Kernels and support vector machines

2DV516/2DT916

Jonas Nordqvist

jonas.nordqvist@lnu.se

May 7, 2024

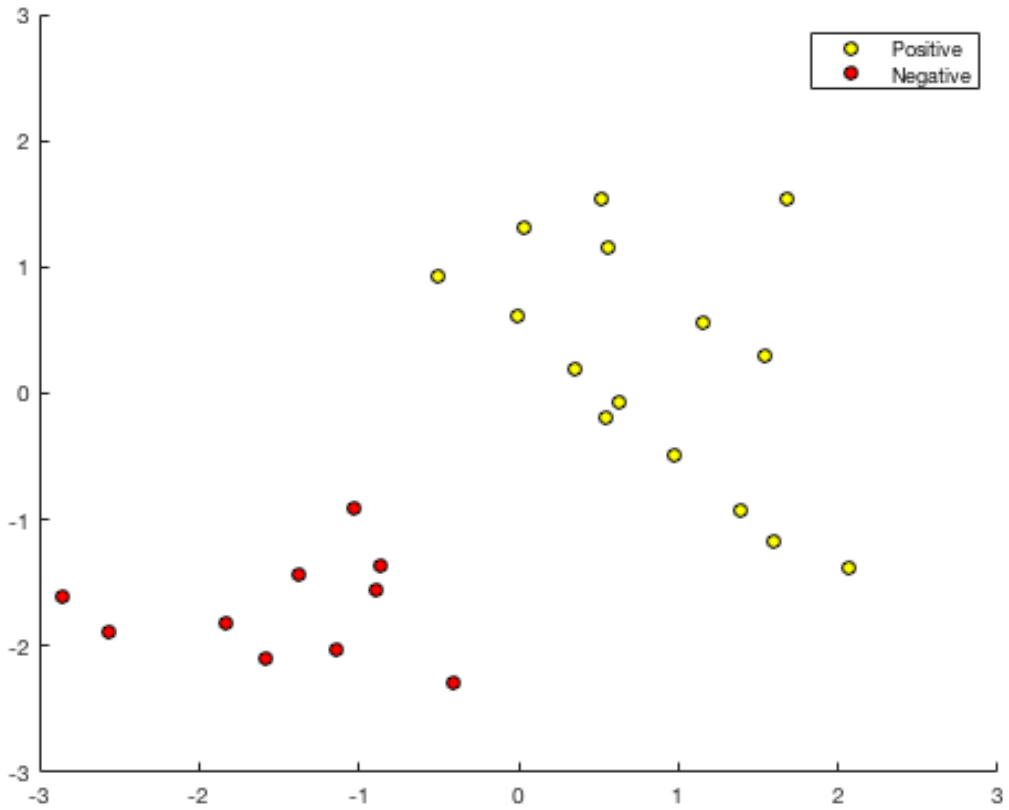
Agenda

- Maximal margin classifier
- Support vector classifier
- Kernels
- Support vector machines
- Kernel Ridge Regression

Reading Instructions
Chapter 8

Example: an ill-defined problem

Find the separating hyperplane



But first... what is a hyperplane?

Hyperplanes

Hyperplanes are a geometrical structure, in particular a subspace of dimension one less than its ambient space, *i.e.* the space in which the hyperplane is defined.

Hyperplanes

Hyperplanes are a geometrical structure, in particular a subspace of dimension one less than its ambient space, *i.e.* the space in which the hyperplane is defined.

If the ambient space is of dimension:

- $1 \implies$ a hyperplane is a dot.
- $2 \implies$ a hyperplane is a line.
- $3 \implies$ a hyperplane is a plane.
- $4 \implies$ a hyperplane is a 3-dimensional plane
- $p \implies$ the hyperplane is a $(p-1)$-dimensional plane

Hyperplanes

Hyperplanes are a geometrical structure, in particular a subspace of dimension one less than its ambient space, *i.e.* the space in which the hyperplane is defined.

If the ambient space is of dimension:

- $1 \implies$ a hyperplane is a dot.
- $2 \implies$ a hyperplane is a line.
- $3 \implies$ a hyperplane is a plane.
- $4 \implies$ a hyperplane is a 3-dimensional plane
- $p \implies$ the hyperplane is a $(p-1)$-dimensional plane

The hyperplane always parts the space in two disjoint parts, for example above or below a line.

Given p points in the ambient space a hyperplane through these points are unique up to a constant, e.g. $2Y + 6X + 4 = 0$ and $Y + 3X + 2 = 0$.

Hyperplanes

A hyperplane in p-dimensional ambient space can be described by an equation of the form

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0. \tag{1}$$

Any point x which lies in the hyperplane satisfies (1).

There are essentially three scenarios for a point $x' = (x'_1, x'_2, \ldots, x'_p)$, either

- x' lies in the hyperplane and

$$\beta_0 + \beta_1 x'_1 + \cdots + \beta_p x'_p = 0,$$

- x' lies 'above' the hyperplane and

$$\beta_0 + \beta_1 x'_1 + \cdots + \beta_p x'_p > 0,$$

- or x' lies 'below' the hyperplane and

$$\beta_0 + \beta_1 x'_1 + \cdots + \beta_p x'_p < 0.$$

Geometric motivation

Let Π be a hyperplane in a p-dimensional space, and put $\beta := (\beta_1, \ldots, \beta_p)^\intercal$. For simplicity we assume $\beta_0 := 0$, then the equation of Π is given by

$$\Pi : \beta_1 X_1 + \cdots + \beta_p X_p = 0.$$

Geometric motivation

Let Π be a hyperplane in a p-dimensional space, and put $\beta := (\beta_1, \ldots, \beta_p)^\intercal$. For simplicity we assume $\beta_0 := 0$, then the equation of Π is given by

$$\Pi : \beta_1 X_1 + \cdots + \beta_p X_p = 0.$$

Assume that $x^* = (x_1^*, \ldots, x_p^*)$ lies 'above' Π then we will show that

$$\beta_0 + \beta_1 x_1 * + \cdots + \beta_p x_p * > 0.$$

Recall that since $\beta_0 = 0$ we have $\beta_0 + \beta_1 x_1 * + \cdots + \beta_p x_p * = (x^*)^T \beta$.

Geometric motivation

Let Π be a hyperplane in a p-dimensional space, and put $\beta := (\beta_1, \ldots, \beta_p)^\intercal$. For simplicity we assume $\beta_0 := 0$, then the equation of Π is given by

$$\Pi : \beta_1 X_1 + \cdots + \beta_p X_p = 0.$$

Assume that $x^* = (x_1^*, \ldots, x_p^*)$ lies 'above' Π then we will show that

$$\beta_0 + \beta_1 x_1 * + \cdots + \beta_p x_p * > 0.$$

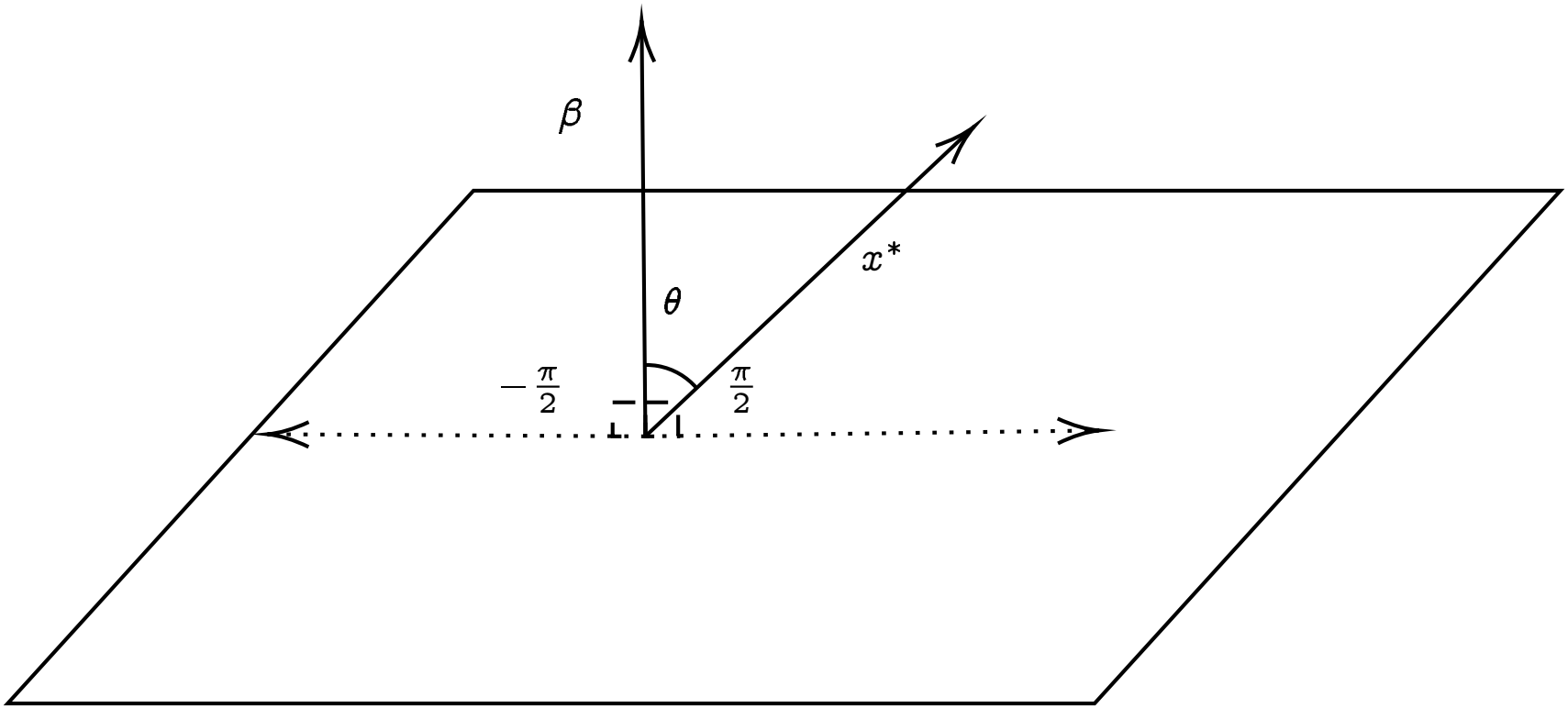
Recall that since $\beta_0 = 0$ we have $\beta_0 + \beta_1 x_1 * + \cdots + \beta_p x_p * = (x^*)^T \beta$.

By the definition of the scalar product we have

$$(x^*)^T \beta = ||x^*|| \cdot ||\beta|| \cos(\theta),$$

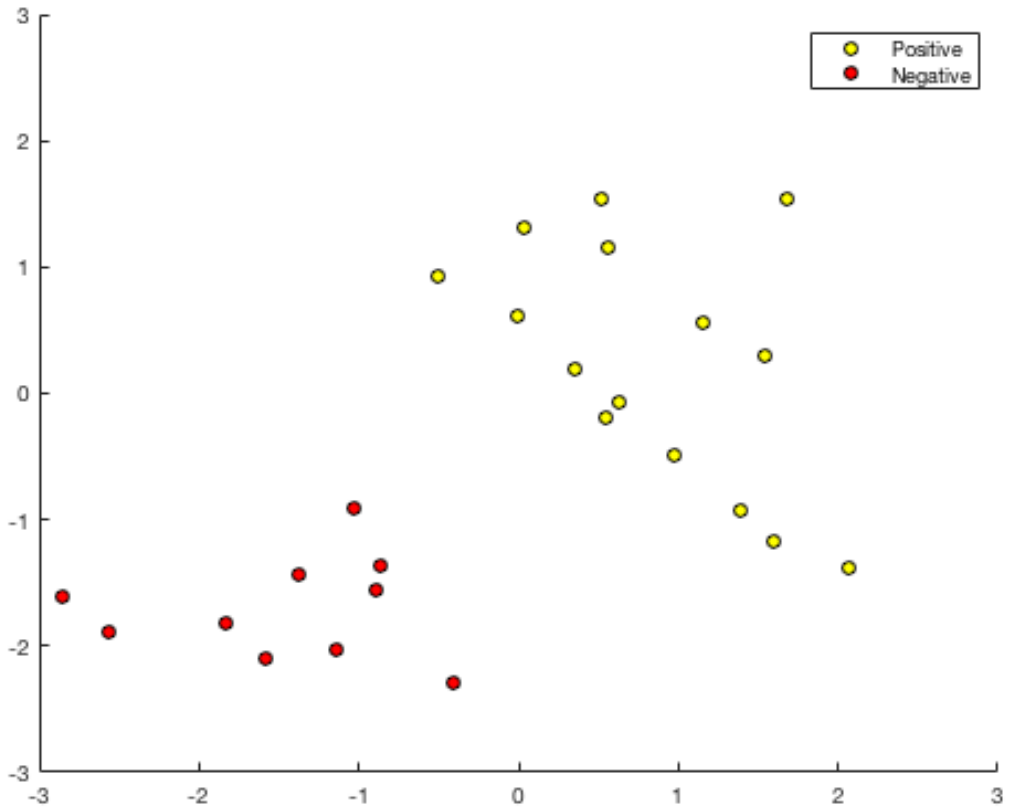
where θ is the angle between x and β. The sign of this quantity is determined completely by $\cos(\theta)$. Thus, $(x^*)^T \beta > 0$ if and only if $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$.

Geometric motivation



Back to the problem

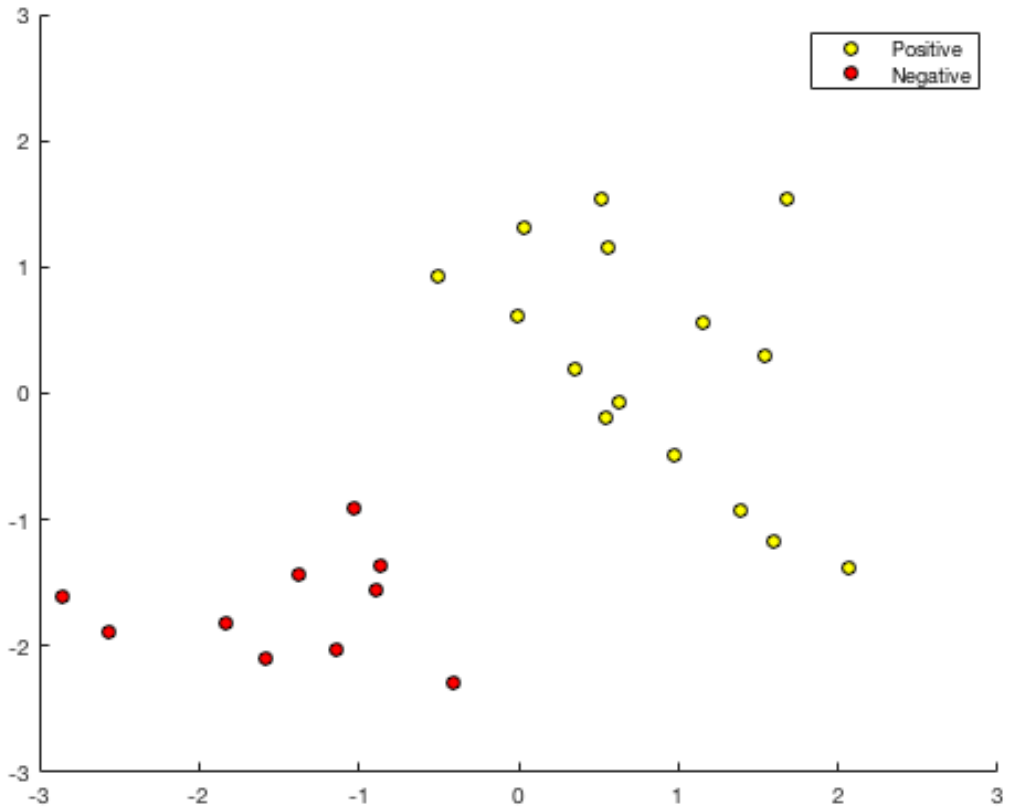
We want to find a hyperplane that separates our data.



Is this always possible given any data?

Back to the problem

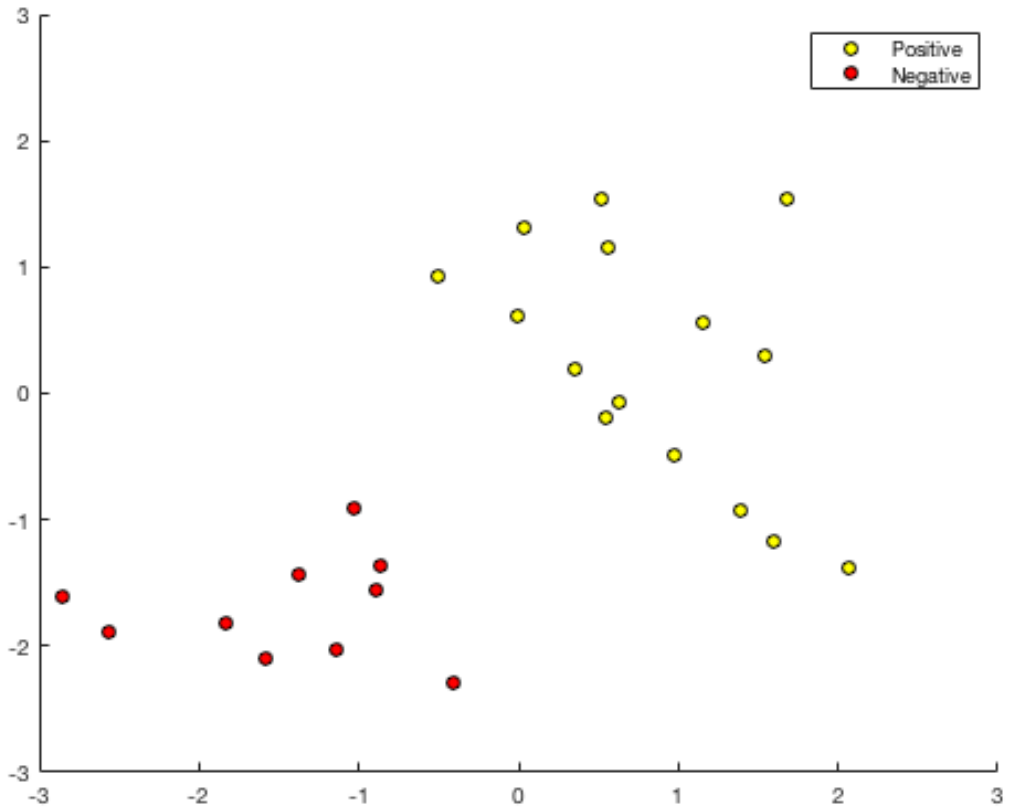
We want to find a hyperplane that separates our data.



Is this always possible given any data? No!

Back to the problem

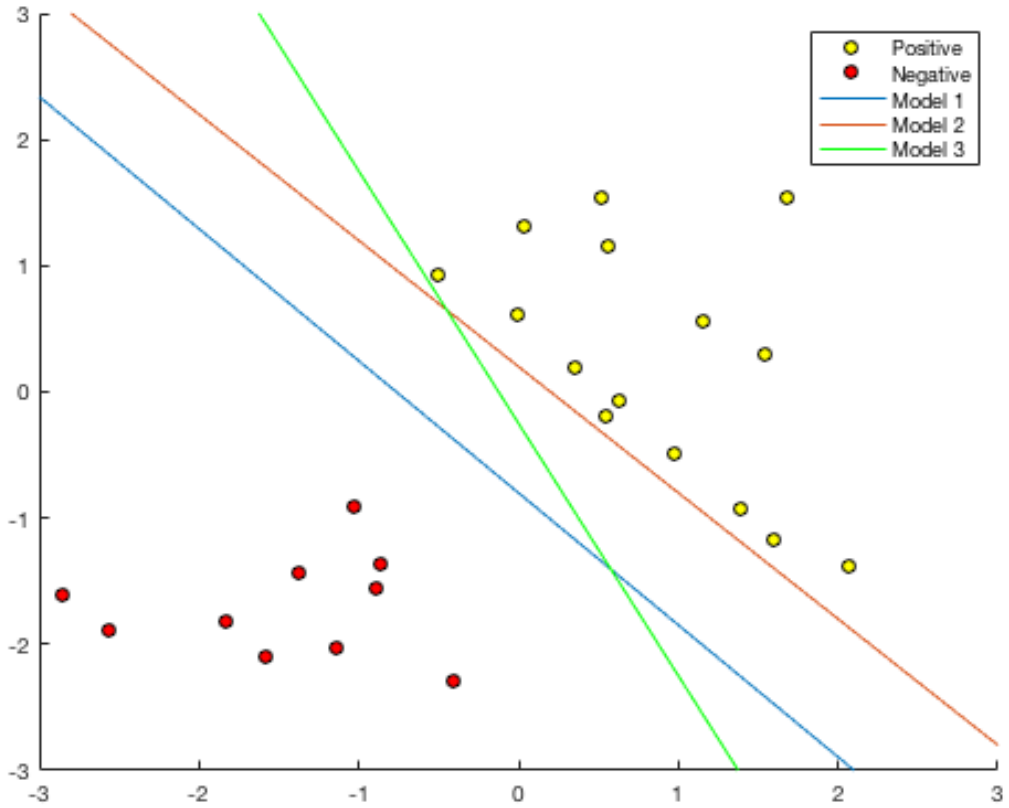
We want to find a hyperplane that separates our data.



Is this always possible given any data? No! Why is the problem ill-defined?

Back to the problem

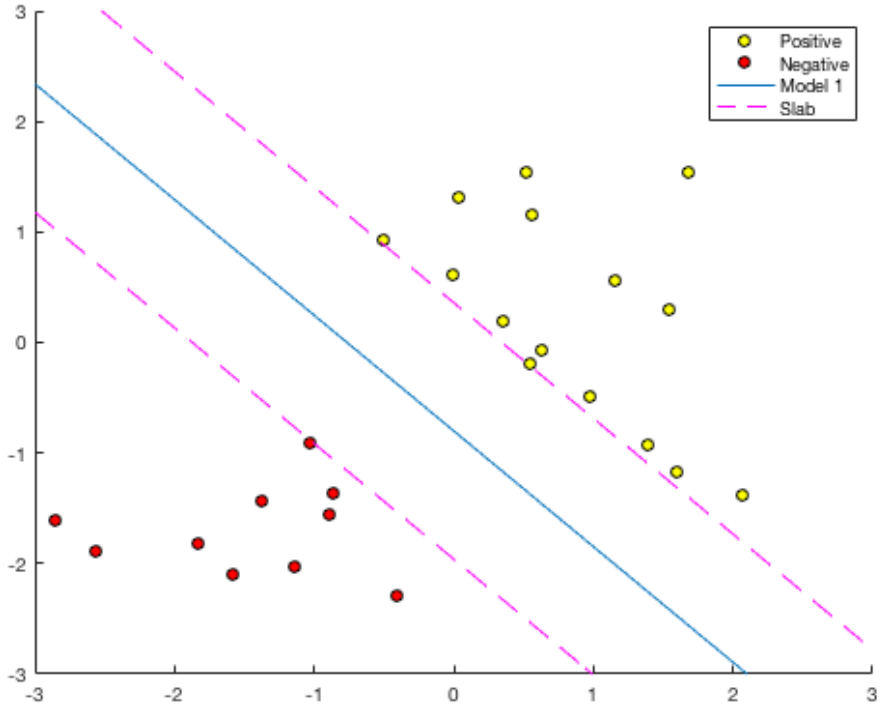
We want to find a hyperplane that separates our data.



Is this always possible given any data? No! Why is the problem ill-defined? No unique solution!

Maximal margin classifier

We want to choose the hyperplane which separates the data, and which has the largest margin (or cushion or slab) separating the two classes.



Note that there are only three points contributing to the computation of the slab, the ones on the margin. These points are called *support vectors*.

Formulating the (hard) problem

Remark

We will consider the binary classification case. For this problem it is convenient to use 1 *for positive and* -1 *for negatives labels in* y *as this implies*

$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq 0,$$

for all $1 \leq i \leq n$.

Denote by M the distance from the hyperplane to the two classes.[1]
The main objective is the following

$$\max_{\beta_0,\beta_1,\ldots,\beta_p} M \tag{2}$$

$$\text{subject to} \sum_{i=1}^{p} \beta_i^2 = ||\beta|| = 1 \tag{3}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq M. \tag{4}$$

[1] By distance to a class we mean the shortest distance from any point in the class to the hyperplane.

Distance formula

Lemma

The distance between the point x_i and the hyperplane is given by

$$\frac{1}{||\beta||} y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}).$$

Distance formula

Lemma

The distance between the point x_i and the hyperplane is given by

$$\frac{1}{||\beta||} y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}).$$

A hyperplane with the equation

$$\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p = 0, \tag{5}$$

has normal $\beta = (\beta_1, \ldots, \beta_p)^\intercal$.

Hence, the shortest path from a point x_i to the hyperplane goes along the line described by β and the point x_i. So, the equation of the line is given by $x_i + t\beta$, $t \in \mathbb{R}$.

The line satisfies (5) in exactly one point

$$\beta_0 + \beta_1(x_{i1} + t\beta_1) + \cdots + \beta_p(x_{ip} + t\beta_p) = 0$$

Distance formula

Solving for t yields

$$t = -\frac{\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}}{\beta_1^2 + \cdots + \beta_p^2} = -\frac{1}{||\beta||^2}(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}).$$

Denote by x^* the point in the hyperplane which is the intersection between the hyperplane and the line. Then the smallest distance between x_i and x^* and thus the plane is given by

$$||x^* - x_i|| = ||t\beta|| = |t|||\beta||,$$

and we obtain

$$|t|||\beta|| = \frac{1}{||\beta||}|\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}| = \frac{1}{||\beta||} y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}).$$

Reformulating the problem

Denote by $\beta := (\beta_1, \ldots, \beta_p)$, and

$$||\beta|| = \sqrt{\beta_1^2 + \cdots + \beta_p^2}.$$

The distance from the hyperplane to any point x_i is given by $y_i(x_i^\mathsf{T}\beta + \beta_0)$, by (3), and in particular if $||\beta||$ is no longer necessarily equal to 1 we have

$$\frac{1}{||\beta||} y_i(x_i^\mathsf{T}\beta + \beta_0) \geq M \iff y_i(x_i^\mathsf{T}\beta + \beta_0) \geq M||\beta||.$$

Put $M = 1/||\beta||$, and hence maximizing the margin M implies minimizing $||\beta||$. This is further equivalent to $\min ||\beta||^2$. So, our problem can instead be formulated as

$$\min \frac{1}{2}||\beta||^2, \text{ subject to } y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq 1,$$

or equivalently

$$\min \frac{1}{2}\beta^\mathsf{T}\beta, \text{ subject to } y_i(\beta_0 + \beta^\mathsf{T} x_i) \geq 1,$$

Reformulating the problem

Denote by $\beta := (\beta_1, \ldots, \beta_p)$, and

$$||\beta|| = \sqrt{\beta_1^2 + \cdots + \beta_p^2}.$$

The distance from the hyperplane to any point x_i is given by $y_i(x_i^\mathsf{T}\beta + \beta_0)$, by (3), and in particular if $||\beta||$ is no longer necessarily equal to 1 we have

$$\frac{1}{||\beta||} y_i(x_i^\mathsf{T}\beta + \beta_0) \geq M \iff y_i(x_i^\mathsf{T}\beta + \beta_0) \geq M||\beta||.$$

Put $M = 1/||\beta||$, and hence maximizing the margin M implies minimizing $||\beta||$. This is further equivalent to $\min ||\beta||^2$. So, our problem can instead be formulated as

$$\min \frac{1}{2}||\beta||^2, \text{ subject to } y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq 1,$$

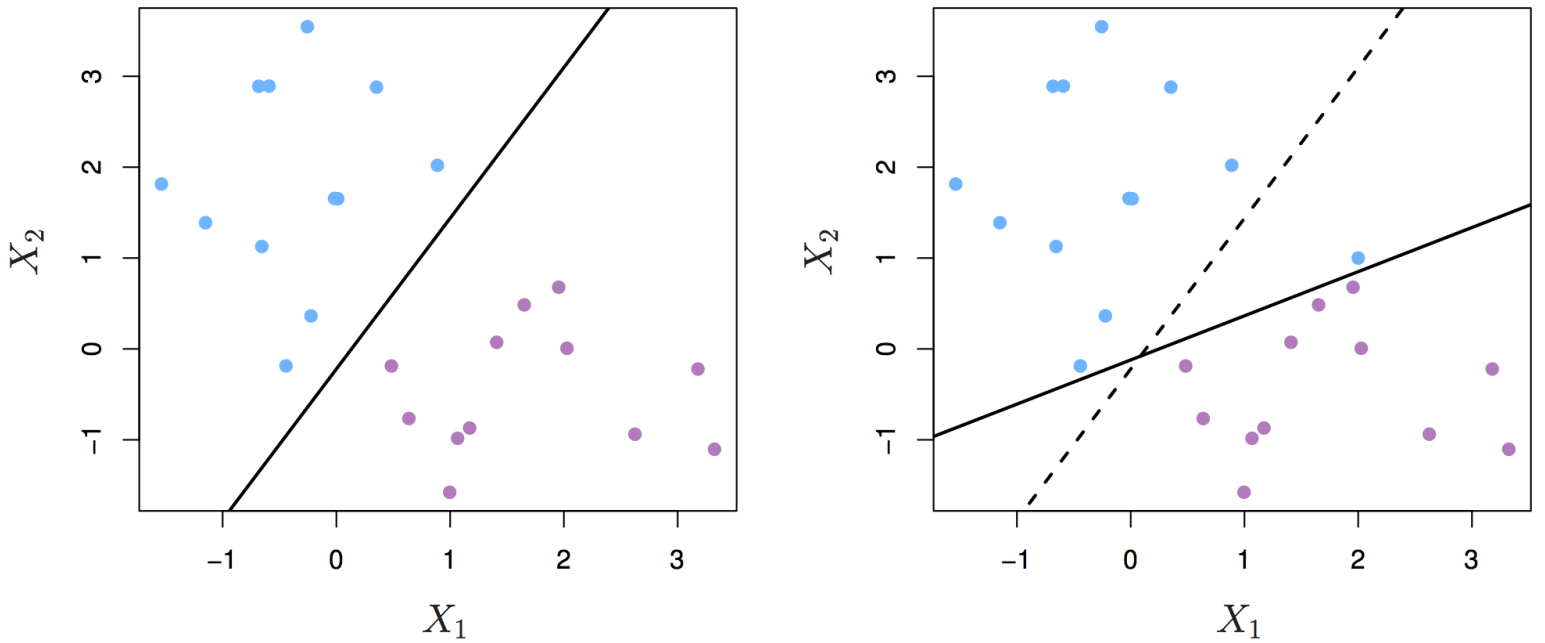
or equivalently

$$\min \frac{1}{2}\beta^\mathsf{T}\beta, \text{ subject to } y_i(\beta_0 + \beta^\mathsf{T} x_i) \geq 1,$$

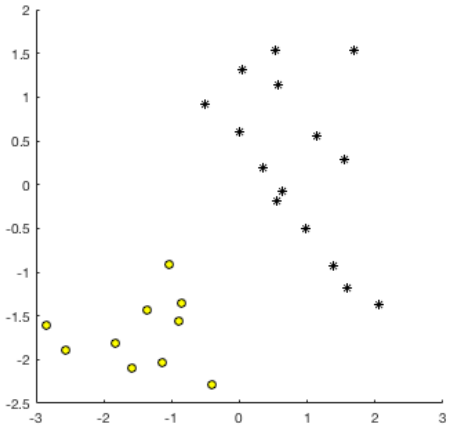
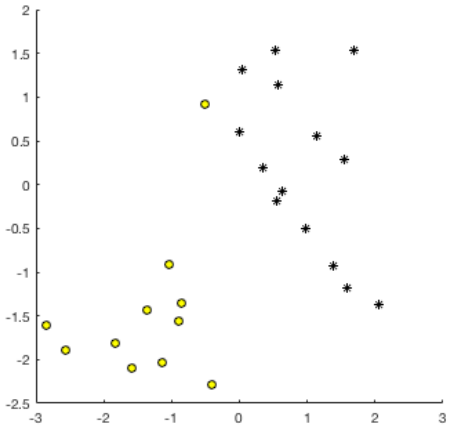
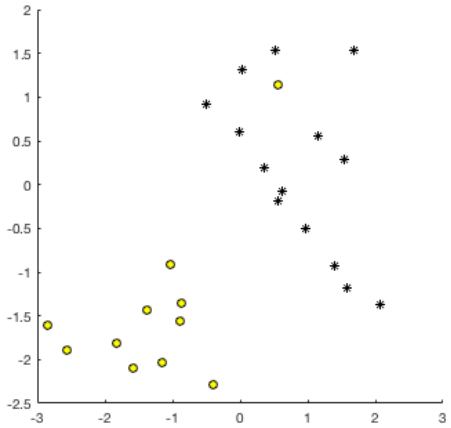
Note that the points which give equality in the above constraint are the support vectors.

Maximal margin classifier is non-robust

The maximal margin classifier has good performance on very special problems, but it is very non-robust. Here this means: minor changes in the input data, may yield major changes in the decision boundary.

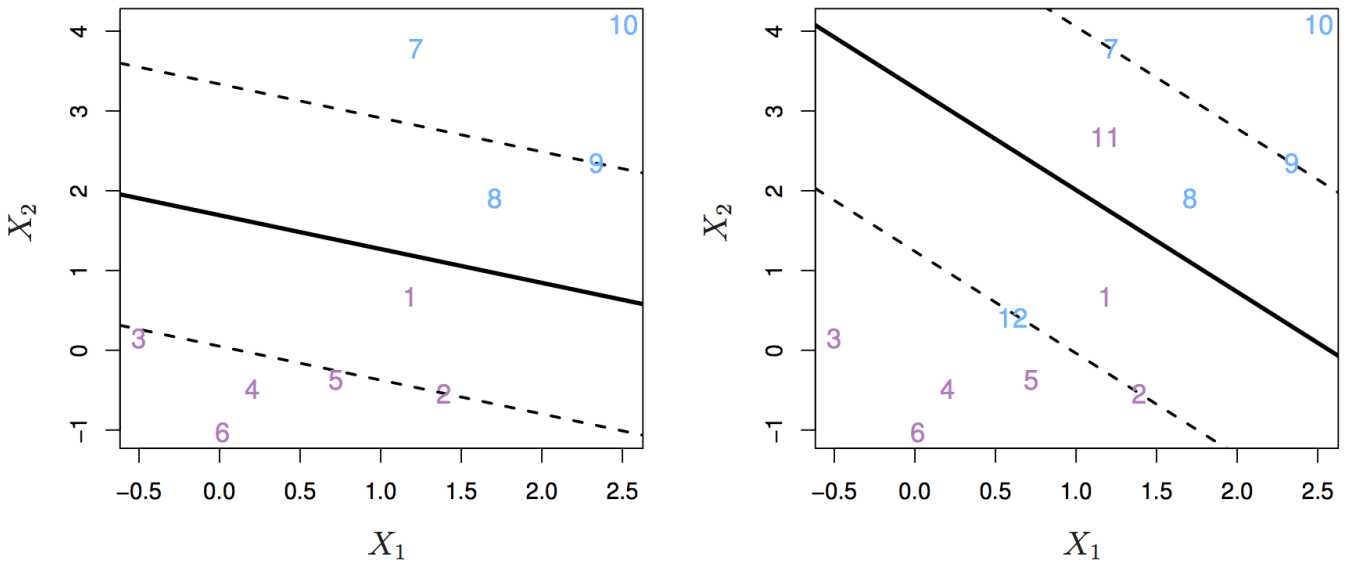


Decision boundary examples

Soften the margin

A natural extension of the maximal margin classifier is the support vector classifier, which allows for some violation of the margin, but in most cases instances are on the correct side of the margin.



Formulating the support vector classifier

To soften the margin, we need to introduce a slack variables $\varepsilon_i \geq 0$ for each instance.

Formulating the support vector classifier

To soften the margin, we need to introduce a slack variables $\varepsilon_i \geq 0$ for each instance.

The slack ε_i measures how much x_i may violate the margin. Our two objectives are now

Formulating the support vector classifier

To soften the margin, we need to introduce a slack variables $\varepsilon_i \geq 0$ for each instance.

The slack ε_i measures how much x_i may violate the margin. Our two objectives are now

- make the slack variables as small as possible

Formulating the support vector classifier

To soften the margin, we need to introduce a slack variables $\varepsilon_i \geq 0$ for each instance.

The slack ε_i measures how much x_i may violate the margin. Our two objectives are now

- make the slack variables as small as possible
- minimize $||\beta||$

$$\min_{\beta_0,\beta,\varepsilon} \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{n} \varepsilon_i$$

$$\text{subject to } y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq (1 - \varepsilon_i) \text{ and } \varepsilon_i \geq 0$$

Formulating the support vector classifier

To soften the margin, we need to introduce a slack variables $\varepsilon_i \geq 0$ for each instance.

The slack ε_i measures how much x_i may violate the margin. Our two objectives are now

- make the slack variables as small as possible
- minimize $||\beta||$

$$\min_{\beta_0,\beta,\varepsilon} \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{n} \varepsilon_i$$

$$\text{subject to } y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq (1 - \varepsilon_i) \text{ and } \varepsilon_i \geq 0$$

Note that if

- $\varepsilon_i = 0$, x_i is on the correct side of the margin
- $\varepsilon_i > 0$, x_i has violated the margin
- $\varepsilon_i > 1$, x_i is on the wrong side of the hyperplane

Formulating the support vector classifier

The parameter C can be seen as a regularization parameter.

Formulating the support vector classifier

The parameter C can be seen as a regularization parameter.

$$\min_{\beta_0,\beta,\varepsilon} \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{n}\varepsilon_i$$

$$\text{subject to } y_i(\beta_0 + x_i^\intercal\beta) \geq (1-\varepsilon_i) \text{ and } \varepsilon_i \geq 0$$

C will serve as a tradeoff between

- making the margin large
- making sure that most examples have margin at least $1/||\beta||$

Formulating the support vector classifier

The parameter C can be seen as a regularization parameter.

$$\min_{\beta_0,\beta,\varepsilon} \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{n}\varepsilon_i$$

$$\text{subject to } y_i(\beta_0 + x_i^\mathsf{T}\beta) \geq (1-\varepsilon_i) \text{ and } \varepsilon_i \geq 0$$

C will serve as a tradeoff between

- making the margin large
- making sure that most examples have margin at least $1/||\beta||$

If $y_i(\beta_0 + x_i^\mathsf{T}\beta) \geq 1$ then there is no cost of misclassification, but if not, then we have to 'pay' the price $\varepsilon_i = 1 - y_i(\beta_0 + x_i^\mathsf{T}\beta)$. Hence, the 'price' is given by

$$\varepsilon_i = \max\{0, 1 - y_i(\beta_0 + x_i^\mathsf{T}\beta)\},$$

and replacing ε_i by this expression in our minimization scheme yields

$$\min_{\beta_0,\beta} \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{n}\max\{0, 1 - y_i(\beta_0 + x_i^\mathsf{T}\beta)\}.$$

This perhaps looks like a familiar setting for ML and we'll revisit this in a couple of slides.

Formal explanation

We may consider the Lagrangian relaxation of the problem and obtain the following Lagrangian primal function

$$L_P = \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{n}\varepsilon_i + \sum_{i=1}^{n}\alpha_i((1-\varepsilon_i) - y_i(\beta_0 + x_i^\intercal\beta)) - \sum_{i=1}^{n}\mu_i\varepsilon_i.$$

This is primal Lagrangian L_P. Setting derivative w.r.t. β, β_0 and ε to zero yields

$$\frac{\partial L_P}{\partial \beta} = 0 \iff \beta = \sum_{i=1}^{n}\alpha_i y_i x_i$$

$$\frac{\partial L_P}{\partial \beta_0} = 0 \iff 0 = \sum_{i=1}^{n}\alpha_i y_i$$

$$\frac{\partial L_P}{\partial \varepsilon} = 0 \iff \alpha_i = C - \mu_i, \quad \forall i$$

Substituting back in L_P we obtain the dual L_D which we want to maximize. We get

$$L_D = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^\intercal x_j.$$

Support vector classifier

Using the L_P-formulation and the partials of the previous slide yields the dual formulation L_D

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^\intercal x_j.$$

which is supposed to be maximized subject to the constraints $\alpha_i \in [0, C]$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$.

Support vector classifier

Using the L_P-formulation and the partials of the previous slide yields the dual formulation L_D

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^\mathsf{T} x_j.$$

which is supposed to be maximized subject to the constraints $\alpha_i \in [0, C]$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$.

Our final take on this is that the support vector classifier can be written as

$$\hat{\beta} = \sum_{i=1}^{n} \alpha_i y_i x_i, \qquad \text{implying } f(x) = \beta_0 + x^\mathsf{T} \beta = \beta_0 + \sum_{i=1}^{n} \alpha_i y_i x^\mathsf{T} x_i.$$

Some of the α_i are zero and for the ones it is not, the corresponding x_i is a support vector. Hence, classification done by computing the dot product between the test point x and all the support vectors x_i.

Support vector classifier

The function which we fit (again) is the hypothesis

$$f(x) = \beta_0 + x^\mathsf{T}\beta = \beta_0 + \sum_{i=1}^{n} \alpha_i y_i x^\mathsf{T} x_i,$$

and the resulting classifier is

$$G(x) = \mathbf{sgn}[f(x)].$$

Support vector classifier

The function which we fit (again) is the hypothesis

$$f(x) = \beta_0 + x^\mathsf{T}\beta = \beta_0 + \sum_{i=1}^{n} \alpha_i y_i x^\mathsf{T} x_i,$$

and the resulting classifier is

$$G(x) = \mathbf{sgn}[f(x)].$$

A very important take for the proceeding part of our SVM journey is that classification of a point x only depends on the dot product with x and the support vectors x_i

$$G(x) = \mathbf{sgn}[f(x)] = \mathbf{sgn}\left[\beta_0 + \sum_{i=1}^{n} \alpha_i y_i x^\mathsf{T} x_i\right].$$

Support vector classifier

The function which we fit (again) is the hypothesis

$$f(x) = \beta_0 + x^\mathsf{T}\beta = \beta_0 + \sum_{i=1}^{n} \alpha_i y_i x^\mathsf{T} x_i,$$

and the resulting classifier is

$$G(x) = \mathbf{sgn}[f(x)].$$

A very important take for the proceeding part of our SVM journey is that classification of a point x only depends on the dot product with x and the support vectors x_i

$$G(x) = \mathbf{sgn}[f(x)] = \mathbf{sgn}\left[\beta_0 + \sum_{i=1}^{n} \alpha_i y_i x^\mathsf{T} x_i\right].$$

The dot product is an example of a so-called *inner product* which is typically denoted by $\langle \cdot, \cdot \rangle$. Henceforth, we adopt this notation which will turn out to be convenient for us. Hence,

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i y_i \langle x, x_i \rangle.$$

SVM in the cost + regularization paradigm

It is possible to formulate the support vector classifier in terms of a more familiar setting of optimization, namely

$$\min_{\beta_0,\beta_1,\ldots,\beta_p} \left(\sum_{i=1}^{n} \max[0, 1 - y_i f(x_i)] + \frac{\lambda}{2} \sum_{j=1}^{p} \beta_j^2 \right),$$

i.e. cost + regularization penalty.

SVM in the cost + regularization paradigm

It is possible to formulate the support vector classifier in terms of a more familiar setting of optimization, namely

$$\min_{\beta_0,\beta_1,\ldots,\beta_p} \left(\sum_{i=1}^{n} \max[0, 1 - y_i f(x_i)] + \frac{\lambda}{2} \sum_{j=1}^{p} \beta_j^2 \right),$$

i.e. cost + regularization penalty.

The cost function here is known as the *hinge loss*. It is convex but not (everywhere) differentiable, thus requiring some extra work in order to perform gradient descent.

SVM in the cost + regularization paradigm

It is possible to formulate the support vector classifier in terms of a more familiar setting of optimization, namely

$$\min_{\beta_0,\beta_1,\ldots,\beta_p} \left(\sum_{i=1}^{n} \max[0, 1 - y_i f(x_i)] + \frac{\lambda}{2} \sum_{j=1}^{p} \beta_j^2 \right),$$

i.e. cost + regularization penalty.

The cost function here is known as the *hinge loss*. It is convex but not (everywhere) differentiable, thus requiring some extra work in order to perform gradient descent.

In this manner we see that the support vector classifier behaves quite similar to logistic regression with ridge regularization.

SVM in the cost + regularization paradigm

It is possible to formulate the support vector classifier in terms of a more familiar setting of optimization, namely

$$\min_{\beta_0,\beta_1,\ldots,\beta_p} \left(\sum_{i=1}^{n} \max[0, 1 - y_i f(x_i)] + \frac{\lambda}{2} \sum_{j=1}^{p} \beta_j^2 \right),$$

i.e. cost + regularization penalty.

The cost function here is known as the *hinge loss*. It is convex but not (everywhere) differentiable, thus requiring some extra work in order to perform gradient descent.

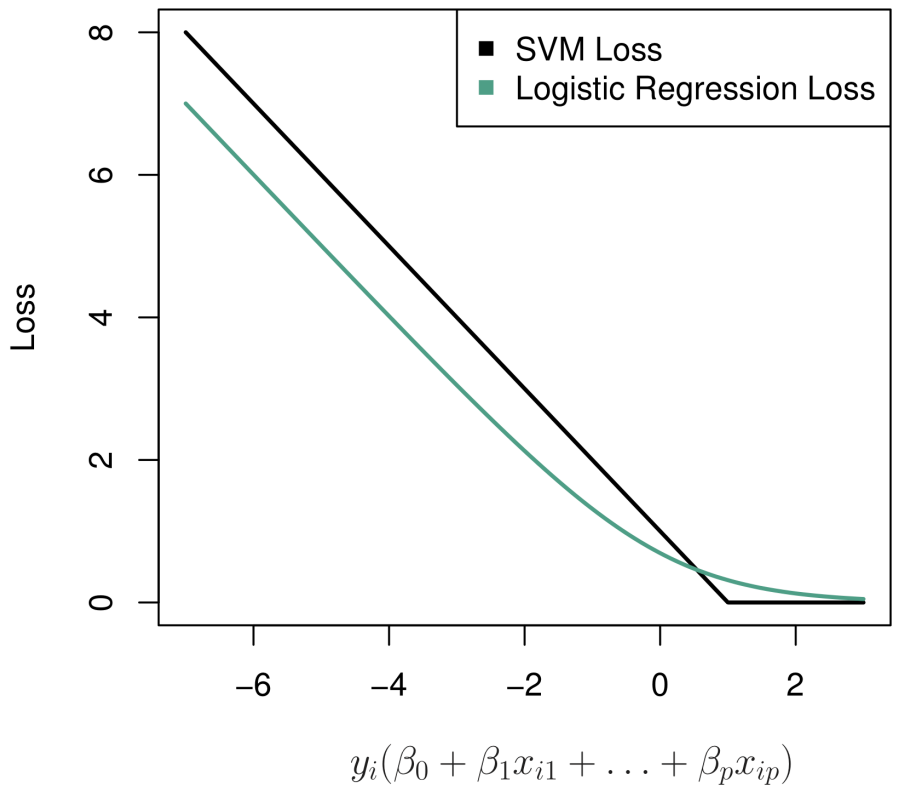
In this manner we see that the support vector classifier behaves quite similar to logistic regression with ridge regularization.

Note that

$$\text{hinge} + \text{quadratic} = \text{convex}.$$

Hinge loss and binomial deviance

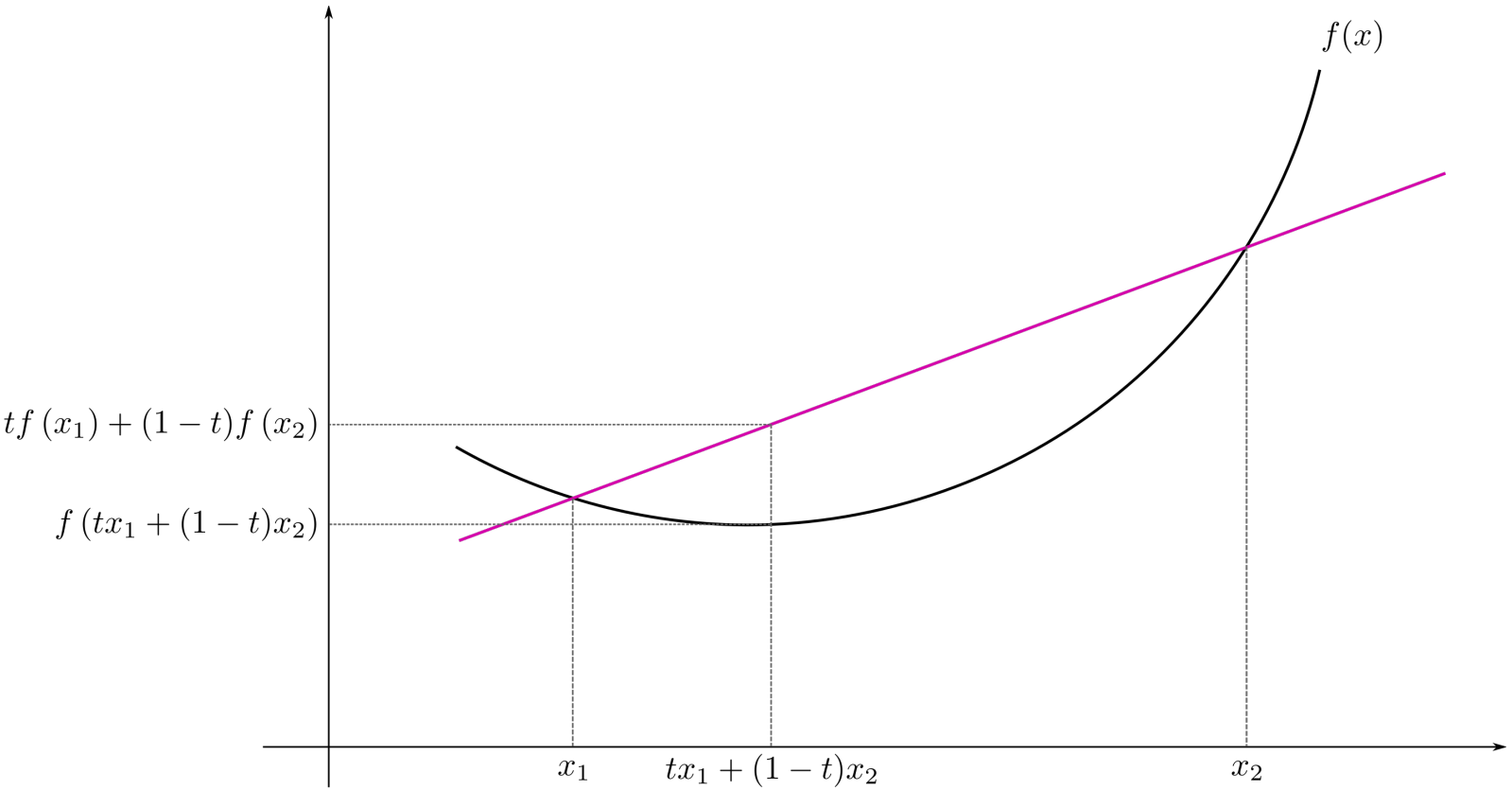
Presented below are the cost functions for support vectors classifiers and logistic regression respectively.



Convex functions

A function f is said to be convex if for every x, y in its domain and every λ in the unit interval we have

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y).$$



Hinge + quadratic

Theorem

Any local minimum of a convex function is a global minimum.

Hinge + quadratic

Theorem

Any local minimum of a convex function is a global minimum.

- Good news, since this implies that gradient descent yields the minimum
- Bad news are, as previously announced, the loss function is *not* everywhere differentiable.
- A solution is to use: sub-gradients
- This however is out of scope of the course and we'll only introduce it for sake of completeness.

Gradient descent version of the problem

Our cost function is

$$J(\beta) = \left(\sum_{i=1}^{n} \max[0, 1 - y_i f(x_i)] + \frac{\lambda}{2} \sum_{j=1}^{p} \beta_j^2 \right).$$

Note that if $g(x) = \max(x, 0)$, then

$$g'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise.} \end{cases}$$

Thus we have

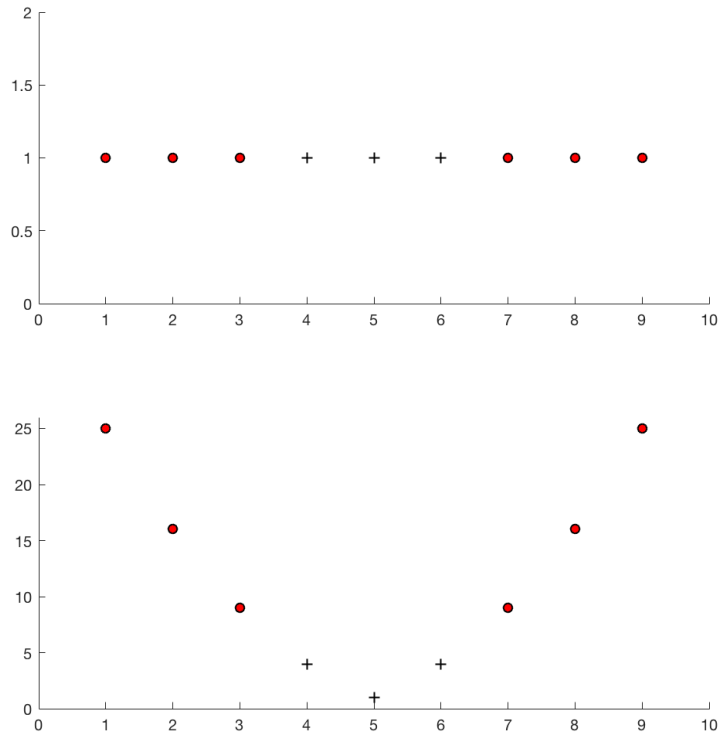
$$\nabla J(\beta) = \lambda\beta + \sum_{i=1}^{n} s(x_i, y_i),$$

where

$$s(x, y) = \begin{cases} -yx, & \text{if } yx < 1 \\ 0, & \text{otherwise.} \end{cases}$$

Extending further – solving non-linear problems

Perform a transformation of the data in order to make it linearly separable. Here is an example where $x \mapsto (x, x^2)$.



Problem: There are many ways to transforming the data.

Implicit enlargement of the feature space

We've already seen how to solve non-linear problems for instance by enlarging the feature space. However there are some problems associated to this task. What if there are 100 features, and we want to find all polynomial combinations of these of degree less than or equal to 5? Then we extend our feature space to include $>$ 79 million(!) features $\Rightarrow$ computational issues.

Our aim now is to find some other suitable way to implicitly enlarge the feature space.

The main idea that we want to obtain is how to enlarge the feature space to allow for non-linear decision boundaries, without having to add pre-defined new features.

Kernels

The feature space can implicitly be extended by the use of *kernels*.

Kernels can be thought of as a generalization of the dot-product in some implicit feature-space, which might be significantly larger (or even infinite). It can also be considered a measure of similarity.

Example of kernels K to use are

- linear kernel (this gives the support vector classifier):

$$K(x_i, x_j) = \langle x_i, x_j \rangle.$$

- polynomial kernel:

$$K(x_i, x_j) = (1 + \langle x_i, x_j \rangle)^d.$$

- Gaussian (RBF) kernel:

$$K(x_i, x_j) = \mathbf{exp}(-\gamma ||x_i - x_j||^2) = e^{-\gamma \sum_{k=1}^{p} (x_{ki} - x_{kj})^2}$$

Kernels

The feature space can implicitly be extended by the use of *kernels*.

Kernels can be thought of as a generalization of the dot-product in some implicit feature-space, which might be significantly larger (or even infinite). It can also be considered a measure of similarity.

Example of kernels K to use are

- linear kernel (this gives the support vector classifier):

$$K(x_i, x_j) = \langle x_i, x_j \rangle.$$

- polynomial kernel:

$$K(x_i, x_j) = (1 + \langle x_i, x_j \rangle)^d.$$

- Gaussian (RBF) kernel:

$$K(x_i, x_j) = \exp(-\gamma ||x_i - x_j||^2) = e^{-\gamma \sum_{k=1}^{p} (x_{ki} - x_{kj})^2}$$

Why is this better? Requires (significantly) less computations as we do not have to explicitly extend the feature space.

Kernels

As we said before Kernels are dot-products in extended features spaces, *i.e.* another formulation of kernel is

$$K(x, y) = \langle \varphi(x), \varphi(y) \rangle,$$

where φ denotes a projection onto the extended feature space.

New kernels can be constructed from other kernels in the following way (the list is *not* complete). Assume that $c > 0$ is a constant, f is any function and q is a polynomial with non-negative coefficients and finally that k_1, k_2 are valid kernels. Then the following are valid kernels

- $K(x, y) = ck_1(x, y)$
- $K(x, y) = f(x)k_1(x, y)f(y)$
- $K(x, y) = q(k_1(x, y))$
- $K(x, y) = \exp(k_1(x, y))$
- $K(x, y) = k_1(x, y) + k_2(x, y)$
- $K(x, y) = k_1(x, y)k_2(x, y)$

Gaussian kernel is a kernel

We recall that the Gaussian kernel was given by

$$K(x,y) = \exp(-\gamma||x-y||^2).$$

First we note that

$$||x-y||^2 = \langle x,x\rangle + \langle y,y\rangle - 2\langle x,y\rangle.$$

Denote by $f(x) = \langle x,x\rangle$. Then we have

$$\exp(-\gamma||x-y||^2) = \exp(2\gamma f(x)\langle x,y\rangle f(y)).$$

By the results from last slide this is in fact a kernel.

Gaussian kernel is very flexible

The Gaussian kernel captures *local* information since $K(x, y) \to 0$ as $||x - y|| \to \infty$. This also allows for capturing highly complex datamargins.

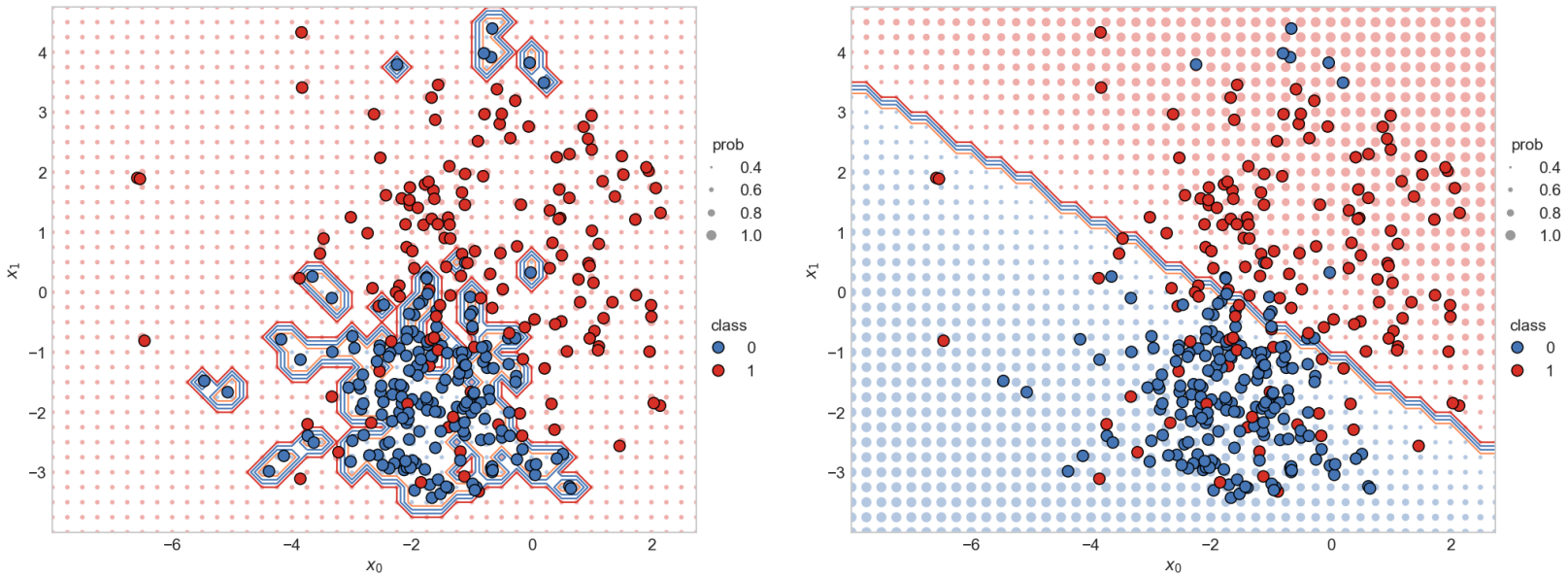


Figure: Left: RBF kernel, Right: Linear kernel

The strength of kernels

Consider the polynomial kernel: $K(x, y) = (1 + \langle x, y \rangle)^2$. Let $x = (x_1, x_2)$ and $y = (y_1, y_2)$.

The strength of kernels

Consider the polynomial kernel: $K(x, y) = (1 + \langle x, y \rangle)^2$. Let $x = (x_1, x_2)$ and $y = (y_1, y_2)$. Then

$$\begin{aligned} K(x, y) &= (1 + \langle x, y \rangle)^2 = (1 + x_1 y_1 + x_2 y_2)^2 \\ &= 1 + (x_1 y_1)^2 + (x_2 y_2)^2 + 2x_1 y_1 + 2x_2 y_2 + 2x_1 y_1 x_2 y_2). \end{aligned}$$

However, note that

$$K(x, y) = \langle (1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2), (1, y_1^2, y_2^2, \sqrt{2}y_1, \sqrt{2}y_2, \sqrt{2}y_1 y_2) \rangle.$$

Hence, the kernel implicitly computed a scalar product in a higher dimensional space, without having to map x and y to its higher features. This is known as the *kernel trick*.

The strength of kernels

Consider the polynomial kernel: $K(x,y) = (1 + \langle x, y\rangle)^2$. Let $x = (x_1, x_2)$ and $y = (y_1, y_2)$. Then

$$\begin{aligned} K(x,y) &= (1 + \langle x, y\rangle)^2 = (1 + x_1 y_1 + x_2 y_2)^2 \\ &= 1 + (x_1 y_1)^2 + (x_2 y_2)^2 + 2x_1 y_1 + 2x_2 y_2 + 2x_1 y_1 x_2 y_2). \end{aligned}$$

However, note that

$$K(x,y) = \langle (1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2), (1, y_1^2, y_2^2, \sqrt{2}y_1, \sqrt{2}y_2, \sqrt{2}y_1 y_2)\rangle.$$

Hence, the kernel implicitly computed a scalar product in a higher dimensional space, without having to map x and y to its higher features. This is known as the *kernel trick*.

In the case that the map $\phi(x) = (1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2)$ maps to a degree $d = 6$ polynomial it is not of much use, but technically we may let $d \to \infty$, which is not feasible if we need explicit computations.

Kernel Ridge Regression

Linear (ridge) regression is given by solving

$$\widehat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} (\theta^{\mathsf{T}} x_i - y_i)^2 + \lambda ||\theta||^2$$

This is solved by the normal equations

$$\widehat{\theta} = (X^{\mathsf{T}} X + n\lambda I)^{-1} X^{\mathsf{T}} y.$$

Kernel Ridge Regression

Linear (ridge) regression is given by solving

$$\widehat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} (\theta^{\mathsf{T}} x_i - y_i)^2 + \lambda ||\theta||^2$$

This is solved by the normal equations

$$\widehat{\theta} = (X^{\mathsf{T}} X + n\lambda I)^{-1} X^{\mathsf{T}} y.$$

By applying a transformation ϕ to x as a preprocess we obtain

$$\widehat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} (\theta^{\mathsf{T}} \phi(x_i) - y_i)^2 + \lambda ||\theta||^2$$

solved by

$$\widehat{\theta} = (\Phi(X)^{\mathsf{T}} \Phi(X) + n\lambda I)^{-1} \Phi(X)^{\mathsf{T}} y$$

If $\phi(x) = (1, x, x^2, x^3, \ldots, x^d)$ then the above is equivalent to polynomial regression. This problem simplifies due to the kernel trick.

Support vector machines

When combining support vector classifiers with kernels we obtain what is usually called the support vector machines.

The SVM formulation is

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i y_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \phi(x_i)^\intercal \phi(x_j),$$

where $\phi(x_i)^\intercal \phi(x_j)$ is represented using a kernel.

Advantages of support vector machines

- very flexible (illustrated further in Assignment 3)
- robust (c.f. maximal margin classifier)
- built-in regularization C (which has to be tuned).

SVM hyperparameters

There are several hyperparameters of the support vector machines which has to be tuned in order to obtain good performance.

Some examples of hyperparameters are

- The penalty factor C
- The scale parameter γ for Gaussian kernels
- The degree d for polynomial kernels

In order to achieve optimal results these hyperparameters should be optimized and unfortunately it can be quite costful to find them. Also, note that poor choice of the hyperparameters can radically shift the results. So don't be scared of by poor results right away.

General advice: hyperparameter tuning may by very time consuming, hence if you have many training points use only a subset for training the hyperparameters, which can give good estimates.

Binarization techniques

There are two very common and much used binarization techniques, *i.e.* methods for handling mutliclass classification.

Previously you've seen the *one-vs-all* techniques (Lecture 3). In `sklearn` for SVM multi-class the built-in technique is *one-vs-one*.

Algorithm: one-vs-one classification
(X, y) is the dataset, k is the number of classes

1. For $i = 1, \ldots, k$
   - 1.1 For $j = 1, \ldots, i-1$
     - 1.1.1 Put $X_t := X[(y == i \,||\, y == j), :]$
     - 1.1.2 Put $y_t := y[(y == i \,||\, y == j)]$
     - 1.1.3 Train a (binary) classifier using (X_t, y_t)
2. Store each of the $\binom{k}{2}$ classifiers and let majority voting decide predictions

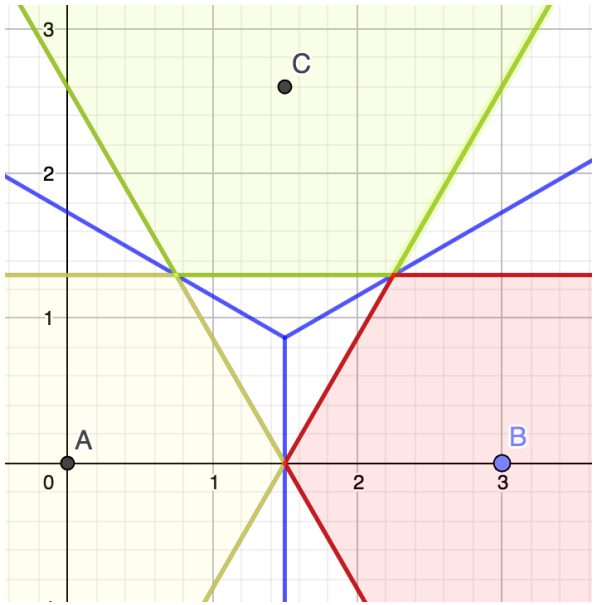
Numpy note: The OR-command $||$ may in numpy be used for instance by `np.logical_or`.

OVA vs OVO

Consider a three-class classification task with only three training examples as seen below. Suppose that a maximal margin classifier is used and trained on the mentioned training examples, together with either (i) OVO binarization or (ii) OVA binarization.

a) In each of these cases (i) and (ii). How many classifiers needs to be trained?

b) Are the same decision boundaries produced by (i) and (ii)?



SVM in `sklearn`

In scikit-learn the command for creating a support vector machines is given by

- `svm.SVC()`

after importing svm from `sklearn`.

The typical named-valued-pairs which you have to pass are

- `'kernel'` rbf (default), linear, poly
- `'gamma'` for instance γ for Gaussian and coefficient for poly
- `'C'` is simply the C parameter
- `'degree'` is simply the degree for polynomial kernel

Assignment summary

This lecture comes with 3 exercises in Assignment 3.

1. Various kernels: In this exercise you will compare various kernels for SVM, on a dummy dataset.
2. Implicit versus explicit (VG-exercise): In this exercise you should compare the training time using implicit and explicit mappings to higher dimension spaces.
3. One versus all MNIST: (only 2dv516) In this exercise you will train a support vector machine with an rbf kernel to recognize handwritten digits in the MNIST dataset. You will also implement your own version of the one-versus-all scheme for multiclass-problems to be compared to the built-in one-vs-one scheme.