Machine Learning for researchers (300001030)
Unit 1. Supervised Methods

Javier Vales Alonso

Doctorate transversal activity

2020

Technical University of Cartagena

Table of contents

- Introduction
- Linear regression
  - Curve fitting approach
  - Probabilistic approach
  - Bayesian approach
  - The bias/variance trade-off
- Linear classification models
  - Logistic regression
  - Multi-class logistic regression

Table of contents (II)

Performance measurements

- Confusion matrix
- Precision and recall

K-Nearest neighbors (K-NN)

Decision trees

- Information gain
- Tree overfitting
- Implementation notes

Table of contents (III)

Random forest

- Ensembles
- Random forests

Support vector machines (SVM)

- Kernel methods
- Maximum margin classifiers
- Soft margin classifiers

Further reading

How to study this unit?

1. Do a first reading of the unit's slides. Try to focus on the general ideas and do the first review over the maths involved.
2. Next, read and run the notebooks section by section. Leave the questions indicated there for later.
3. Do a more in-depth review of the maths with the slides and the notebook side by side. Try to understand all the developments involved. In case of doubts, read the suggested references (see Syllabus) or contact the teacher.
4. Finally, answer the exercises in the notebook and submit them back through AV.

Introduction

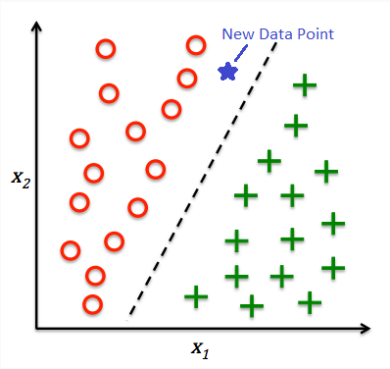
Introduction

Supervised learning problems are characterized by the availability of a *labeled data set* $\{(\boldsymbol{x}_1, t_1), (\boldsymbol{x}_2, t_2), \ldots, (\boldsymbol{x}_N, t_N)\}$, where the observations $\{\boldsymbol{x}_n\}$, for $n{=}1, \ldots, N$, are D-dimensional *input* variables for which the corresponding *target* values $\{t_n\}$ are known.

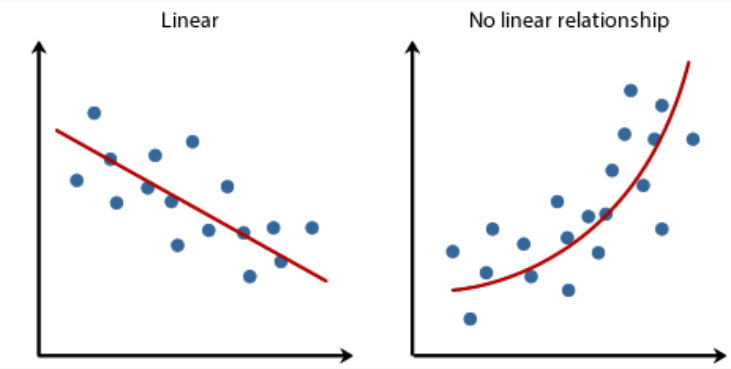
The goal of supervised learning methods is to provide suitable predictions of t for new input instances of $\boldsymbol{x}$.

If the target t is one from a finite set of values (e.g., spam/not spam, joy/sorrow/surprise, etc.) the learning problem is called *classification*, which can be binary or multi-class. When t takes values on a continuous domain the problem is called *regression*.

Introduction (II)



Example of a binary classification.
Predicted output is a class (either a red circle or a green cross).



Example of linear regression.
Predicted output is a real number (the red line or the red curve).

Introduction (III)

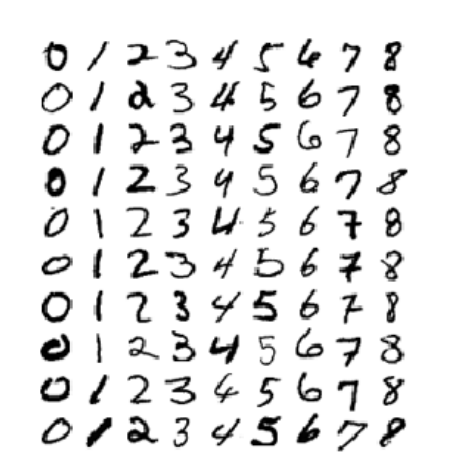
Frequently, in classification problems, labeling is performed manually. Thus, it can be costly in terms of time and resources. For example, a medical specialist has to examine MRI images and identify possible pathologies in them.

In other cases, the *target* is known *a posteriori* and then added to the data set. For example, in a daily market forecasting problem, the actual stock prices become known the next day.

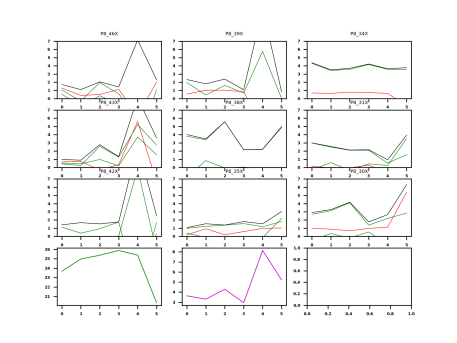
Another possibility is obtaining the *targets* using a secondary process, independent of the observed input data. For example, the stellar parallax data is used to obtain distance ranges to nearby stars. Then this data can be used as targets in a learning problem trying to predict it from other observed variables.

Introduction (IV)

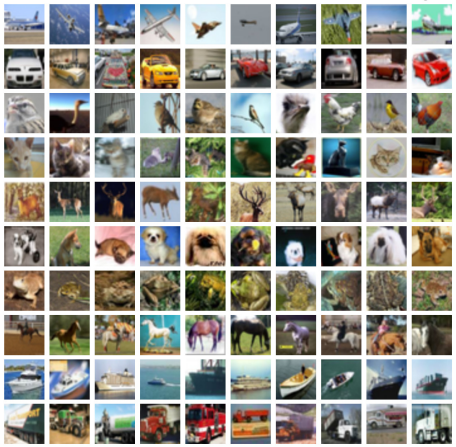
Labeling associated to some of the example data sets introduced in Unit 0:



Each instance in the MNIST database is labeled with the digit represented in the picture, from 0 to 9.



In the punching bag data set, each instance is labeled with the corresponding type of hit: Jab/Cross/Hook



In the CIFAR-10 data set each picture is labeled with the class (one of the following 10: airplane/automobile/bird/cat/deer/-dog/frog/horse/ship/truck)

Roadmap

The next sections introduce some of the principal supervised learning algorithms. The roadmap of the presentation is the following:

1. **Linear regression**. The basic treatment for parametric model-based regression problems is presented. Departing from intuitive ideas, it is shown how this model can be built on a solid statistical base. Besides, many ML concepts are introduced, such as the overfitting and the underfitting, regularization, cross-validation, or the bias/variance trade-off.
2. **Linear classification models**. The main characteristics of these models are summarized before binary, and multi-class logistic regression is discussed under a probabilistic context.

Roadmap (II)

3. **Performance measurements**. A summary of the most common performance measurements used in ML is provided.
4. K-**Nearest neighbors**. This method, which on the contrary to the ones studied previously is non-parametric and instance-based, is described in this section both for classification and regression.
5. **Decision trees and Random forests**. The next two sections discuss two related algorithms. First, the decision tree method is described. This is considered a white-box model since it is easy for a person to understand the decision-making process. Second, the random forest method is presented in connection with the concept of ensembles.

Roadmap (III)

7. **Support vector machines**. These widely used method is presented in the context of linear classification of maximum margin.
8. **Further reading**. Finally, some relevant topics left away in this course are briefly commented, together with links for interested students.

Linear regression

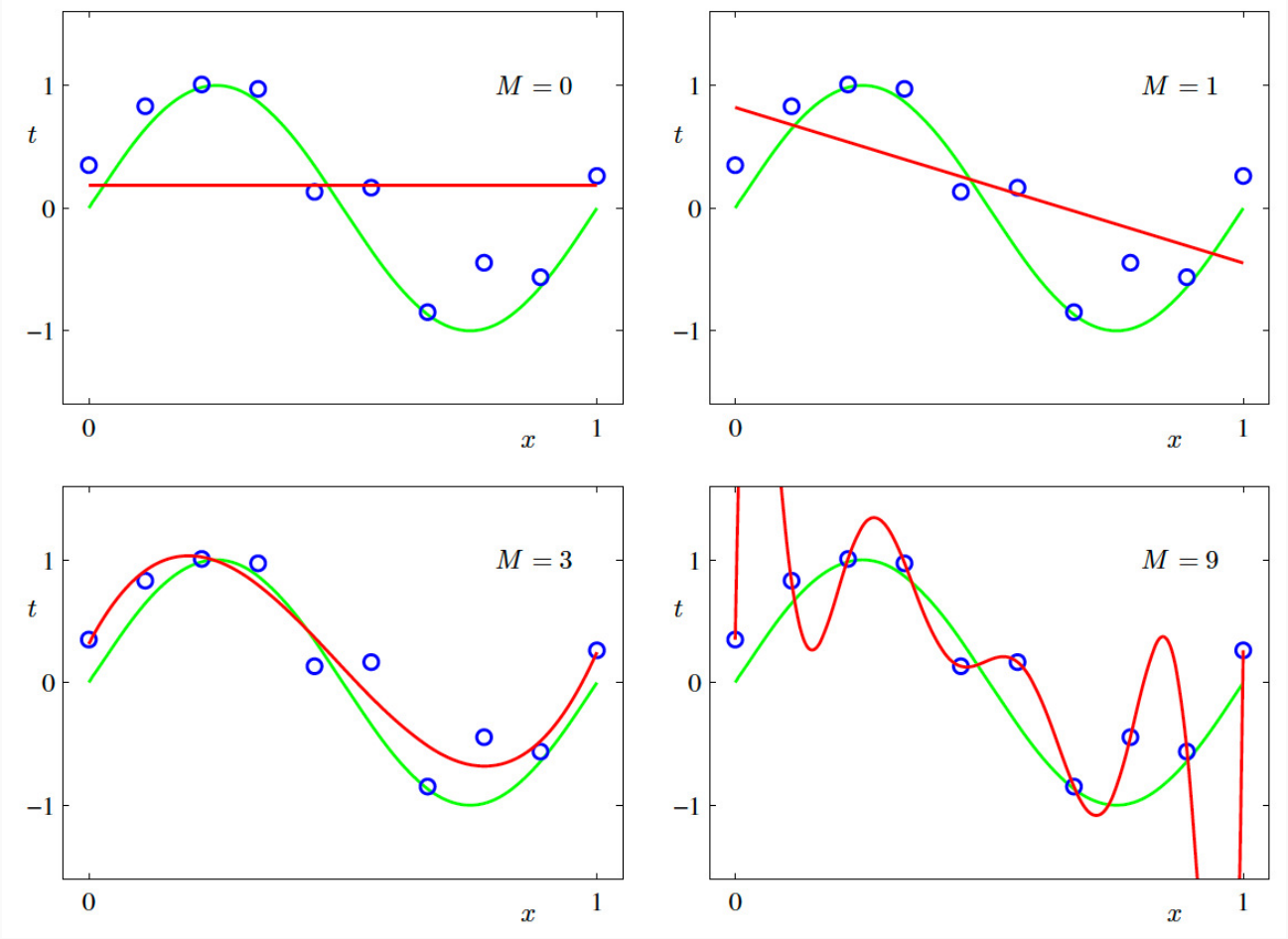
Curve fitting approach

The simplest (somewhat limited) way of looking at regression is considering it as a *curve fitting* problem. That is, selecting from a family of candidate curves the one predicting better the observed targets.

Several questions arise at this point:

- How the candidate or *hypothesis* functions can be defined?
- How to measure the error between the hypothesis and the targets?
- To which extent can we be confident about the predictions?

Curve fitting approach (II)



Curve fitting examples. The training set of N=10 points, shown as blue circles, has a single feature x and get target values $t = sin(2\pi x)$ (the green curve) plus a small Gaussian random noise. Fittings correspond to M-degree polynomials (red curves). **Take a moment to consider the previous questions in view of this figure**

Hypothesis functions

A simple way of building hypothesis functions is from a set of *basis functions* $\{\phi_j\}$, for $j=1,\ldots,M-1$, and associating to each of them a *weight* w_j. This way the hypothesis[1] is written as:

$$y(\boldsymbol{x},\boldsymbol{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\boldsymbol{x}). \tag{1}$$

Where w_0 is called the *bias* parameter, and $\boldsymbol{w}$ denotes the vector $(w_0,\ldots,w_{M-1})^T$. By assuming a dummy basis function $\phi_0(\boldsymbol{x}) = 1$, last equation can be rewritten simply as:

$$y(\boldsymbol{x},\boldsymbol{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\boldsymbol{x}) \tag{2}$$

[1] Even if the basis functions are non-linear on $\boldsymbol{x}$, our regression approach is still called *linear* since $y(\boldsymbol{x},\boldsymbol{w})$ is linear on the w coefficient since $y(\boldsymbol{x}, \lambda_1\boldsymbol{w}_1 + \lambda_2\boldsymbol{w}_2)=\lambda_1 y(\boldsymbol{x},\boldsymbol{w}_1) + \lambda_2 y(\boldsymbol{x},\boldsymbol{w}_2)$.

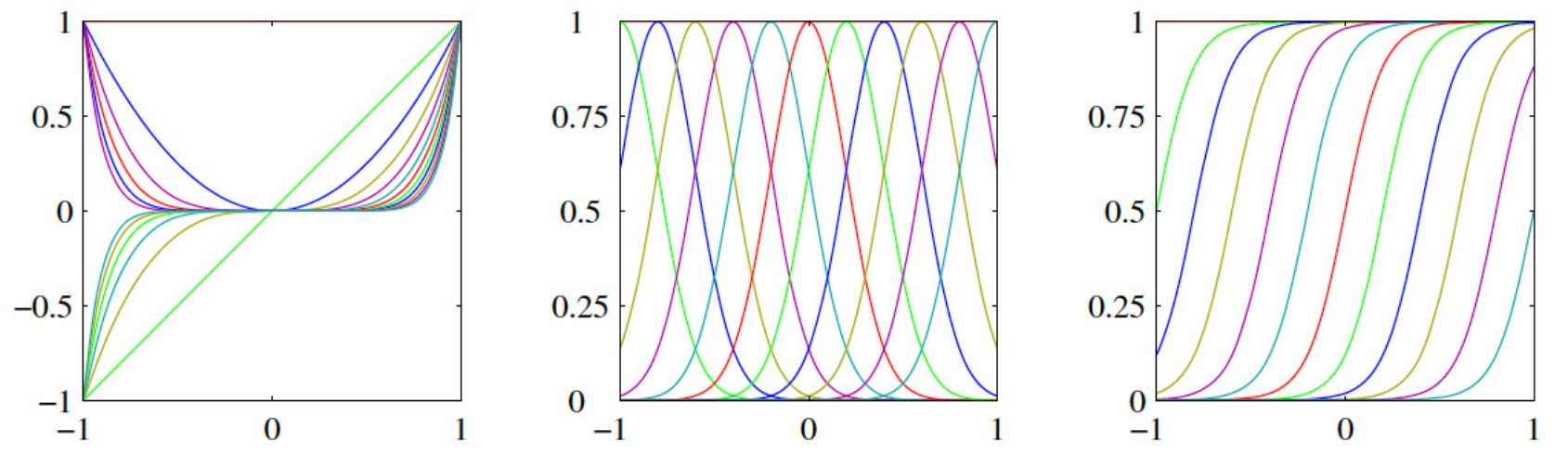
Hypothesis functions (II)

The basis functions can be selected among many possible choices. For example, most trivial is setting $\phi_j(\boldsymbol{x}) = (\boldsymbol{x})_j$, that is, just the j^{th} feature value. Moreover, since $\boldsymbol{x}$ are D-dimensional, then $M - 1 = D$.

Another option could be letting the basis functions be K-degree polynomials of the features (then $M - 1 = K^D$)

Other common choices are Gaussian functions, Sigmoidal functions, Fourier basis (basis are sinusoidal functions localized in space), or Wavelets (basis are localized both in space and frequency).

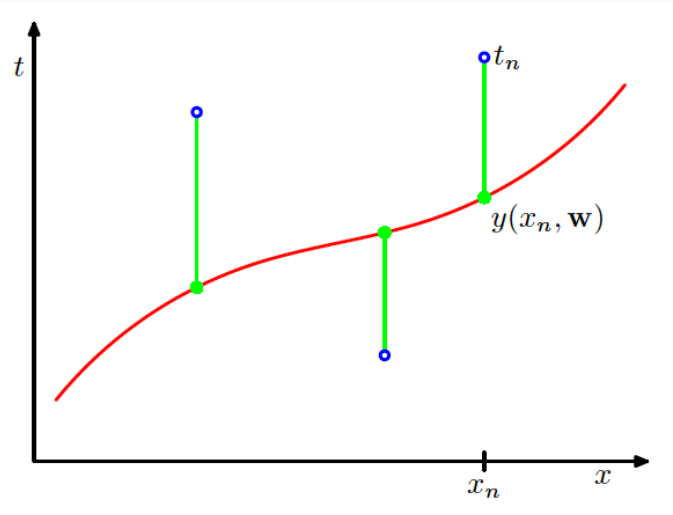
Hypothesis functions (III)



Examples of basis functions, 1-degree polynomials (left), gaussians (center), and sigmoidal (right)

Cost function

The "intuitive" way for measuring the error (the *loss*) in the curve fit is using the mean squared error (MSE) between the targets t_n and the predictions $y(\boldsymbol{x}_n, \boldsymbol{w})$ of the instances in the training set.



$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}[y(\boldsymbol{x}_n, \boldsymbol{w}) - t_n]^2 = \frac{1}{2}\sum_{n=1}^{N}[\sum_{j=0}^{M-1} w_j \phi_j(\boldsymbol{x}_n) - t_n]^2 \quad (3)$$

where the $\frac{1}{2}$ factor is added for later convenience and does not affect the optimal weights.

Cost minimization

Let $\boldsymbol{\phi}(\boldsymbol{x})$ be the vector $(\phi_0(\boldsymbol{x}), \ldots, \phi_{M-1}(\boldsymbol{x}))^T$. Then,

$$J(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} (\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) - t_n)^2 \tag{4}$$

The optimal weights $\boldsymbol{w}^*$ (corresponding to the minimum MSE) can be computed by setting the gradient $\nabla J(\boldsymbol{w}) = 0$. That is,

$$\nabla J(\boldsymbol{w}) = \sum_{n=1}^{N} (\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) - t_n) \boldsymbol{\phi}(\boldsymbol{x}_n)^T = 0 \tag{5}$$

Cost minimization (II)

Thus,

$$\boldsymbol{w}^T \sum_{n=1}^{N} \boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T = \sum_{n=1}^{N} t_n \boldsymbol{\phi}(\boldsymbol{x}_n)^T \implies \boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{w} = \boldsymbol{\Phi}^T\boldsymbol{t} \quad (6)$$

Where $\boldsymbol{t} = (t_1, \ldots, t_N)^T$ and $\boldsymbol{\Phi}$ is called the *design* matrix and is given by

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}_0(\boldsymbol{x}_1) & \boldsymbol{\phi}_1(\boldsymbol{x}_1) & \ldots & \boldsymbol{\phi}_{M-1}(\boldsymbol{x}_1) \\ \boldsymbol{\phi}_0(\boldsymbol{x}_2) & \boldsymbol{\phi}_1(\boldsymbol{x}_2) & \ldots & \boldsymbol{\phi}_{M-1}(\boldsymbol{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{\phi}_0(\boldsymbol{x}_N) & \boldsymbol{\phi}_1(\boldsymbol{x}_N) & \ldots & \boldsymbol{\phi}_{M-1}(\boldsymbol{x}_N) \end{bmatrix} \quad (7)$$

Normal equations

And the optimum is given by

$$\boldsymbol{w}^* = (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{t} \tag{8}$$

These are known as the *normal equations* for the least-squares problem and provide a closed-form expression to determine $\boldsymbol{w}^*$.

And predictions are computed as:

$$y(\boldsymbol{x}, \boldsymbol{w}^*) = (\boldsymbol{w}^*)^T \boldsymbol{\phi}(\boldsymbol{x}) \tag{9}$$

Iterative methods

The use of the normal equations can be impractical in some cases:

- Inverting matrix $\mathbf{\Phi}^T\mathbf{\Phi}$ has complexity $O(M^{2.8})$ (see Straseen complexity). Thus, it can be costly for large values of M.
- Training data may not be available in a single batch, but sequentially and, therefore, normal equations could not be used if predictions are required on the fly.

For these reasons, it is of interest to have iterative algorithms.

Gradient descent optimization

Iterative methods are based on the *gradient descent* optimization:

1. Set initial weights for $\boldsymbol{w}$ (e.g., randomly)
2. Repeat:

   $$\boldsymbol{w}^{(\text{new})} = \boldsymbol{w}^{(\text{old})} - \eta \nabla J(\boldsymbol{w}^{(\text{old})}) = \boldsymbol{w}^{(\text{old})} - \eta[\boldsymbol{\Phi}^T \boldsymbol{\Phi} \boldsymbol{w}^{(\text{old})} - \boldsymbol{\Phi}^T \boldsymbol{t}]$$

   until convergence.

The parameter η is called the *learning rate* and it controls the convergence properties of the algorithm. Besides, note that $\nabla J(\boldsymbol{w})$ has been already obtained in eq. (5) and rewritten to matrix form as in eq. (6).

Overfitting

In slide 14 polynomial of degree M=3 seems the best fit, but the loss $J(\boldsymbol{w}^*)$ is minimum for M=9 (it contains all training set). This situation is called *over-fitting*, and happens when the hypothesis does not generalize well to new points, even if it fits very well to points in the training set. A way of overcoming this problem is using a more extensive data set. But, what if more data is unavailable?

Additional insight is given in the table to the right which shows the coefficients of the best polynomials. The higher degree, the higher the fluctuations and so the coefficients.

	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Regularization

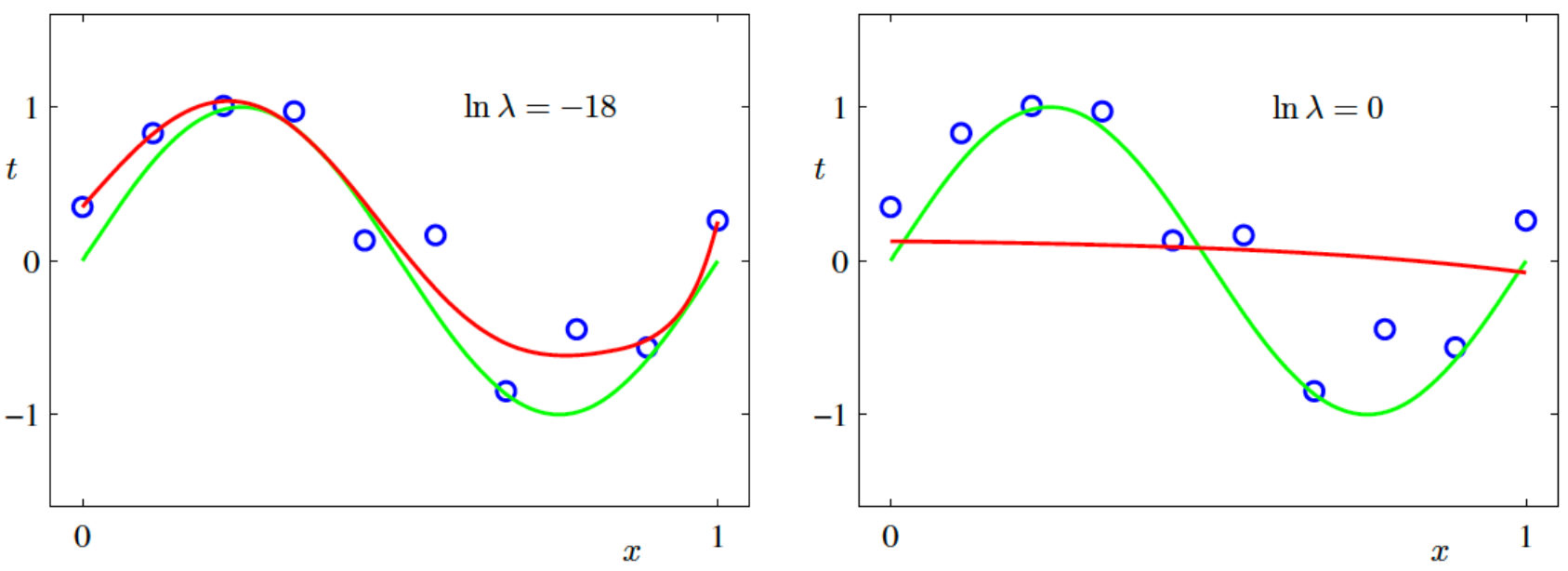
Last observation suggests to penalize solutions with large coefficient, which can be done adding a *regularization* term into the loss function given in eq. (3), leading to:

$$\widetilde{J}(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}[\sum_{j=0}^{M-1} w_j\phi_j(\boldsymbol{x}_n) - t_n]^2 + \frac{\lambda}{2}\sum_{j=0}^{M-1}|w_j|^q \qquad (10)$$

The case of $q = 2$ is called *ridge* regression, and the second term in the right-hand side of the equation above can be written as $\frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}$. It has the advantage of having a closed form version of the normal equations:

$$\boldsymbol{w}^* = (\lambda I + \boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{t} \qquad (11)$$

Regularization (II)



Ridge regression for two regularization parameters λ applied to M=9 degree polynomials.

The case of $q = 1$ is called *lasso* regression. It has the advantage that if λ is sufficiently large, some of the w coefficient are driven to zero, leading to a *sparse* model which selects which basis functions play no role.

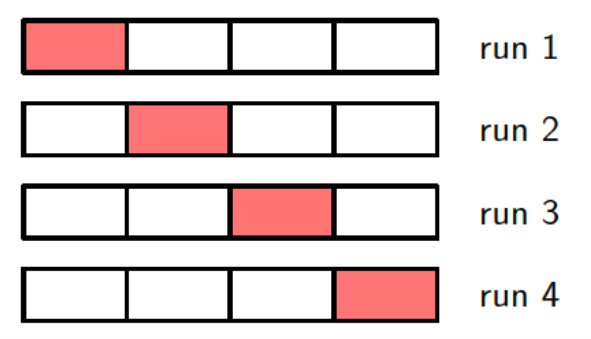
Cross validation

In a practical application, it is necessary to determine a suitable regularization parameter λ as well as other parameters (e.g., the internal parameters of the basis functions). These variables, governing the *model complexity* are called *hyperparameters*.

A suitable way for determining them is using the so-called S-fold *cross-validation*. The idea is to keep aside from the training set a randomly selected part of the instances (a *validation set*), and test on it different models trained with the remaining training data. S-fold cross-validation averages the performance by doing S independent partitions of the training set and testing the trained models against the corresponding validation sets.

Cross validation (II)

Example of S-fold validation dividing the data set in S parts of equal size. The validation set for each run is shaded in red. In each run the model is trained using the rest of the training set and tested against the independet data from the validation set.



If this procedure is applied over a limited size data set, then some over-fitting to the validation data can occur. A workaround is to keep aside a third *test set* on which the selected model is finally evaluated.

One drawback of cross-validation is that it reduces the data available to fit the model, which on small data sets is problematic. Using Bayes linear regression, this issue can be avoided, as discussed later.

Probabilistic approach

In our previous treatment, we have ignored the intrinsic random nature of the training set where the target has a noisy component as shown in the curve fitting examples of slide 14.

Indeed, the target can be assumed to be

$$t = y(\boldsymbol{x}, \boldsymbol{w}) + \epsilon$$

where ϵ is a zero mean Gaussian random variable with variance β^{-1} (β is called the *precision*). Thus,

$$p(t|\boldsymbol{x}, \boldsymbol{w}, \beta) = \mathcal{N}(t|y(\boldsymbol{x}, \boldsymbol{w}), \beta^{-1}) = \mathcal{N}(t|\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n), \beta^{-1})$$

Probabilistic approach

Hence, given a set $\boldsymbol{X}=\{\boldsymbol{x_1},\ldots,\boldsymbol{x_N}\}$ of N *independent* observations, its likelihood is:

$$p(t|\boldsymbol{X},\boldsymbol{w},\beta)=\prod_{n=1}^{N}\mathcal{N}(t_n|\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n),\beta^{-1})$$

Thus, its log-likelihood is

$$\ln p(t|\boldsymbol{X},\boldsymbol{w},\beta)=\frac{N}{2}\ln\beta-\frac{N}{2}\ln(2\pi)-\beta\frac{1}{2}\sum_{n=1}^{N}[t_n-\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x})]^2$$

The last term of the log-likelihood is $\beta J(\boldsymbol{w})$, and since $\boldsymbol{w}$ does not appear on the other terms, its maximum likelihood (ML) estimator under Gaussian noise coincides with the optimal weights obtained through the curve fitting approach, i.e., $\widehat{\boldsymbol{w}}_{\text{ML}}=\boldsymbol{w}^*$.

Bayesian approach

The Bayesian view considers the probability as a measure of *uncertainty*, unlike the classical, frequentist, paradigm. This way, the Bayesian view treats model parameters as *uncertain* variables to which a probability measure can be applied. When new evidence (i.e., a data set) becomes available, the uncertainty about the parameter can be reduced using the Bayes' rule:

Given random variables a and b, and $p(a|b)$, then

$$p(b|a) = \frac{p(a,b)}{p(a)} = \frac{p(a|b)p(b)}{p(a)} \tag{12}$$

where $p(a)$ is obtaining by marginalizing $p(a,b)$ over the distribution of b. In this setup, $p(b)$ is called the *prior* distribution of b and $p(b|a)$ the *posterior* distribution of b given the evidence a.

Bayesian approach (II)

For example, in our linear regression problem, a *prior* multi-variate zero-mean isotropic Gaussian distribution[2] with precision α can be assumed for the weights $\boldsymbol{w}$, so that, $p(\boldsymbol{w}) = \mathcal{N}(0, \alpha^{-1} I)$.

Given evidence $\boldsymbol{X}, \boldsymbol{t}$ the posterior distribution of $\boldsymbol{w}$ is:

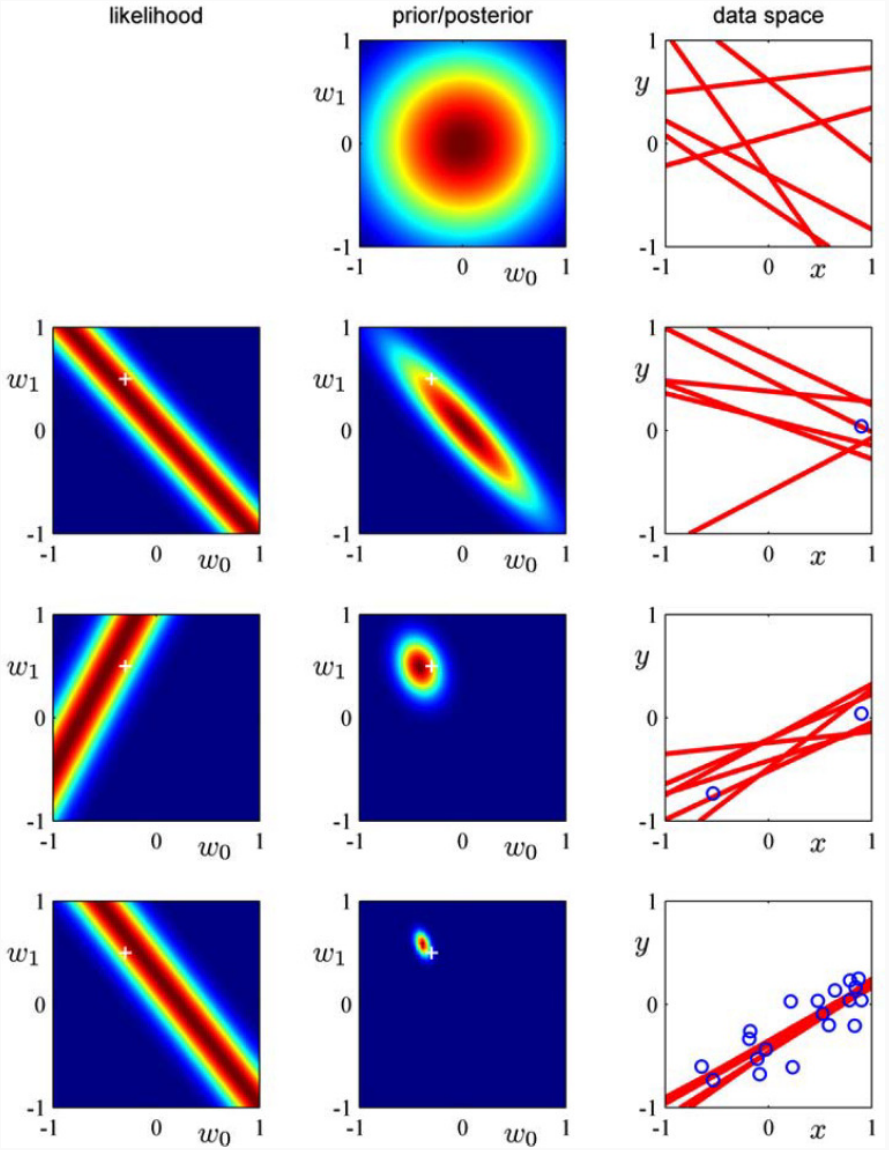
$$p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{t}) = \mathcal{N}(\boldsymbol{w}|\beta \boldsymbol{S}_N \boldsymbol{\Phi}^T \boldsymbol{t}, \boldsymbol{S}_N)$$

where $\boldsymbol{S}_N = (\alpha I + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1}$.

The maximum a posteriori estimator (MAP) of $\boldsymbol{w}$ is given by the maximization of the previous expression. Noteworthy, this maximization coincides with the minimization of the regularized loss $\widetilde{J}(\boldsymbol{w})$ with $\lambda = \frac{\alpha}{\beta}$.

[2] See Multi-variate Gaussian Distribution appendix in AV.

Bayesian approach (III)



Example of sequential Bayesian learning for a single featured linear model given by $y(x, \boldsymbol{w}) = w_0 + w_1 x$ with target distribution $p(t|x, \boldsymbol{w}) = \mathcal{N}(w_0 + w_1 x, \beta^{-1} I)$.
The prior of $\boldsymbol{w}$ is Gaussian and each new point (x, t) (blue circle) has a likelihood $p(t|x, \boldsymbol{w})$. Thus, $p(\boldsymbol{w}|t) \propto p(t|x, \boldsymbol{w}) p(\boldsymbol{w})$, whose normalization yields to the posterior distribution. This distribution becomes the priori one for the next iteration, and so on.
The right-most column shows hypothesis functions drawn from the $\boldsymbol{w}$ distribution.
As new evidence (points) become available, the uncertainty (in the distribution of $\boldsymbol{w}$) is reduced.

Bayesian approach (IV)

Actually, rather than on $\boldsymbol{w}$ we are interested in the predictive distribution defined by $p(t|\boldsymbol{t})$ for new values of $\boldsymbol{x}$, which correspond to the marginalization of $p(t|\boldsymbol{t}, \boldsymbol{w})$ over the distribution of $\boldsymbol{w}$. That is,

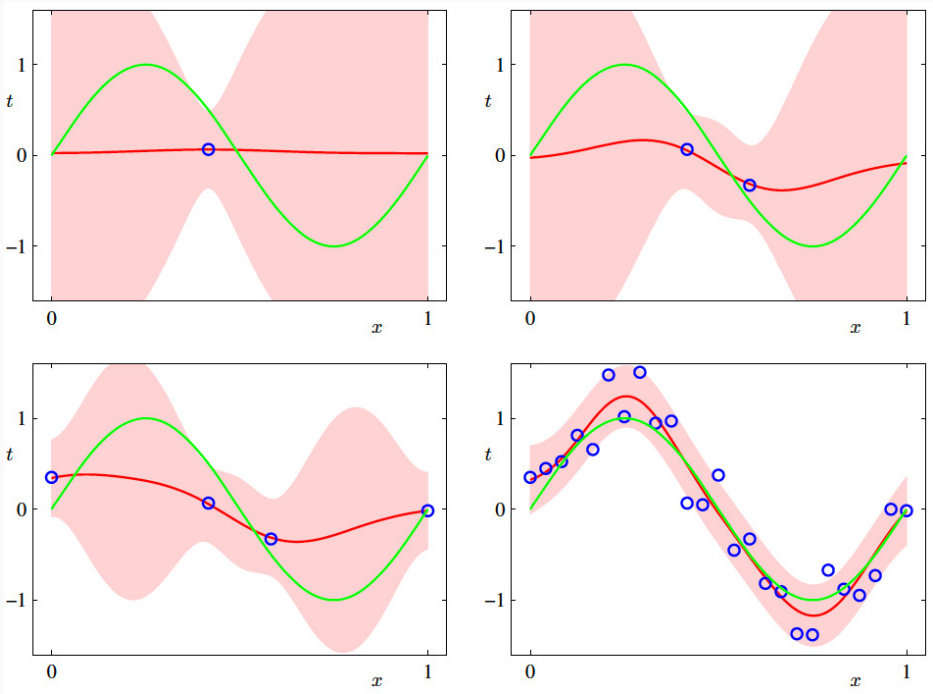
$$p(t|\boldsymbol{t}, \boldsymbol{w}) = \int p(t|\boldsymbol{w}) p(\boldsymbol{w}|\boldsymbol{t}) d\boldsymbol{w}$$

Solving previous integral (see Bishop 3.3.2) leads to

$$p(t|\boldsymbol{t}, \boldsymbol{w}) = \mathcal{N}(t|(\lambda I + \boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{t} \boldsymbol{\phi}(\boldsymbol{x}), \beta^{-1} + \boldsymbol{\phi}^T(\boldsymbol{x}) \boldsymbol{S}_N \boldsymbol{\phi}(\boldsymbol{x}))$$

The MAP estimator of t is the Gaussian mean, and, as discussed, matches the one from regularized regression with parameter $\lambda = \frac{\alpha}{\beta}$.

Bayesian approach (V)



The predictive distribution is important because it allows to construct confidence intervals for the prediction. In the figure the confidence regions spanning one standard deviation either side of the mean are shown. The basis is composed by 9 Gaussian functions.

Bayesian approach (VI)

In the curve fitting treatment, the suitability of a prediction has to be assessed by evaluating the performance over a validation set. With the predictive distribution, confidence regions can be directly constructed.

It is also possible to apply a Bayesian treatment which considers a mixture of L models $\{\mathcal{M}_l\}$, and each defining its corresponding family parameters, and even priors over α and β. Interested students may consult Bishop 3.4 and 3.5.

The bias-variance trade-off

Further insight into the models' nature can be obtained by analyzing how the loss can be decomposed. Let us note that the data set can be actually considered as being random, and therefore the associated loss shall be understood as a random variable too.

Therefore, it is possible to compute the expectation of this loss (with respect to the random distribution of the data set), yielding to:

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$

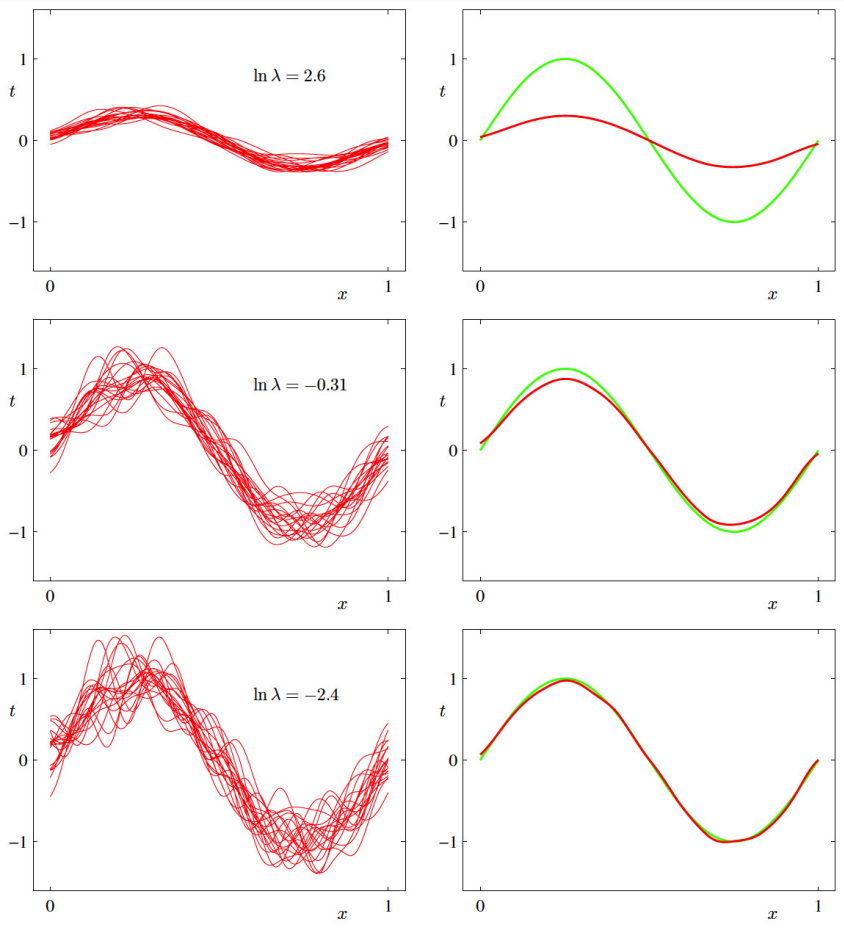
The bias-variance trade-off (II)

These terms are related to different causes:

- The bias is due to the lack of flexibility of the model, such as a small set of basis functions. A high-bias model will most likely underfit the training data.
- On the other hand, the variance is due to overfitting, that is because the model is too flexible.
- The noise is the error of the data source, and as such, irreducible by the training algorithm.

The complexity (flexibility) of the model is controlled by the regularization parameter λ. The optimal solution requires a trade-off between the bias and the variance and corresponds with intermediate λ values.

The bias-variance trade-off (III)



Bias/tradeoff example based on 100 data sets, each having $N = 25$ data points. The model has $M = 25$ (24 Gaussian basis functions and the bias parameter - not to be confused with the statistical bias). The left column shows the result of fitting the model to the data sets for various values of $\ln \lambda$. The right column shows the corresponding average of the 100 fits (red) along with the sinusoidal function from which the data sets were generated (green). The best choice is the intermediate value of λ. A high λ reduces model complexity, so it will make predictions with small variance (but their mean is far from the actual curve). On the other hand, a small λ value allows models with high complexity, and high variance (although, the mean of the predictions nearly matches the original curve).

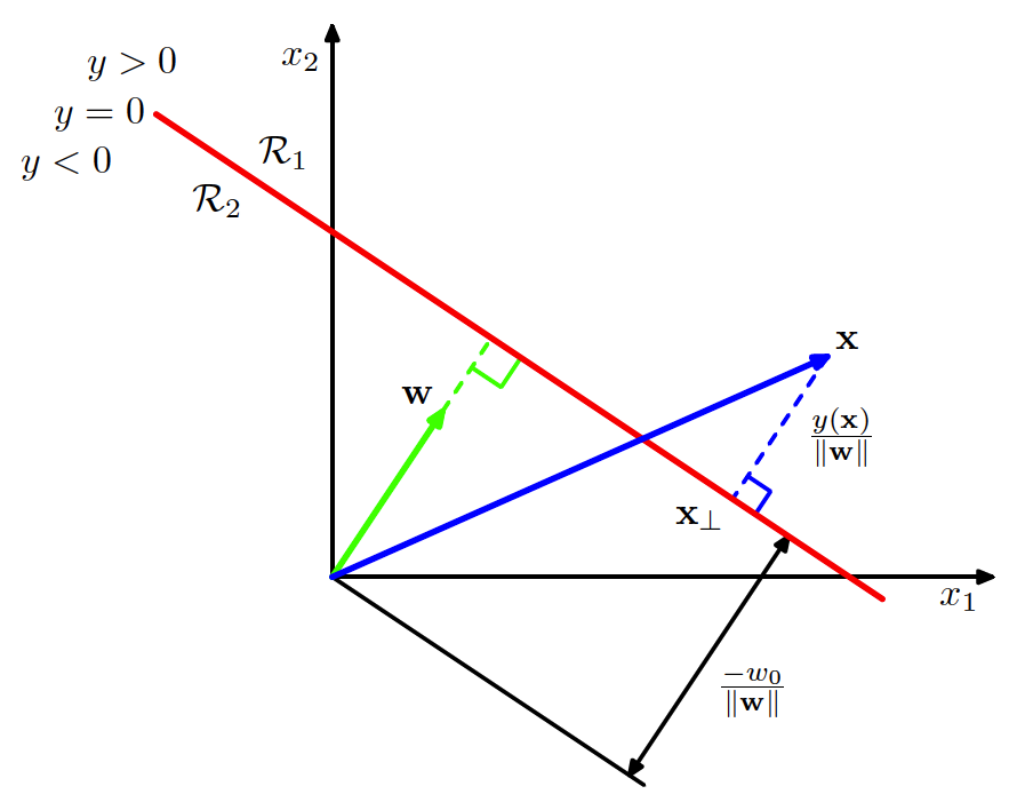
Linear classification models

Linear classification models

In *classification* problems the goal is to take a D-dimensional input vector $\boldsymbol{x}$ and assign it to one of K classes $\mathcal{C}_k$ for $k{=}1, \ldots, K$.

The regions assigned to different classes are separated by *decision boundaries*. The term *linear classification* is used when these decision boundaries are $(D-1)$-dimensional hyperplanes. If a data set can be separated without misclassification errors by some $(D-1)$-hyperplane, then it is called *linearly separable*.

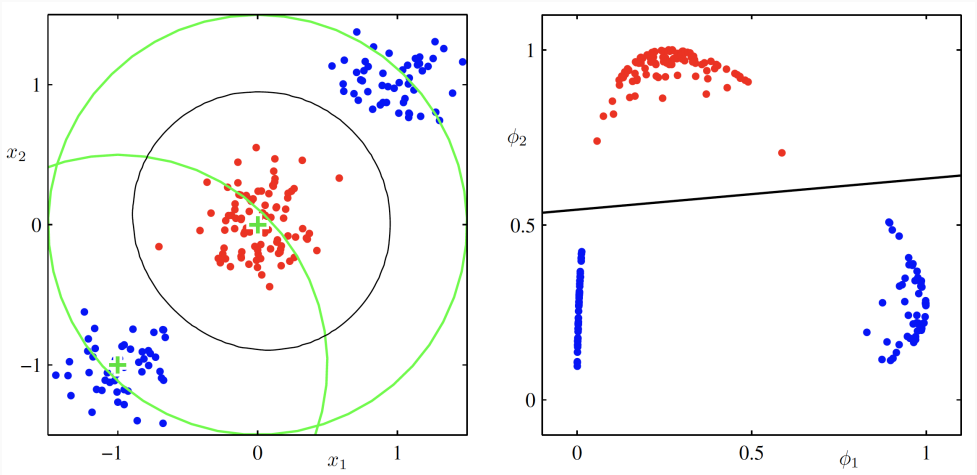
Linear classification models (II)



Linear decision boundary (red) for a 2-dimensional feature space is a 1-dimensional hyperplane (just a line). If D=3 it would be a plane. A $(D-1)$-dimensional hyperplane is given by equation $w_0 + \boldsymbol{w}^T\boldsymbol{x} = 0$.

Linear classification models (III)

We assume the input transformation to a *feature* space using a set of fixed (non-linear) $M-1$ basis functions $\{\phi_m(\boldsymbol{x})\}$, for $m=1,\ldots,M-1$ and the dummy function $\phi_0(\boldsymbol{x}) = 1$. Thus, in vector notation, $\boldsymbol{\phi} = (\phi_0(\boldsymbol{x}),\ldots,\phi_{M-1}(\boldsymbol{x}))^T$. The decision boundaries are linear in the feature space, but nonlinear in the original input space.

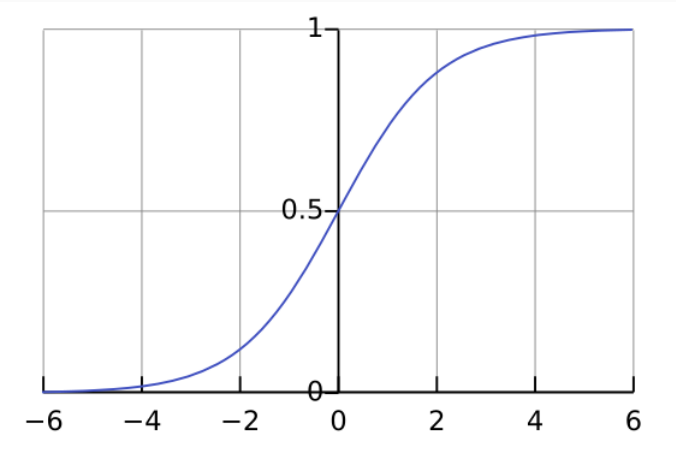


Example of linear non-separable data set, which can be transformed (using Gaussian basis functions ϕ_1 and ϕ_2) from the original input space (left) to a feature separable space (right).

Logistic regression

Logistic regression is a linear binary classification model which models the posterior probability of $\mathcal{C}_1$ using the *logistic sigmoid* function acting on a linear function of the feature vector:

$$p(\mathcal{C}_1|\boldsymbol{\phi}) = \sigma(\boldsymbol{w}^T\boldsymbol{\phi}) \quad p(\mathcal{C}_2|\boldsymbol{\phi}) = 1 - p(\mathcal{C}_1|\boldsymbol{\phi}) \tag{13}$$



$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

$$\frac{\mathrm{d}\sigma(a)}{\mathrm{d}a} = \sigma(a)(1 - \sigma(a))$$

As shown this function takes values in $[0, 1]$, which makes it good candidate for modeling a probability. In ML, functions which, like the logistic sigmoid, define the output given the set of features are called *activation functions*.

Logistic regression (II)

Given a labeled data set $\boldsymbol{X}=\{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N\}$, $\boldsymbol{t}=\{t_1,\ldots,t_N\}$, where $t_n=1$ if $\boldsymbol{x}_1\in\mathcal{C}_1$, or 0 otherwise. The logistic regression model has M parameters $\{w_m\}$, for m=0, $\ldots$, $M-1$. Let $y_n=\sigma(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n))$. Then, the likelihood is:

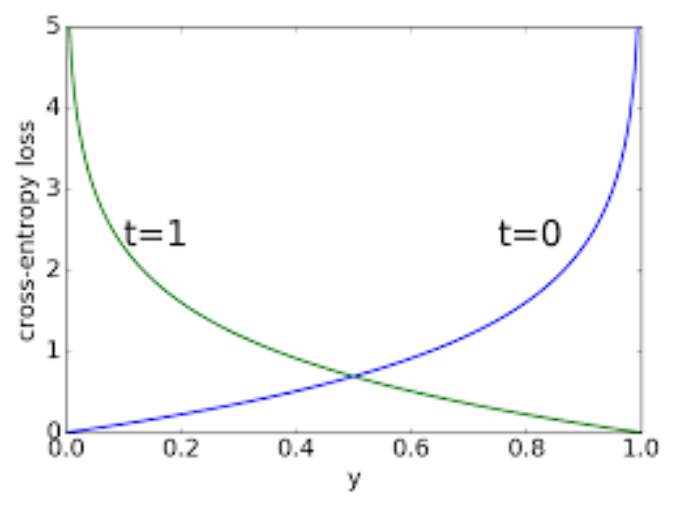
$$p(\boldsymbol{t}|\boldsymbol{w})=\prod_{n=1}^{N}y_n^{t_n}(1-y_n)^{1-t_n}$$

Then, the log likelihood is:

$$J(\boldsymbol{w})=-\ln p(\boldsymbol{t}|\boldsymbol{w})=-\sum_{n=1}^{N}[t_n\ln y_n+(1-t_n)\ln(1-y_n)] \quad (14)$$

This loss function is called *cross-entropy*, and is used often to measure the similarity between two probability distributions.

Logistic regression (III)



When target t_n is 1 (class $\mathcal{C}_1$) cross-entropy penalizes small values of y_n (green curve). For points in class $\mathcal{C}_2$ cross-entropy is high for values of y_n close to 1 (blue curve).

Thus, the gradient of the loss function (calculated by using the derivative of the logistic function) is:

$$\nabla J(\boldsymbol{w}) = \sum_{n=1}^{N} (y_n - t_n)\boldsymbol{\phi}(\boldsymbol{x}_n)$$

Logistic regression (IV)

As seen in the case of linear regression, the maximum likelihood can be computed from a closed-form expression. However, for logistic regression this is not longer possible due to the non-linearity of the logistic sigmoid function. Nevertheless, since $J(\boldsymbol{w})$ is concave, it has a unique minimum. An efficient solving technique is the iterative Newton-Raphson method:

$$\boldsymbol{w}^{(\text{new})} = \boldsymbol{w}^{(\text{old})} - \boldsymbol{H}^{-1} \nabla J(\boldsymbol{w}^{\text{old}})$$

with $\boldsymbol{H}$ being the Hessian matrix of $J(\boldsymbol{w})$. As an exercise, try to apply this method to solve the linear regression problem (the result will require a single step and shall coincide with the normal equations).

Logistic regression (V)

With matrix notation the gradient and the Hessian of $J(\boldsymbol{w})$ are:

$$\nabla J(\boldsymbol{w}) = \boldsymbol{\Phi}^T(\boldsymbol{y} - \boldsymbol{t})$$
$$H = \nabla\nabla J(\boldsymbol{w}) = \boldsymbol{\Phi}^T \boldsymbol{R}\boldsymbol{\Phi}$$

being $\boldsymbol{y} = (y_1, \ldots, y_N)^T$ and $\boldsymbol{R}$ a $N \times N$ diagonal matrix such that element $(\boldsymbol{R})_{nn} = y_n(1 - y_n)$. Thus, the update rule for Newton-Raphson is:

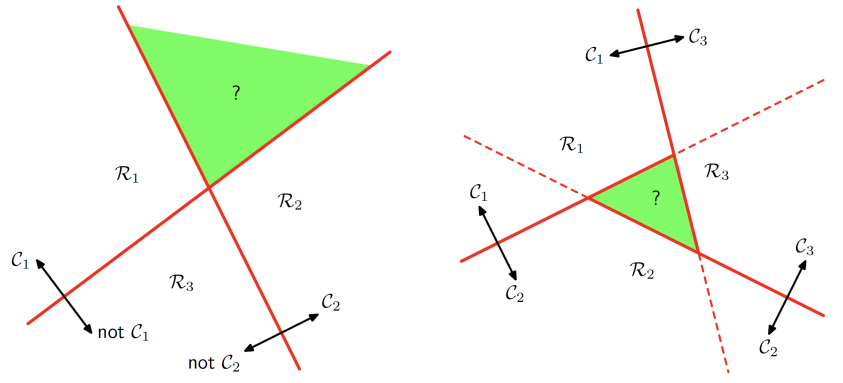
$$\boldsymbol{w}^{(\text{new})} = \boldsymbol{w}^{(\text{old})} - (\boldsymbol{\Phi}^T \boldsymbol{R}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T(\boldsymbol{y} - \boldsymbol{t})$$

This method is known in the literature as Iterative reweighted least squares (IRLS).

Multi-class logistic regression

A simple way to address multi-class classification could be considering collections of binary classifiers:

- *One-vs-all*. For each class a binary classifier is constructed assuming that all points outside the given class belong to the alternative class.
- *One-vs-one*. For each pair of classes a binary classifier is constructed.



Both schemes have regions of uncertainty (green)

Multi-class logistic regression (II)

Instead, it is possible to build a model where a collection of weights vectors $\{\boldsymbol{w}_k\}$ control the posterior probabilities of the classes through the *softmax* activation function:

$$p(\mathcal{C}_k|\boldsymbol{\phi}(\boldsymbol{x})) = y_k(\boldsymbol{\phi}(\boldsymbol{x})) = \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}} = \frac{e^{\boldsymbol{w}_k^T \boldsymbol{\phi}(\boldsymbol{x})}}{\sum_{j=1}^{K} e^{\boldsymbol{w}_j^T \boldsymbol{\phi}(\boldsymbol{x})}}$$

For this case, let the target be represented by *1-of-K* notation, e.g., if K=5 then a pattern from class 2 would be given by target[3]vector $\boldsymbol{t} = (0, 1, 0, 0, 0)^T$. Besides, let $\boldsymbol{y}_n$ denote the vector $(y_1(\boldsymbol{\phi}(\boldsymbol{x}_n)), \ldots, y_K(\boldsymbol{\phi}(\boldsymbol{x}_n)))^T$.

[3]This representation also permits probabilistic targets, where $(\boldsymbol{t})_k = p(\boldsymbol{x} \in \mathcal{C}_k)$.

Multi-class logistic regression (III)

Proceeding like in the binary case, the loss function can be expressed as:

$$J(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K) = -\ln p(\boldsymbol{T}|\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K) = -\sum_{n=1}^{N}\sum_{k=1}^{K}(\boldsymbol{t}_n)_k \ln(\boldsymbol{y}_n)_k$$

which is called the (multi-class) cross-entropy.

Moreover, the gradients with respect to each $\boldsymbol{w}_k$ are:

$$\nabla_{\boldsymbol{w}_j} J(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K) = -\sum_{n=1}^{N}[(\boldsymbol{y}_n)_j - \boldsymbol{t}_n)_j]\boldsymbol{\phi}(\boldsymbol{x}_n)$$

Performance measurements

Performance measurements

Before continuing it is important to discuss some other common performance measurements used in the ML literature.

In regression problems, the most used is the MSE (already discussed). Its root is also used (Root MSE) to have the error in the same units as the regressor output. Other distances rather than euclidean are also possible (see slide 59 for some examples).

The classifier's evaluation is rich in options. Besides the cross-entropy discussed earlier, the *accuracy* is frequent. It is defined as the ratio of well-classified instances to the number of instances (in a given test set). Note that for a K-class classifier, the minimum accuracy is $1/K$ (a purely random decision).

Confusion matrix

If the test set is skewed (e.g. if 90% of instances belongs to class 1 and 10% to class 2), the accuracy is a poor choice, cause a dumb classifier which always selects the class 1 will have an accuracy of 0.9.

A better way to evaluate a classifier is looking at its *confusion matrix* which shows the for a row i and column j the ratio of instances which belongs to class i and is labeled as the j class. The diagonal provides the ratio of correct classification for each class.

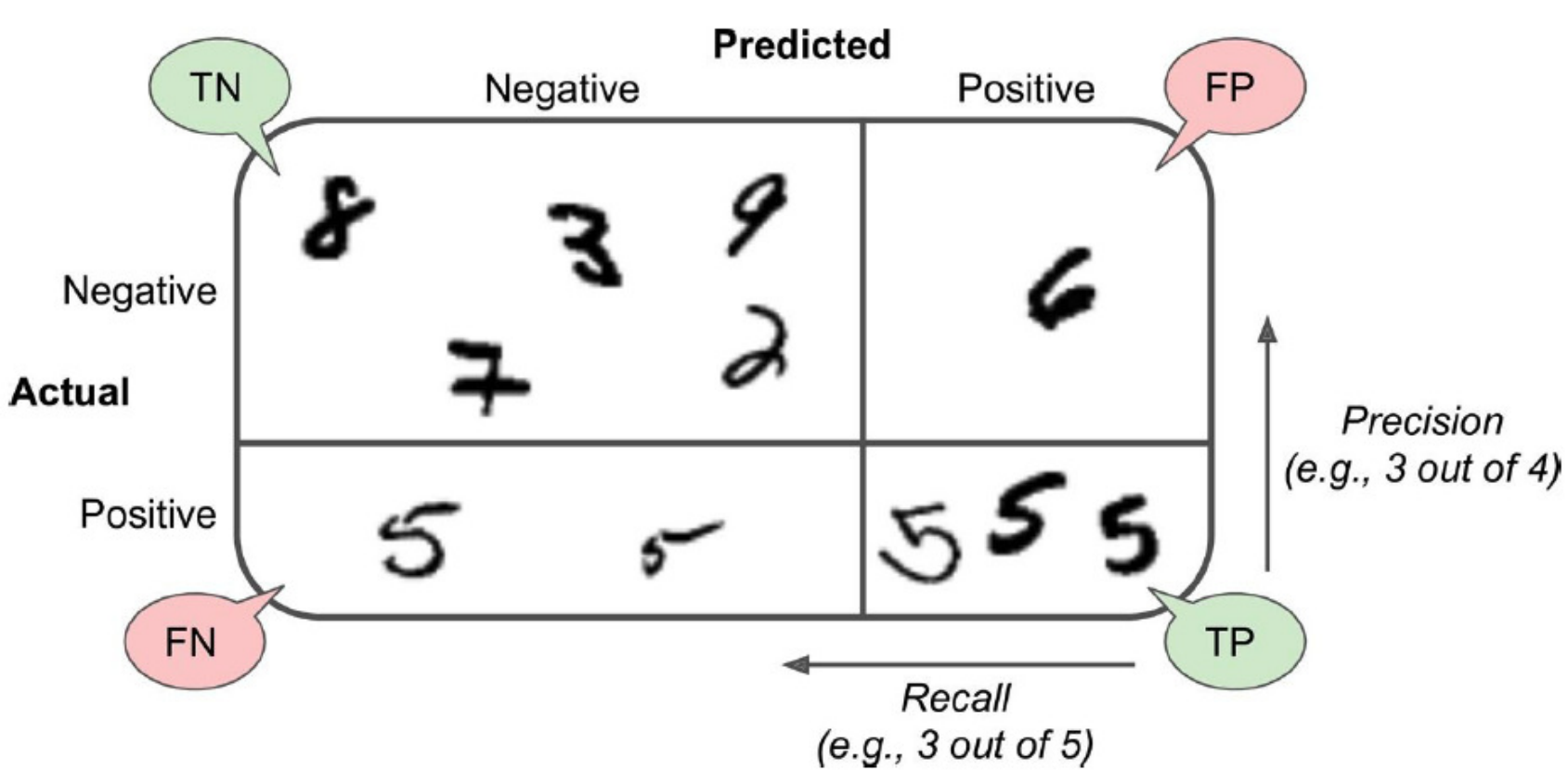
For binary classifiers, several additional performance measurements related to the confusion matrix are used. The following slide shows the great figure summarizing all measurements from Wikipedia's entry.

Confusion matrix (II)

		True condition				
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$	F_1 score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$		

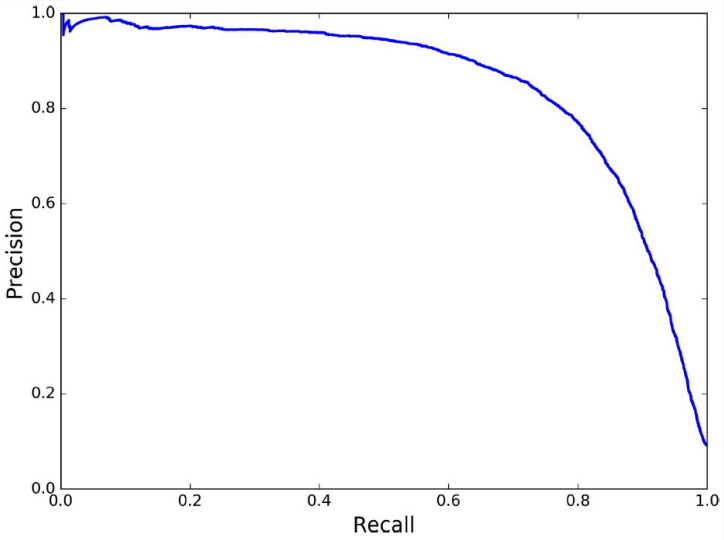
Performance measurements for a binary classifier. In ML literature the most used are the *precision* and the *recall*.

Precision and recall

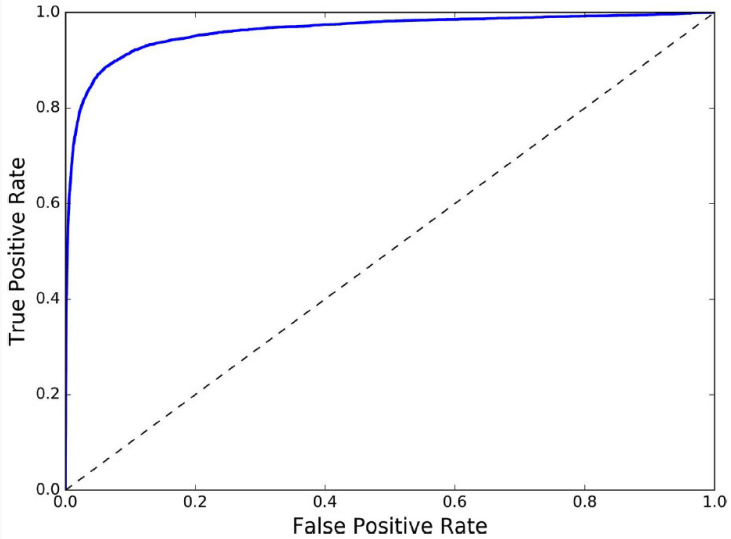


Example of precision and recall calculation for a binary classifier on the MNIST data aimed at detecting '5's and non-'5's.

Precision and recall (II)



Classifiers have a trade-off between precision and recall (a high precision is associated to a low recall, and vice versa), this kind of plot is called the precision/recall curve.



The receiver operating characteristic (ROC) curve plots the recall against the false positive rate. As can be seen there is also a trade-off between both measurements.

K-Nearest neighbors (K-NN)

K-Nearest neighbors

K-Nearest neighbors is a nonlinear classification method which assigns a point $\boldsymbol{x}$ to the class with the majority vote among its k-nearest neighbors.

This model can be justified from a Bayesian perspective. Let

$$p(\boldsymbol{x}|\mathcal{C}_k) = \frac{K_k}{N_k V} \tag{15}$$

with N_k the number of points belonging to class k in the data set, K_k the number of points belonging to class k in the smallest closed ball containing the K nearest neighbors, and V the measure (volume) of this closed ball[4].

[4] If more than K points are inside of this ball the method selects at random between the points belonging to the frontier.

K-Nearest neighbors (II)

Since the unconditional density is:

$$p(\boldsymbol{x}) = \frac{K}{NV} \tag{16}$$

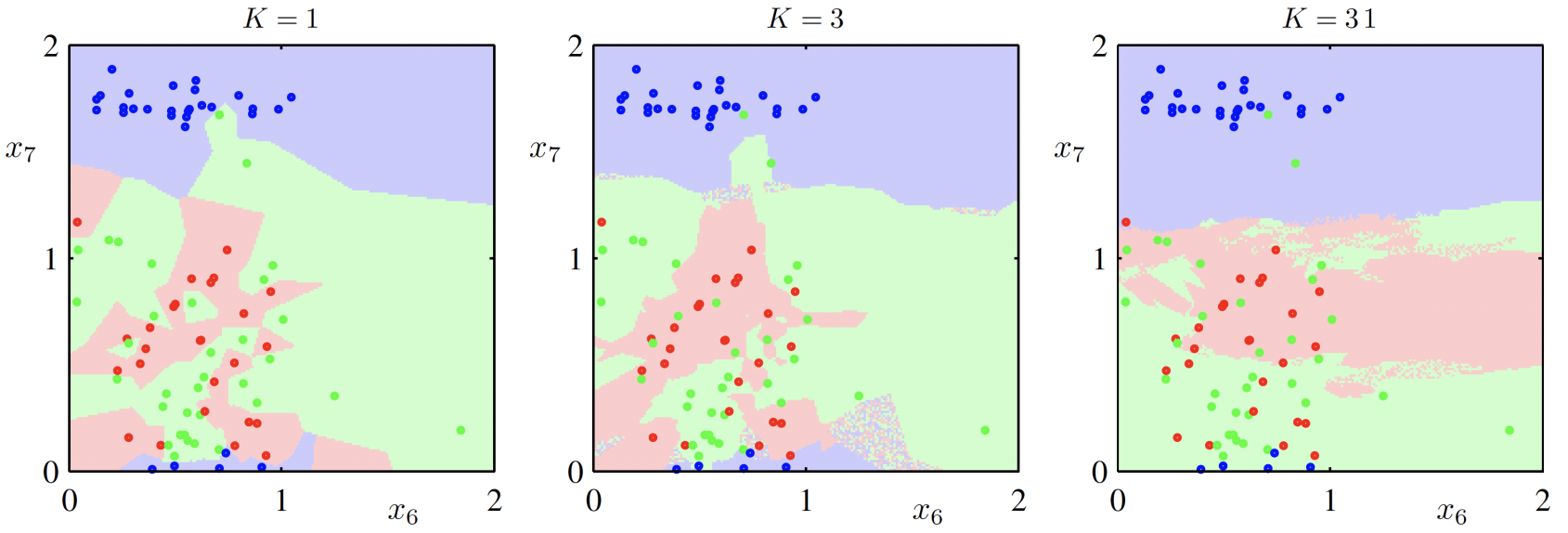
and the priors are:

$$p(\boldsymbol{x}|\mathcal{C}_k) = \frac{N_k}{N} \tag{17}$$

Hence, the posterior probability is:

$$p(\mathcal{C}_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\boldsymbol{x})} = \frac{K_k}{K} \tag{18}$$

K-Nearest neighbors (III)

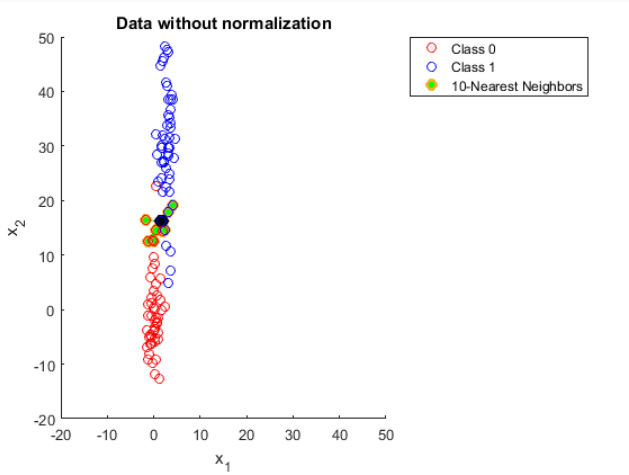


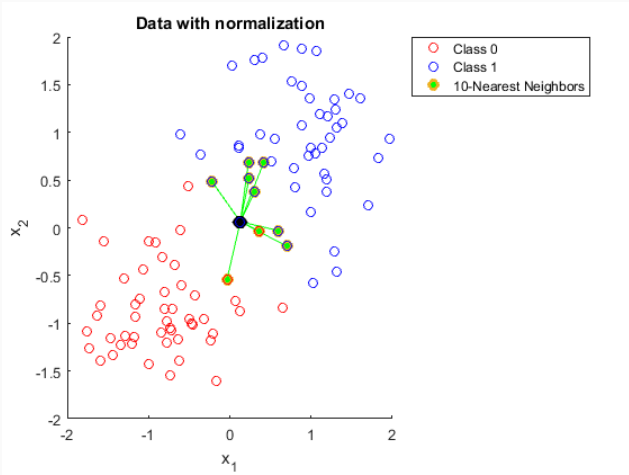
Decision regions using K-NN with K=1, 3 and 31. The decision boundaries are non-linear. Class C_k is selected as the one maximizing the posterior probability $\frac{K_k}{K}$.

K-Nearest neighbors (IV)

Implementation notes:

- Non-euclidean distances are frequently used, e.g., Manhattan or Cosine (see link).
- To avoid heterogeneous scale effects in different dimensions (see figure), it is mandatory to perform a *normalization* process, which brings points to (approximately) the same scale.





K-Nearest neighbors (V)

Implementation notes:

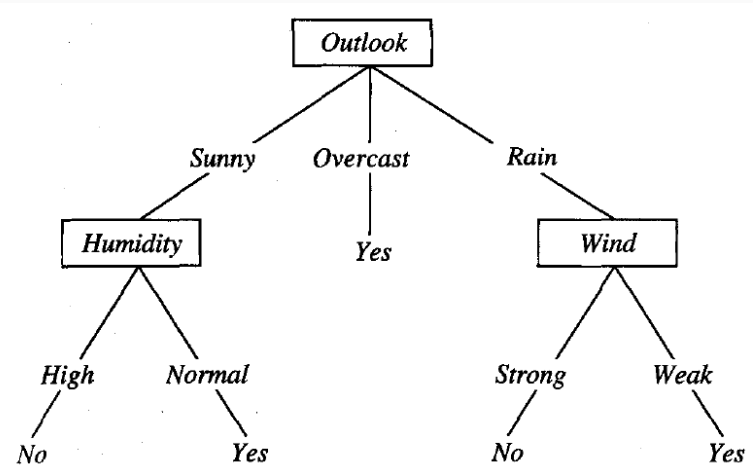
- The algorithm can be used in the original input space, although if its dimensionality is high it is recommended to reduce it by transforming data to a feature space which can be selected manually or automatically (see Unit 3).
- K-NN is sensitive to outlier presence.
- K-NN can be easily modified for regression problems (e.g., by computing an average of the neighbors' targets weighted by the respective distances to the query point).
- K-NN is a *non-parametric*, *instance-based*, method. For classifying a new point, it is necessary to compute the distance to points in the data set, so they have to be stored.

Decision trees

Decision trees

Decision tree (DT) are non-linear algorithms for classification and regression. They are considered *white-box* algorithms since it is easy for a human to understand why a given decision has been made, in contrast to *black-box* methods like ANNs. DTs can also be used when the data set contains errors or missing feature values.

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No



The core idea of DTs is to make predictions following a tree structure like the one shown in the figure. The output is the decision on whether to play or not a tennis game.

Decision trees (II)

DTs are best suited to classification problems whose input variables are *categorical* features, i.e., each variable takes values on a small number of disjoint possible choices (e.g., in the example DT, "weather" has three possible values "Sunny","Cloudy" or "Rainy").

A DT algorithm selects at each step which feature (of those not considered before) classifies better the training examples. Then, for each possible value of that feature, a new node is created in the tree, and the process continues until no more features left.

Information gain

A common way to select the best feature at each step is to use the *information gain* concept.

Given a random variable which takes K possible outputs $\{o_k\}$, for $k{=}1, \ldots, K$, its *entropy* is given by:

$$H = -\sum_{k=1}^{K} p_k \log_2 p_k \tag{19}$$

where p_k denotes the probability that the random variable takes value o_k. Entropy is a central concept in information theory, measuring how much *information* or *uncertainty*, a new sample drawn from the variable will provide (e.g., flipping a coin has 1-bit entropy, rolling a 6-face dice has $\sim$2.6-bit entropy).

Information gain (II)

Given the labeled data set $\boldsymbol{X}$, $\boldsymbol{t}$ its entropy $H(\boldsymbol{X}, \boldsymbol{t})$ can be computed from ML estimations $\widehat{p_k} = N_k/N$ (that is, the ratio of each label in the training set).

Besides, the m^{th} feature takes values on a set of discrete values $\boldsymbol{v}_m = \{v_1, \ldots, v_K\}$, where K depends on the specific feature m. Let $\boldsymbol{X}, \boldsymbol{t}|_{(\boldsymbol{x})_m = v}$ denote the data set comprising only the instances where the m^{th} feature takes value v. The *information gain* (also called *mutual information*) associated to this feature is:

$$I|_{(\boldsymbol{x})_m} = H(\boldsymbol{X}, \boldsymbol{t}) - \sum_{\forall v \in \boldsymbol{v}_m} p((\boldsymbol{x})_m = v) H(\boldsymbol{X}, \boldsymbol{t}|_{(\boldsymbol{x})_m = v}) \qquad (20)$$

where $p((\boldsymbol{x})_m = v)$ is estimated as $\#(\boldsymbol{X}, \boldsymbol{t}|_{(\boldsymbol{x})_m = v})/\#(\boldsymbol{X}, \boldsymbol{t})$, where $\#S$ denotes the number of instances in set S.

Information gain (III)

DT algorithm selects at each step the feature that *maximizes* the information gain. In other words, the feature whose knowledge reduces the most uncertainty about the final output.

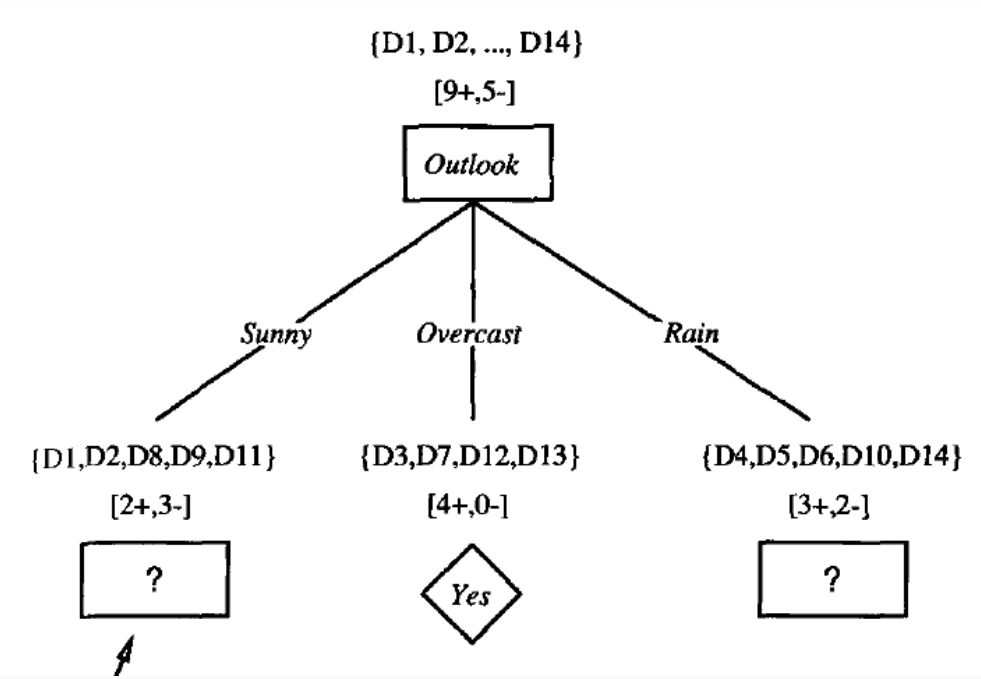
As an example, with the data of the table in slide 61, the total entropy is $H = 0.94$ bit (9 'Yes' and 5 'No'). The information gains are:

- $I|_{\text{outlook}} = 0.246$ bit
- $I|_{\text{temperature}} = 0.029$ bit
- $I|_{\text{humidity}} = 0.151$ bit
- $I|_{\text{wind}} = 0.048$ bit

Thus, the feature selected is the "outlook".

Information gain (IV)

Then, the DT has as root node this feature. For each possible value of it a new branch is opened and the process repeated. A leaf is a node where all the outputs of the data set are the same.



Tree overfitting

DT can be subject to overfitting. An effective way to counteract it is by *pruning* the tree and checking its performance on a validation set. Pruning consist of removing the subtree rooted at some particular node, making it a leaf node, and assigning it the most common classification of the training examples affiliated with that node. Nodes are removed if the resulted tree performs no worst than the original tree over the validation set.

The usual performance measurement is the *accuracy* of the prediction (see slide 51).

Tree overfitting (II)

If the data set is limited, instead of using a validation set, it is possible to estimate the real accuracy (outside of the training set) by constructing a confidence interval and getting its lower bound. For that, the real accuracy can be assumed to be Binomially $\mathcal{B}(p, n)$ distributed with $p =$ observed accuracy and n the number of training examples (if $n \gg 0$ the Binomial converges to a Gaussian distribution).

Implementation notes

Continuous features can be added by creating categorical variables in the decision boundary sets. For example:

Temperature:	40	48	60	72	80	90
PlayTennis:	No	No	Yes	Yes	Yes	No

the boundaries can be created halfway 48-60 and 80-90 to create two new categorical features $\text{temperature}|_{>54}$ and $\text{temperature}|_{>85}$, and removing the continuous one.

Besides, if some value is missed in the data set, it is common to estimate it by using different methods like the mode, the mode subject to the set of instances with the same label, or probabilistically by drawing a sample using the feature distribution.

Implementation notes (II)

DTs operation is a sequential procedure where the data required to evaluate a node can be obtained at that moment (and perhaps that particular data is unnecessary if the DT reaches a leaf before). This suggests developing DTs with associated costs to each node, aimed at reducing the global DT cost (the nodes visited in the tree), e.g., the tests performed in clinical diagnosis.

A simple way to obtain DT with minimal cost (and comparable accuracy) is modifying the information gain of a feature I with cost C to I^2/C.

Implementation notes (III)

Other measures different than the information gain are common:

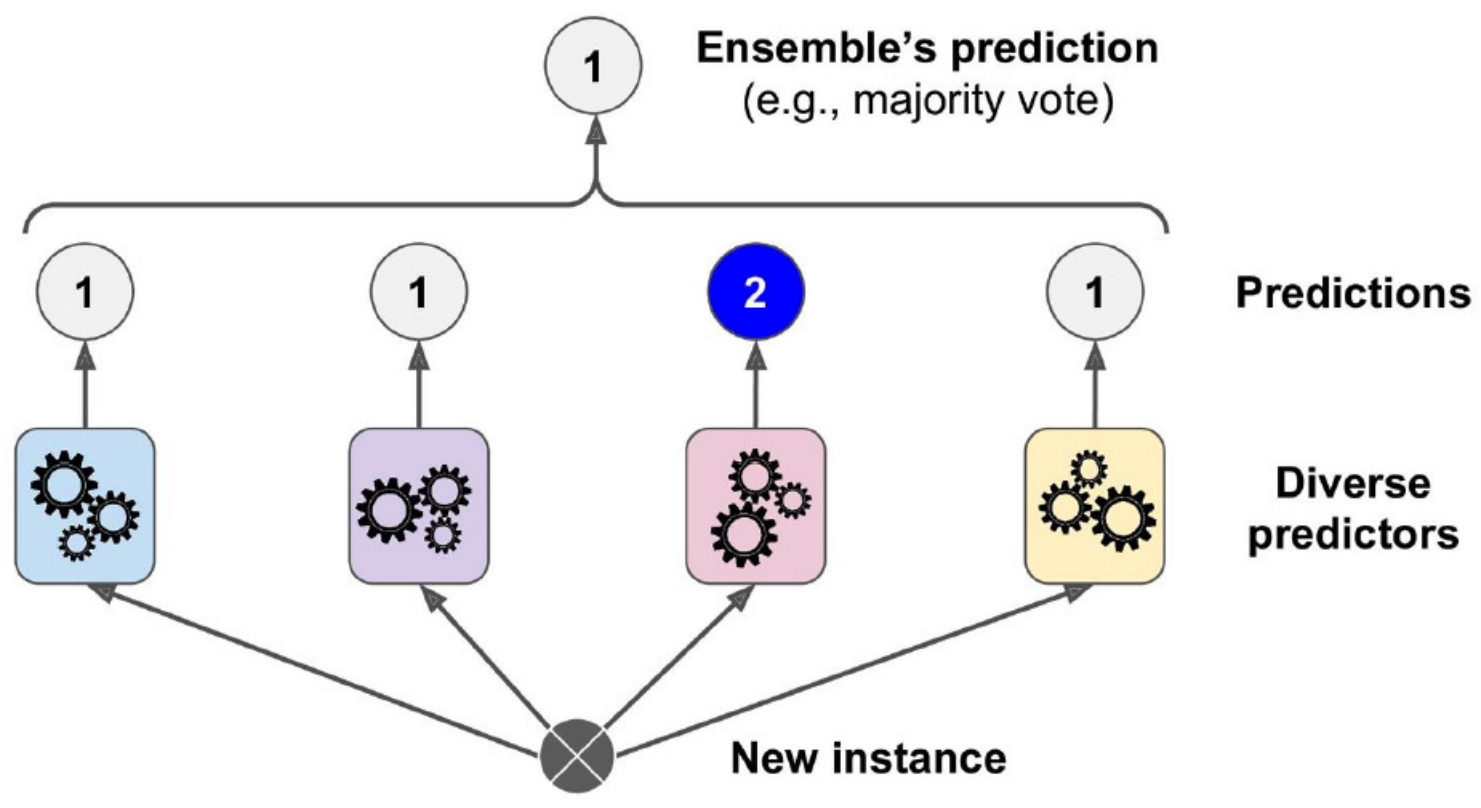
- Gain ratio. To avoid that features which takes many values are selected with priority and have lower impact in the prediction (e.g., a "Date" attribute added to the Tennis data set).
- Gini impurity:

$$G = 1 - \sum_{k=1}^{K} p_k^2 \tag{21}$$

Random forest

Ensembles

An *ensemble* is a structure which combines many classifiers and makes predictions as the majority vote of all classifiers.



Ensembles (II)

The idea is motivated by the law of the large numbers. Let assume K *independent* binary classifiers are available each making predictions with a (very low) accuracy (e.g., say 51%).

If $K = 1000$ and a majority vote is selected, by the law of the large numbers, it is expected that 510 classifiers will be correct against 490 wrongs. Indeed, the probability that the right classifiers outnumber the wrong ones is about 75% (try to compute it as an exercise). If $K = 10000$ this probability climbs over 97%.

In summary, it would be possible to achieve a highly accurate classifier even of defective ones, but only if the classifiers are *independent*.

Ensembles (III)

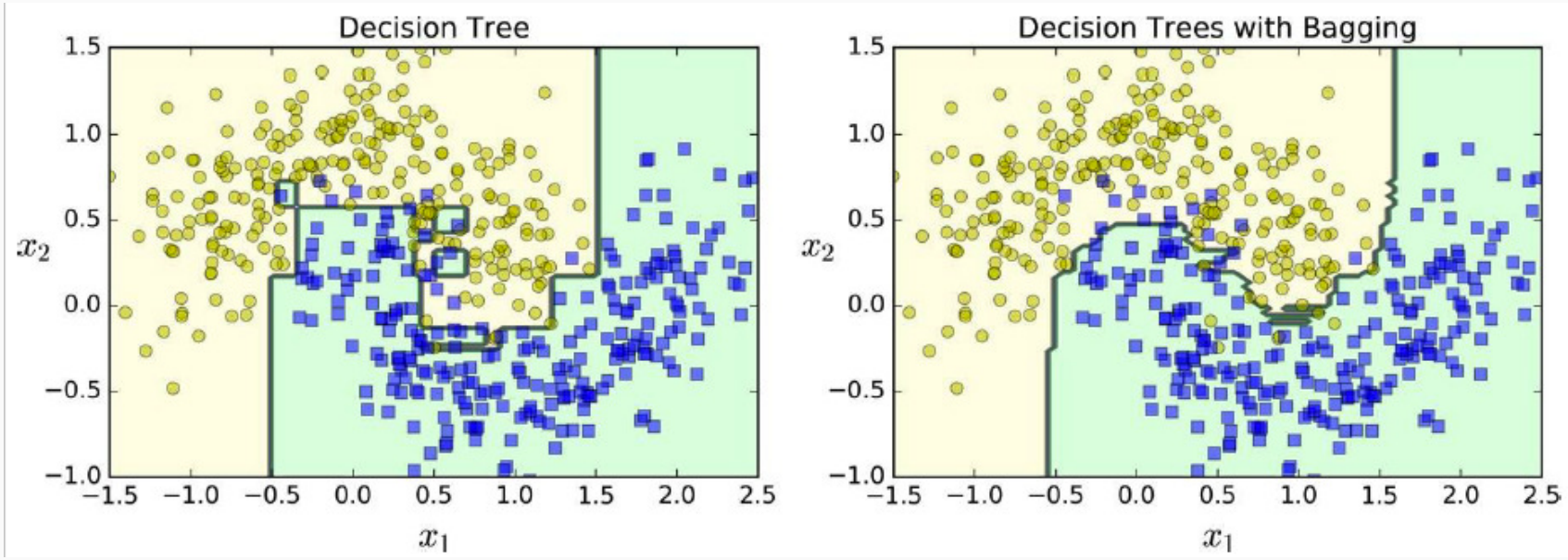
One way to get a diverse set of classifiers is to use very different training algorithms. Another approach is to use the same algorithm with *different training data* by sampling the training instances either with replacement (*bagging*) or without replacement (*pasting*).

Another alternative suitable for high-dimensional data sets is to sample the feature space.

Sampling both training instances and features are called the *random patches* method, while sampling only features, is called the *random subspace* method.

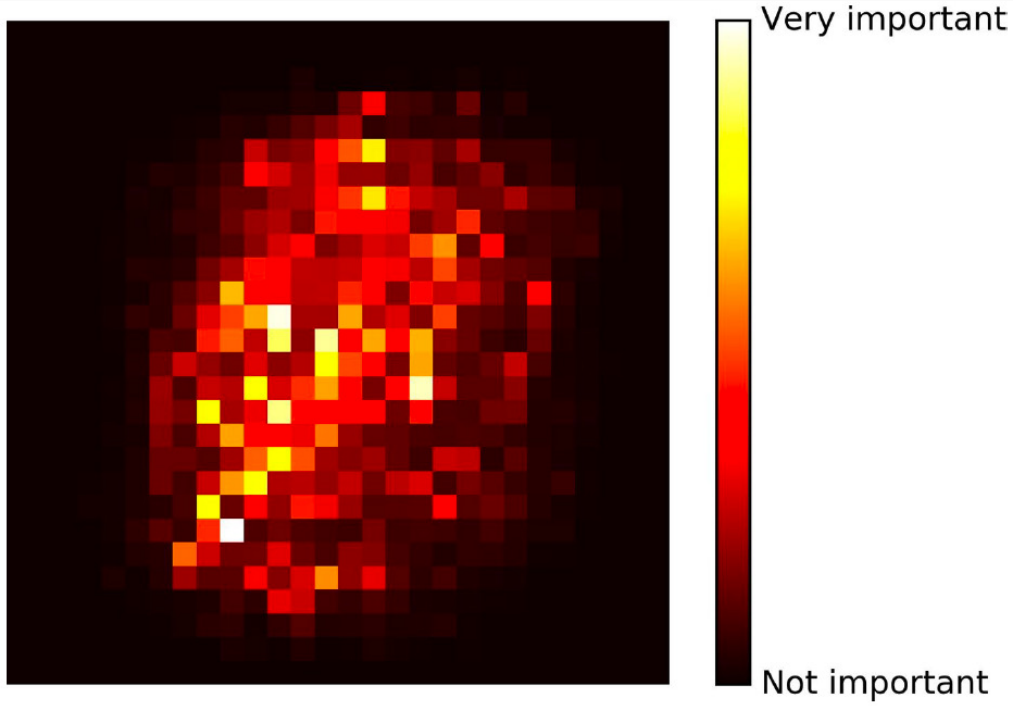
Random forests

A random forest is an ensemble formed by decision trees trained with the *bagging method*. In next example can be seen that the method is nonlinear and how the decision boundary is smoother than a single DT.



Random forests (II)

Random forests are among the most efficient ML methods, and also allows measuring the importance of each feature in the decision, by looking on how much a given feature is used in the nodes of the DT, e.g., for the MNIST data set:



Support vector machines (SVM)

Kernel methods

Given a fixed feature space transformation denoted by ϕ the function $k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})\phi(\boldsymbol{x}')$ is called a *kernel*. The simplest kernel is $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T\boldsymbol{x}'$ and is called the *linear kernel*.

Kernels are used to recast parametric models (e.g., the linear regression, the logistic regression, etc.) into non-parametric *dual* representations which require the evaluation of the kernel at the training data points (or a subset of them) to make predictions.

By using kernels, explicit transformations to and from the feature space are no longer needed, allowing high-dimensional and infinite-dimensional feature spaces.

Kernel methods (II)

To illustrate how the dual representations are obtained, consider the regularized loss function obtained for the linear regression model (see slide 25):

$$\widetilde{J}(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) - t_n)^2 + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w} \tag{22}$$

The optimal $\boldsymbol{w}$ obtained in eq. (11) by setting the gradient of the cost to 0 can be recast to:

$$\boldsymbol{w} = \frac{1}{\lambda}\sum_{n=1}^{N}(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) - t_n)\boldsymbol{\phi}(\boldsymbol{x}_n) = \boldsymbol{\Phi}^T\boldsymbol{a}$$

where $\boldsymbol{a}$ is the vector defined by the set $\{a_n = -\frac{1}{\lambda}(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) - t_n)\}$

Kernel methods (III)

The dual form for the loss is obtained by substituting $\boldsymbol{w}$ by $\boldsymbol{\Phi}^T\boldsymbol{a}$ in eq. (10):

$$\tilde{J}(\boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{a} - \boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^T\boldsymbol{t} + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{a} \quad (23)$$

Let $\boldsymbol{K}$ be the *Gram* matrix $\boldsymbol{\Phi}\boldsymbol{\Phi}^T$. That is, $(\boldsymbol{K})_{nm} = \boldsymbol{\phi}(\boldsymbol{x}_n)^T\boldsymbol{\phi}(\boldsymbol{x}_m)$ $= k(\boldsymbol{x}_n, \boldsymbol{x}_m)$.

Hence,

$$\tilde{J}(\boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{K}\boldsymbol{a} - \boldsymbol{a}^T\boldsymbol{K}\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^T\boldsymbol{t} + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{a} \quad (24)$$

Kernel methods (IV)

By setting the gradient of the loss with respect to $\boldsymbol{a}$ to zero we obtain $\boldsymbol{a}=(\boldsymbol{K}+\lambda \boldsymbol{I})^{-1}\boldsymbol{t}$.

In the dual representation predictions are computed as:

$$y(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{a}^T \boldsymbol{\Phi} \boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^T (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \boldsymbol{t} \qquad (25)$$

being $\boldsymbol{k}(\boldsymbol{x})$ the vector whose elements are $k(\boldsymbol{x}_n, \boldsymbol{x})$.

Note that eq. (25) doesn't require the use of parameters $\boldsymbol{w}$, but provides the prediction in terms of the kernel evaluations against each training point.

Kernel methods (V)

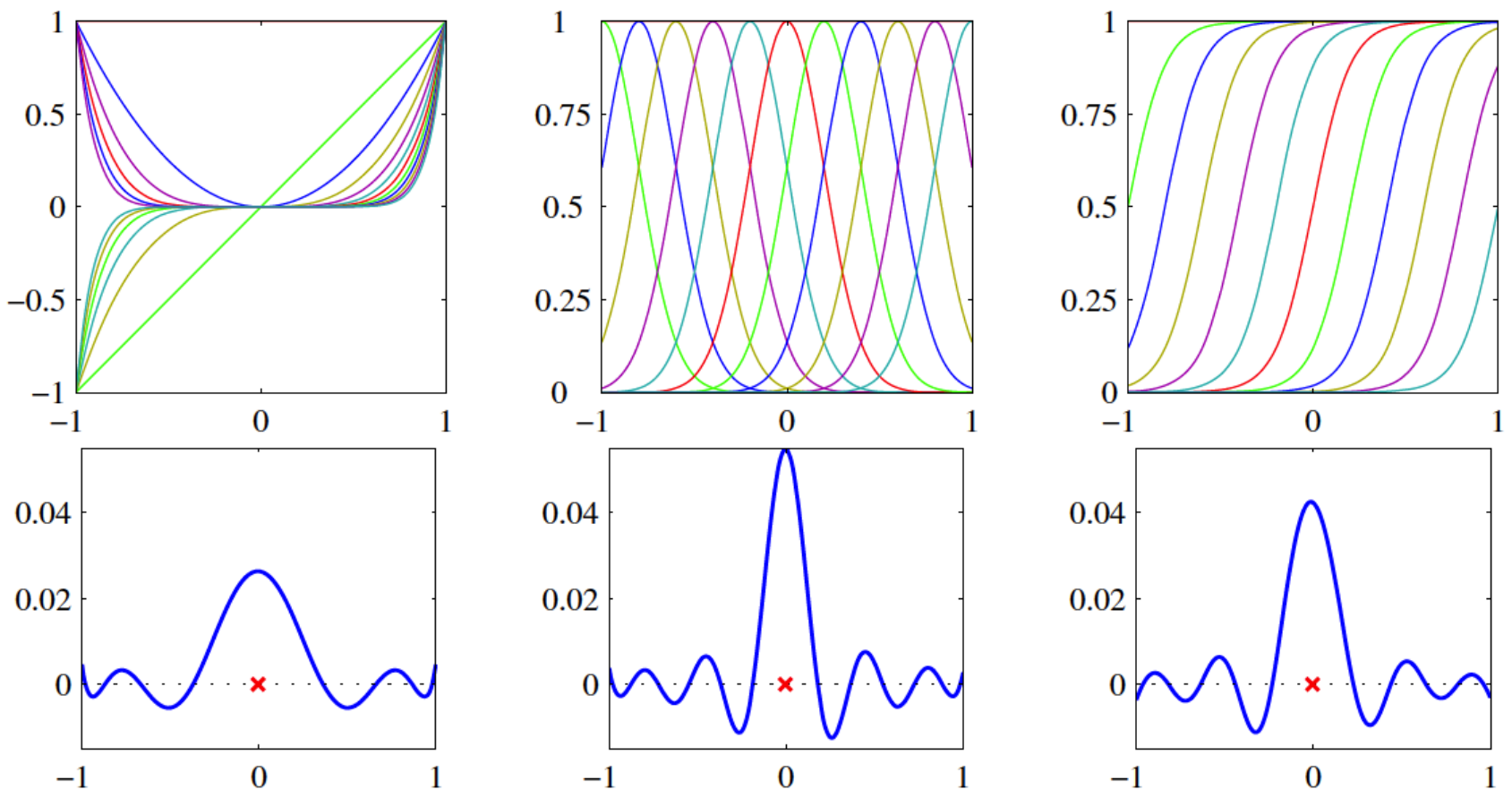
To avoid doing data conversion to/from the feature space the *kernel trick* is used. We illustrate it by means of an example. Let consider a 2-dimensional input space, and the function $k(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^T \boldsymbol{z})^2$. This function can be written[5] as:

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{z}) &= (\boldsymbol{x}\boldsymbol{z})^2 = (x_1 z_1 + x_2 z_2)^2 \\ &= x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\ &= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)(z_1^2, \sqrt{2} z_1 z_2, z_2^2)^T = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{z}) \end{aligned}$$

where the feature mapping is $\boldsymbol{\phi}(\boldsymbol{x}) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2)^T$. Let's remark that the kernel computations are done directly with the input data, without transformations to the feature space.

[5] For the sake of clarity, x_i denotes the i^{th} element of the $\boldsymbol{x}$ vector.

Kernel methods (VI)



The figure shows the kernel evaluations at $k(\boldsymbol{x}, \mathbf{0})$ associated to a Polynomials (left), Gaussians (center) and Sigmoids (right) basis functions.

Kernel methods (VII)

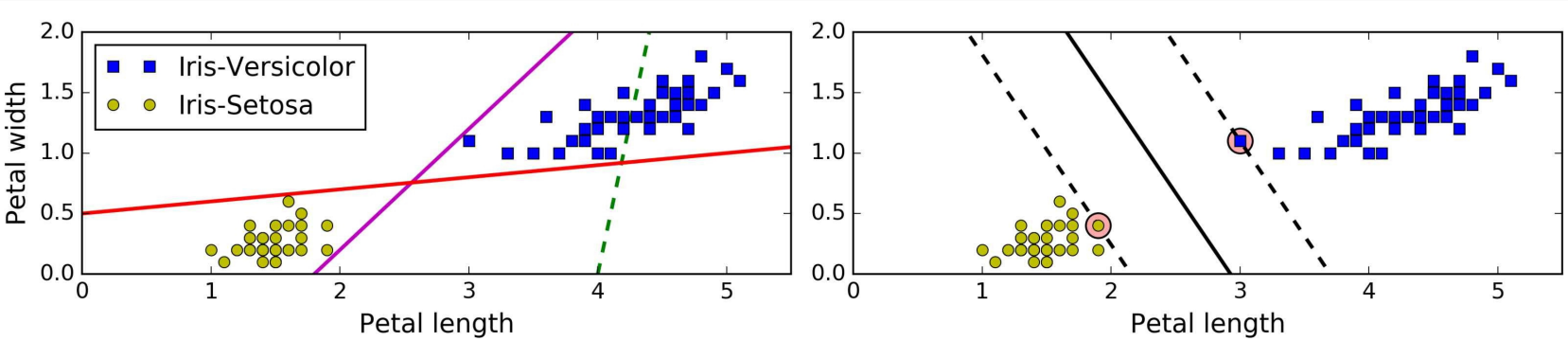
A necessary and sufficient condition for a function $k(\boldsymbol{x}, \boldsymbol{x}')$ to be a valid kernel is that its Gram matrix is positive semidefinite (all eigenvalues positive).

The dual representation for the linear regression requires evaluating the kernel on $\boldsymbol{x}$ for *all* the points in the data set, as well as the kernel of *all pairs* of points in the data set. Therefore, its use can be slow if the data set is large.

Some algorithms require evaluating the kernel only on a set of the training points. These are called *sparse kernel* methods.

Maximum margin classifiers

The next figure shows a data set with linear decision boundaries that produce a perfect separation in the data set. Intuitively, new instances will be better classified in the right figure since the decision boundary has an equal distance or *margin* to both classes.



This is the idea which the support vector machines (SVM) exploit: to construct *maximum* margin classifiers.

Maximum margin classifiers (II)

Let us assume that predictions for a binary classifier are based on the linear model:

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b \tag{26}$$

where b is an explicit bias term. Targets are $t_n \in \{-1, 1\}$ and new data points are classified according to the sign of $y(\boldsymbol{x})$ (note that $t_n y(\boldsymbol{x}) \geq 1$ for points of the training set correctly classified).

The perpendicular distance from a point $\boldsymbol{x}$ to the hyperplane $y(\boldsymbol{x}) = 0$ is given by $r = y(\boldsymbol{x})/\|\boldsymbol{w}\|$ (try to demonstrate it using the figure in slide 41 as reference). Thus, the distance from a point $\boldsymbol{x}_n$ to the boundary *surface* is:

$$\frac{y(\boldsymbol{x}_n)}{\|\boldsymbol{w}\|} = \frac{\boldsymbol{w}^T \boldsymbol{x}_n + b}{\|\boldsymbol{w}\|} \tag{27}$$

Maximum margin classifiers (III)

From all the infinitely possible solutions we aim at selecting those with all the training points correctly classified and with the maximum margin.

The minimum distance gives the margin form the hyperplane to the training points. However, a scale can be assumed such that $\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) + b = 1$ for the closest point (the one with minimal margin). Therefore the margin shall be $ge1$ for all points if they are correctly classified.

Then, achieving the maximum margin requires to maximize $1/\|w\|$ or, conversely, minimizing $\|w\|^2$.

Maximum margin classifiers (IV)

Hence, the optimization problem can be stated in its *primal* form:

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 \tag{28}$$

subject to $t_n[\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1$ (note the factor t_n to force correct classifications).

These constraints can be added to the function to minimize through the corresponding Lagrange multipliers$\{a_n\}$,

$$L(\boldsymbol{w}, b, \boldsymbol{a}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{n=1}^{N} a_n[t_n(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) + b - 1] \tag{29}$$

Maximum margin classifiers (V)

In this new form, by setting the gradients with respect to $\boldsymbol{w}$, b, and $\boldsymbol{a}$ to 0, the next conditions are obtained:

$$\boldsymbol{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\boldsymbol{x}_n) \tag{30}$$

$$0 = \sum_{n=1}^{N} a_n t_n \tag{31}$$

By substituting both in eq. (29), the dual representation is obtained:

$$\max_{\boldsymbol{a}} \widetilde{L}(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m) \tag{32}$$

subject to $a_n \geq 0$, for $n = 1, \ldots, N$, and $\sum_{n=1}^{N} a_n t_n = 0$.

Maximum margin classifiers (VI)

Both the primal and dual forms are quadratic programming problems, thus convex, and since the region determined by the constraints is convex, there is a single optimal.

Solving quadratic problems has order cubic order on the number of variables. Thus, the primal is $O(M^3)$, and the dual is $O(N^3)$. Therefore the dual is advantageous for high-dimensional feature spaces. Besides, the kernel trick allows the direct computation of kernels in the input space.

Maximum margin classifiers (VII)

Besides, the dual form satisfies the KKT conditions, which establish when a point in the variable space is a saddle point (a necessary condition to be an optimum). In the dual form it translates to the conditions:

$$\begin{aligned} a_n &\geq 0 \\ t_n y(\boldsymbol{x}_n) - 1 &\geq 0 \\ a_n(t_n y(\boldsymbol{x}_n) - 1) &= 0 \end{aligned}$$

So, a point in the training set either lies in the boundaries established by the margin ($t_n y(\boldsymbol{x}_n) - 1) = 0$), or its beyond it and its Lagrange multiplier is deactivated ($a_n = 0$).

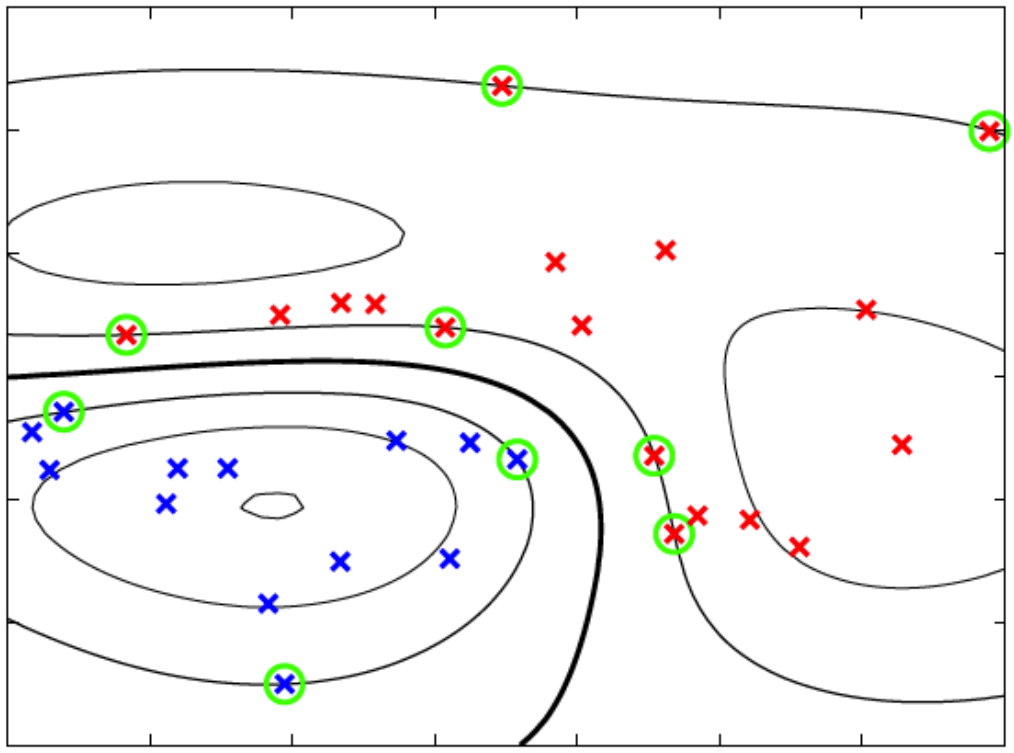
Maximum margin classifiers (VII)

Points in the margin boundary are called *support vectors* (hence the method name). Note that for the rest of the points, since $a_n = 0$, the kernel computation in eq. (32) is no longer needed, leading to a sparse model, where only the support vectors have to be used.

This observation also permits efficient heuristics to speed up the solving of the dual quadratic program. Its solution provides $\boldsymbol{a}$, which in turn allows computing $\boldsymbol{w}$ with eq. (30). Then, the bias b is calculated by selecting some support vector, which has to fulfill $\boldsymbol{w}^T\boldsymbol{x} + b = 1$. Predictions for new data instances are taken evaluating the sign of

$$y(\boldsymbol{x}) = \sum_{x_n \text{ is SV}} a_n t_n k(\boldsymbol{x}, \boldsymbol{x_n}) + b$$

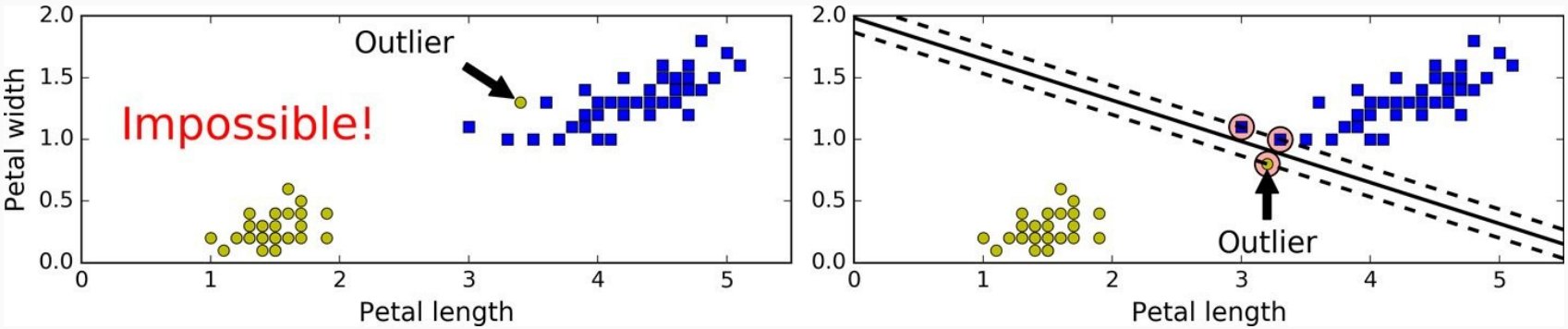
Maximum margin classifiers (VIII)



SVM boundary computed using the Gaussian kernel, $k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\|\boldsymbol{x} - \boldsymbol{x}'\|^2/2\sigma^2\right)$

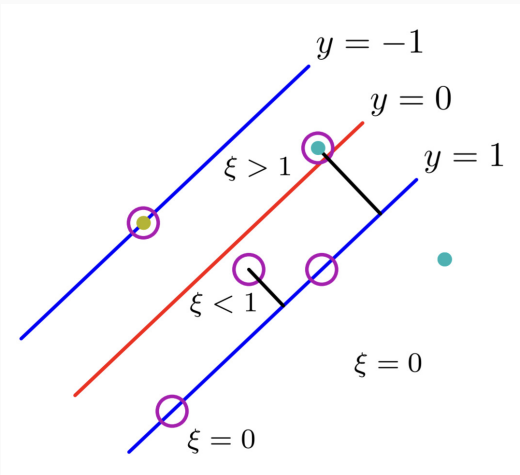
Soft margin classifiers

The SVM can perform very well if the training set is linearly separable. But, in case it isn't, or in the presence of *outliers*, its performance degrades quickly, as shown in the figure below.

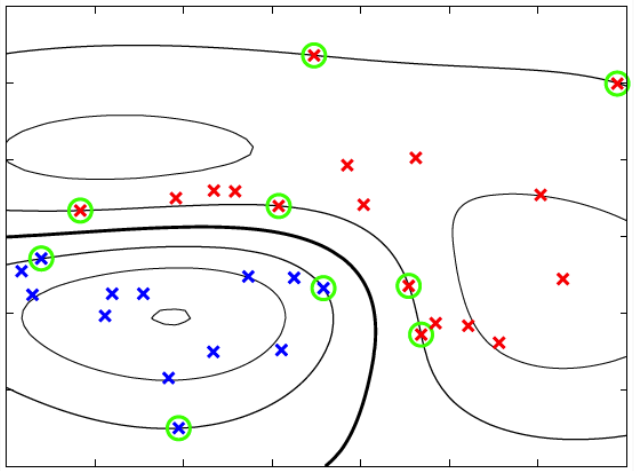


Soft margin classifiers (II)

The basic idea to correct this problem is using a *penalty* function, which allows some points to lie in the margin, or even being in the wrong side of the surface. A typical set for this penalty is shown below (left). As a result, slightly modified versions of the primal form are obtained, which can be solved by applying similar ideas.



Penalties are introduced in the primal form as *slack* variables ε_n.



SVM boundary for a training set which is non linearly separable with Gaussian kernel.

Further reading

Further reading

Artificial neural networks (ANN) are nonlinear methods composed by layers, each formed up by a group of "neurons", which are functions of the forms $f(\boldsymbol{w}^T\boldsymbol{x})$, where $f(\cdot)$ is some activation function. ANN is used both in classification and regression and constitute the base for modern Deep learning models. They have the property that, despite characterized by a non-convex loss, an efficient method to train them exists: the back-propagation, resulting in high-quality predictions. Besides, it has been shown that they can "learn" functions of arbitrary complexity.

A mathematical treatment for ANN is provided in Bishop ch. 5. Besides, in Geron Part II (ch.9-14), a clear and concise description is provided (in particular, see Geron ch. 10 and 13).

Further reading (II)

Bayesian learning has been studied partly for the case of linear regression. For other models, like logistic regression, it is intractable, though Gaussian approximations are possible (see Bishop 4.5). Other models based on this approach are generative Bayesian networks with latent variables (see Bishop ch. 8).

Classifiers can also support multi-labels (assign more than one label in the output, e.g., a picture can be labeled as both containing a "Dog" and a "Cat"). In the same way, regression models may support multi-output (targets of dimensionality greater than 1). See multi-label link and multi-output link, respectively.

Further reading (III)

Decision trees may also be applied to regression (see Geron ch. 6). Besides, other techniques based on ensembles like boosting or stacking are discussed at the end of Geron ch. 7.

SVM can also be used in unsupervised and regression contexts (see Bishop 7.1.3 and 7.1.4). Besides, SVM lack the ability to provide the classification probability, and generalizations to support multi-class are cumbersome. These aspects can be addressed with *relevance vector machines*, which are based on a Bayesian approach (see Bishop 7.2).

Iterative solving based on the gradient descent like mini-batch gradient descent are discussed in Geron ch. 4. Purely sequential algorithms, like least mean squares (LMS) are treated in Bishop 3.1.3, and more generally in Bishop ch. 13.

Further reading (IV)

Finally, it is worth to have a look to the decision boundaries of some common methods implemented in the scikit-learn python library. As can be seen, some methods are able to predict classes probabilities, while others don't.