$\begin{aligned} \min_{w,\xi} \quad & \|w\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & y_i(w^T x_i + b) \geq 1 - \xi_i \quad i = 1, \ldots, N \\ & \xi_i \geq 0 \quad i = 1, \ldots, N \end{aligned}$$

- This problem can be efficiently solved with the Lagrange formalism (not shown).
- We note that this corresponds to the formulation we have seen earlier:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) + \mathcal{R}(\boldsymbol{\theta})$$

Support Vector Machines: Kernel trick



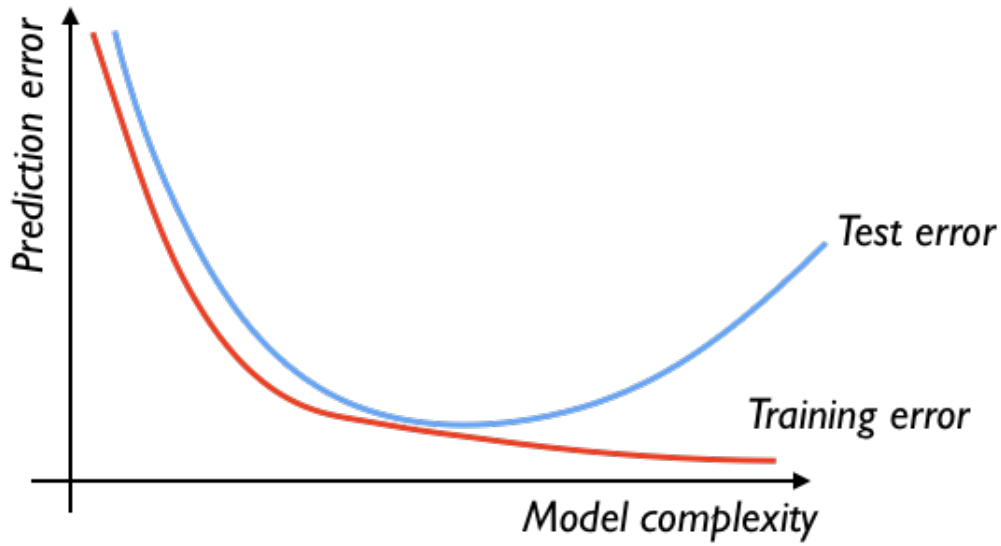
- Limitation: linear decision boundaries
- Solution: transform the features in a higher dimensional space.
- SVM: in-built mechanism to do such transformations efficiently.

How to set hyperparameters

- Training a classifier: finding automatically a large number of feature weights (SVM, LDA) or other parameters (RF) from annotated data.
- Hyperparameters: a small number of parameters that are set to control the training process, such as the number of trees, or the regularization parameter λ (C in SVM).
- The question is: how can we find good hyperparameters?
- Strategy: grid search. We choose the set of hyperparameters that perform best (i.e. that lead to the classifier with the best performance).
- Problem: how can we measure the performance?

Assessment of performance

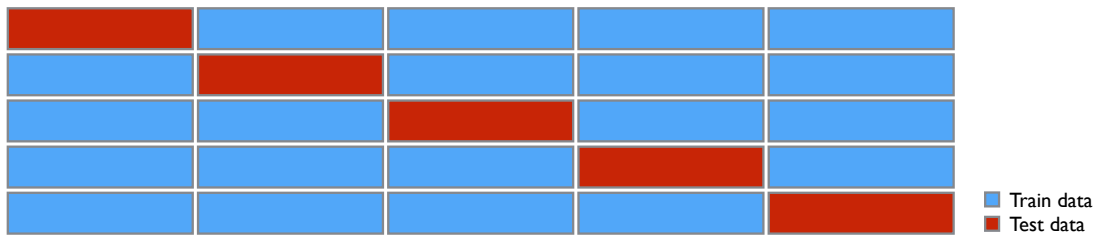
- First idea: we train a classifier on the training set T and calculate the accuracy.
- Problem: Training error is a poor approximation of the test error.



- We need to estimate the test error (error on unseen samples)!

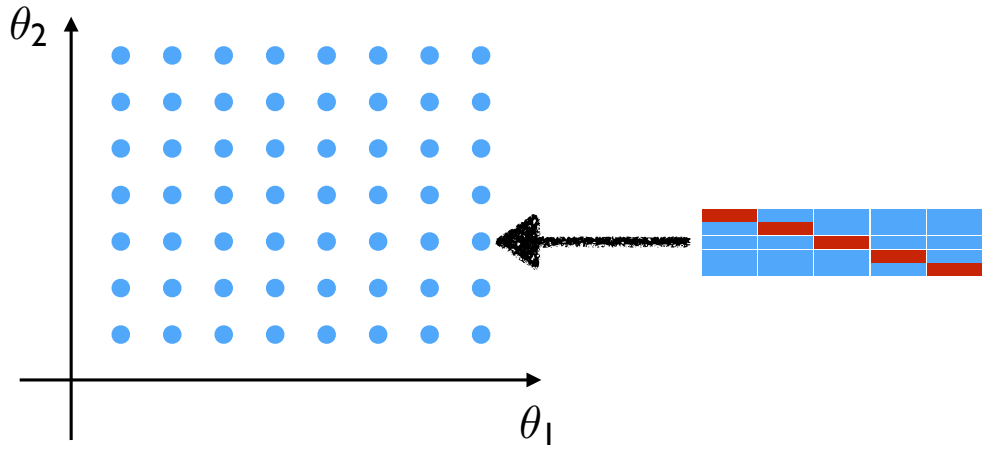
How to set hyperparameters: estimation of the test error

- Second idea: to split the training set in two subsets T_1, T_2. We train on T_1, we test on T_2.
- Problem: annotated data is often expensive, and we would like to have the largest possible sets for training and testing.
- Third idea: Cross Validation. We split the data into K folds. We train on $K-1$ folds, and we test on the remaining fold. We iterate until each fold has been used once for testing.



- This provides us with an estimation of the accuracy for unseen data (test error).
- Hyperparameter selection: we can now choose the hyperparameter with the best accuracy.

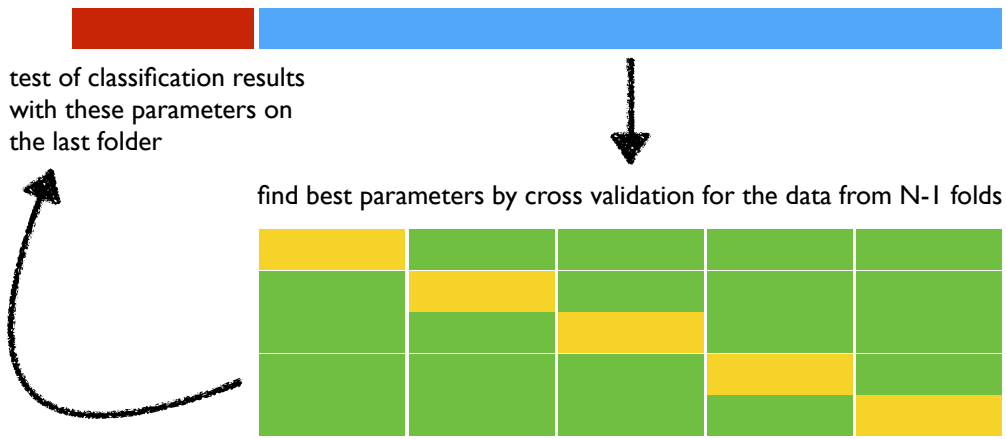
How to set hyperparameters: grid search



- If there are several parameters, we can perform this for every combination of values (grid search).
- Alternatively, we can use random hyperparameter values.
- Problem: time-consuming.
- Ad hoc strategies to reduce the computation time: coarse-to-fine strategy.

Performance evaluation with optimized hyperparameters

- If we optimize the hyperparameters in this way, the final performance might be over-optimistic (we have chosen the hyperparameters that give best performance for the test set, i.e. we have used the test set to set the parameters).
- Solution: nested cross validation.



- Nested cross validation is extremely time consuming.

Performance evaluation in deep learning

- In the context of deep learning with long training times, we often do not use cross validation.
- Split the training set into 3:
  - **Training set**: used to obtain the classifier for a given set of hyperparameters.
  - **Validation set**: used to find good hyperparameters.
  - **Test set**: used to evaluate performance.

Conclusion

- Supervised Machine Learning is concerned with inferring a function f from a set of annotated data, allowing to predict an output variable y from an input variable x.
- Different views:
  - Probabilistic view: we maximize the posterior probability.
  - Discriminant view: we optimize the separation of classes.
- Ultimately, we with to minimize the error, but we need to regularize the predicting function.
- Forms of regularization we have seen so far:
  - Regularization by model averaging (e.g. RF)
  - Minimizing a global loss function containing an error and a regularization term:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) + \mathcal{R}(\boldsymbol{\theta})$$

- All the methods, we have seen so far work on a fixed representation of the objects (often a vector $x \in \mathbb{R}^P$).

References I

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

[Fisher, 1936] Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188.

[Hastie et al., 2009] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition.