Section 4

Soft Margin Support Vector Machines

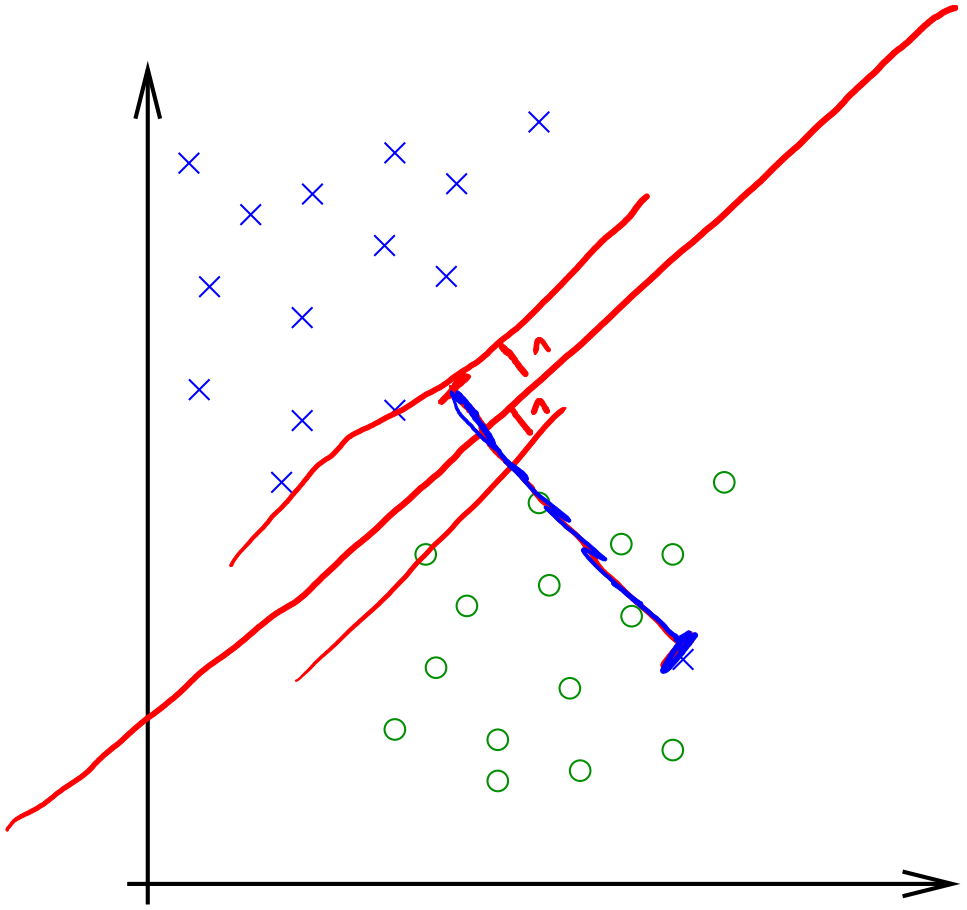
Dealing with noisy data

What if the data is not linearly separable due to some noise?

With our current version of SVM, a single outlier makes the constraint unsatisfiable.

The corresponding Lagrange multiplier α_i would go to infinity and destroy the solution.

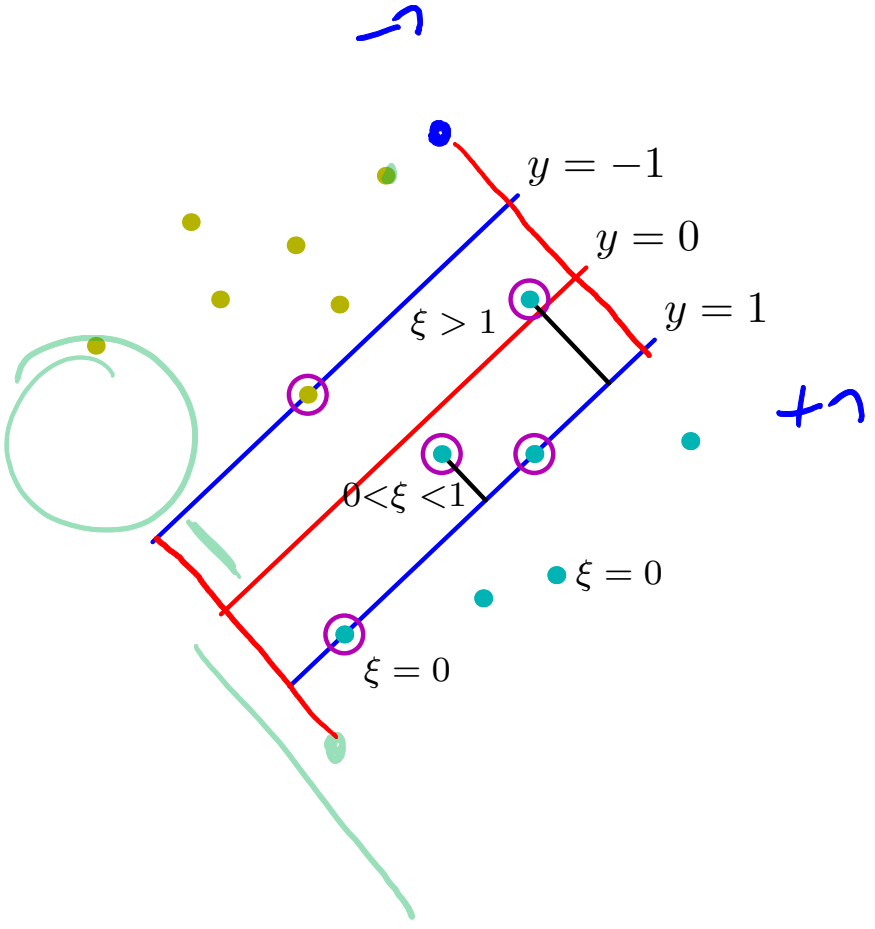
How to make our model more robust?



Slack variables

Idea: Relax the constraints as much as necessary but punish the relaxation of a constraint.

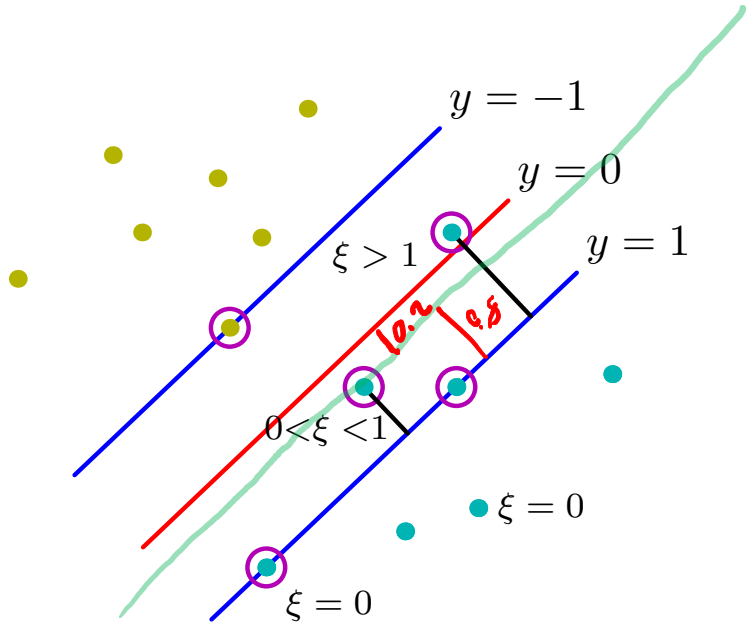
We introduce a slack variable $\xi_i \geq 0$ for every training sample $\boldsymbol{x_i}$ that gives the distance of how far the margin is violated by this training sample in units of $||\boldsymbol{w}||$.



Slack variables

Idea: Relax the constraints as much as necessary but punish the relaxation of a constraint.

We introduce a slack variable $\xi_i \geq 0$ for every training sample $\boldsymbol{x_i}$ that gives the distance of how far the margin is violated by this training sample in units of $||\boldsymbol{w}||$.



Hence our relaxed constraints are

$$\boldsymbol{w}^T \boldsymbol{x}_i + b \geq +1 - \xi_i \quad \text{for } y_i = +1\,,$$
$$\boldsymbol{w}^T \boldsymbol{x}_i + b \leq -1 + \xi_i \quad \text{for } y_i = -1\,.$$

Again, they can be condensed into

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 - \xi_i \quad \text{for all } i\,.$$

Slack variables

Idea: Relax the constraints as much as necessary but punish the relaxation of a constraint.

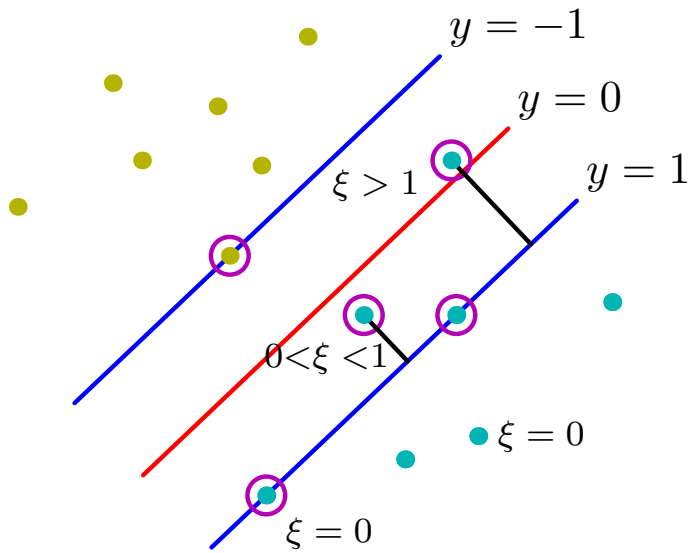
We introduce a slack variable $\xi_i \geq 0$ for every training sample $\boldsymbol{x_i}$ that gives the distance of how far the margin is violated by this training sample in units of $||\boldsymbol{w}||$.

Hence our relaxed constraints are

$$\boldsymbol{w}^T \boldsymbol{x}_i + b \geq +1 - \xi_i \quad \text{for } y_i = +1\,,$$
$$\boldsymbol{w}^T \boldsymbol{x}_i + b \leq -1 + \xi_i \quad \text{for } y_i = -1\,.$$

Again, they can be condensed into

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 - \xi_i \quad \text{for all } i\,.$$



The new cost function is,

$$\min f_0(\boldsymbol{w}, b, \boldsymbol{\xi}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{N}\xi_i\,.$$

The factor $C > 0$ determines how heavy a violation is punished.

$C \to \infty$ recovers hard-margin SVM.

Optimization problem with slack variables

Let $\boldsymbol{x}_i$ be the ith data point, $i = 1, \ldots, N$, and $y_i \in \{-1, 1\}$ the class assigned to $\boldsymbol{x}_i$. Let $C > 0$ be a constant.

To find the hyperplane that separates most of the data with maximum margin we

$$\begin{aligned} \text{minimize} \quad & f_0(\boldsymbol{w}, b, \boldsymbol{\xi}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1 + \xi_i \geq 0 \qquad i = 1, \ldots, N\,, \\ & \xi_i \geq 0 \qquad i = 1, \ldots, N\,. \end{aligned}$$

Here we used the 1-norm for the penalty term $\sum_i \xi_i$. Another choice is to use the 2-norm penalty, $\sum_i \xi_i^2$.

The penalty that performs better in practice will depend on the data and the type of noise that has influenced it.

Lagrangian with slack variables

LAGRANGE MULTIPLIERS

1. Calculate the Lagrangian

$$L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i[y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1 + \xi_i] - \sum_{i=1}^{N}\mu_i\xi_i\,.$$

2. Minimize $L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu})$ w.r.t. $\boldsymbol{w}$, b and $\boldsymbol{\xi}$.

$w^*(\alpha,\mu)$ $\quad$ $L^*(\alpha,\mu) = ?$

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = \boldsymbol{w} - \sum_{i=1}^{N}\alpha_i y_i \boldsymbol{x_i} = 0; \qquad \frac{\partial L}{\partial b} = \sum_{i=1}^{N}\alpha_i y_i = 0,$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \mu_i = 0 \quad \text{for} \quad i = 1, \ldots, N$$

From $\alpha_i = C - \mu_i$ and dual feasibility $\mu_i \geq 0$, $\alpha_i \geq 0$ we get

$$0 \leq \alpha_i \leq C\,.$$

$g(\alpha,\mu) = ?$

Dual problem with slack variables

This leads to the dual problem:

$$\text{maximize} \quad g(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \boldsymbol{x_i}^T \boldsymbol{x_j}$$

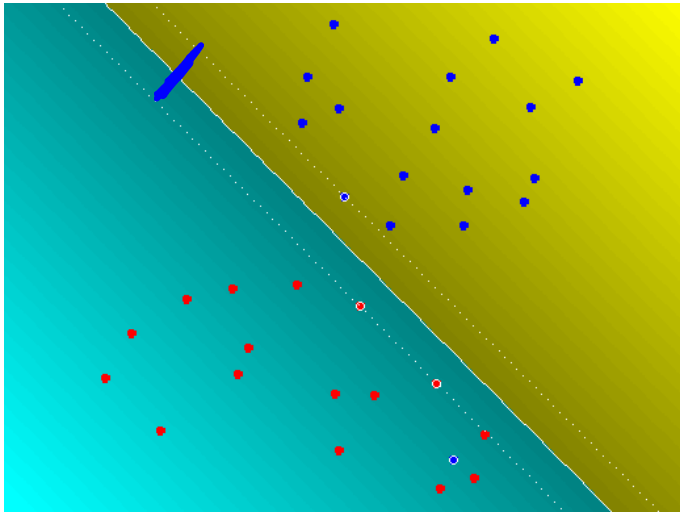
$$\text{subject to} \quad \sum_{i=1}^{N} \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C \qquad i = 1, \ldots, N \, .$$

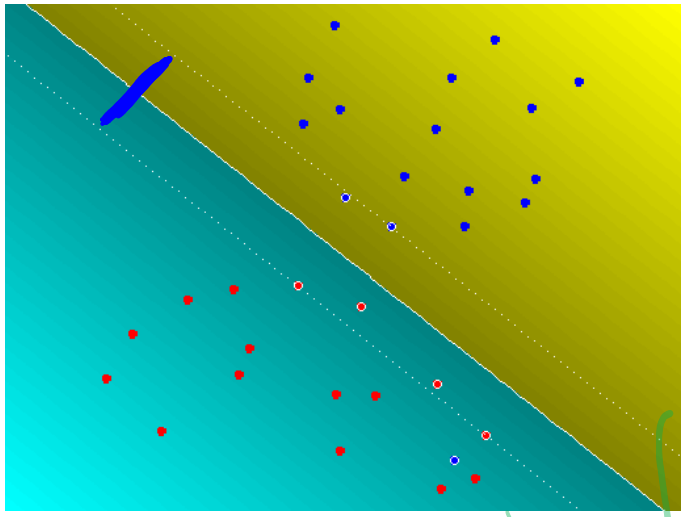
This is nearly the same dual problem as for the case without slack variables.

Only the constraint $\alpha_i \leq C$ is new. It ensures that α_i is bounded and cannot go to infinity.

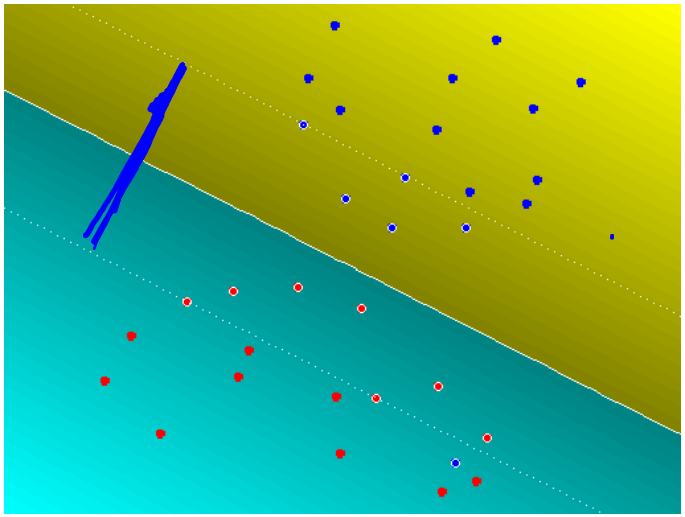
Influence of the penalty C



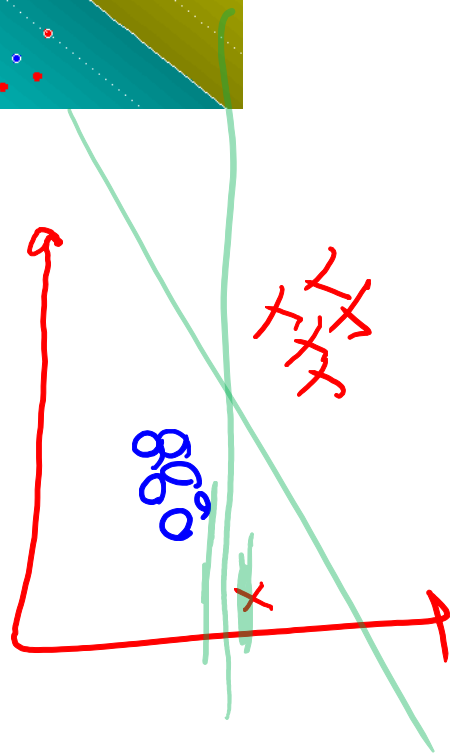
$C = 100$



$C = 10$



$C = 1$



Hinge loss formulation

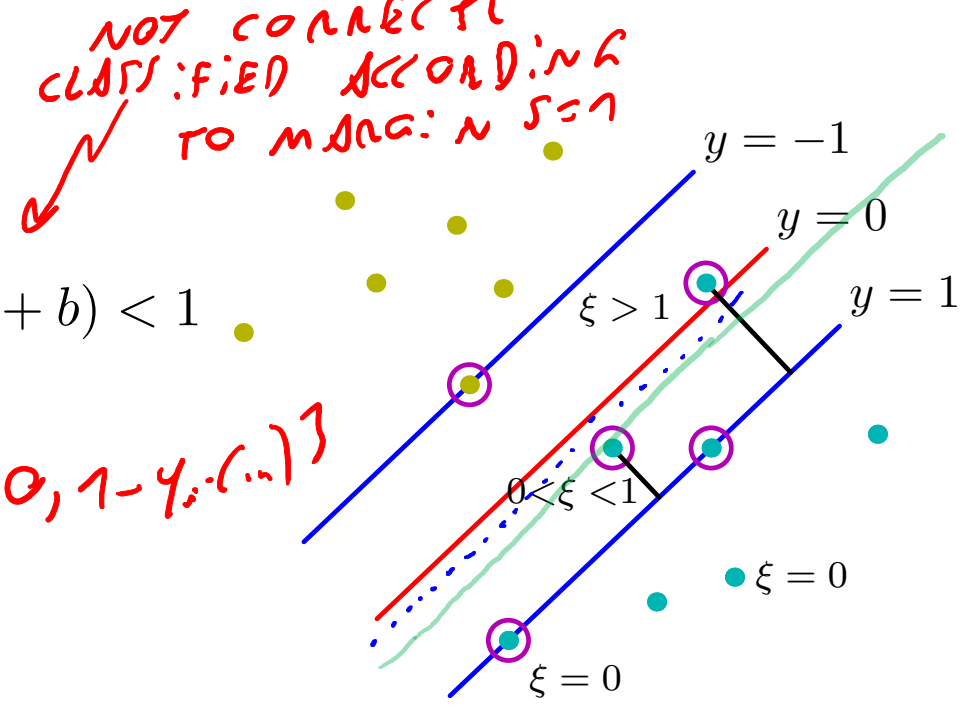
We can have another look at our constrained optimization problem.

$$\begin{aligned} \text{minimize} \quad & f_0(\boldsymbol{w}, b, \boldsymbol{\xi}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1 + \xi_i \geq 0, \qquad i = 1, \ldots, N\,, \\ & \xi_i \geq 0, \qquad i = 1, \ldots, N\,. \end{aligned}$$

Clearly, for the optimal solution the slack variables are

$$\xi_i = \begin{cases} 1 - y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b), & \text{if } y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) < 1 \\ 0 & \text{else} \end{cases}$$

since we are minimizing over $\boldsymbol{\xi}$.



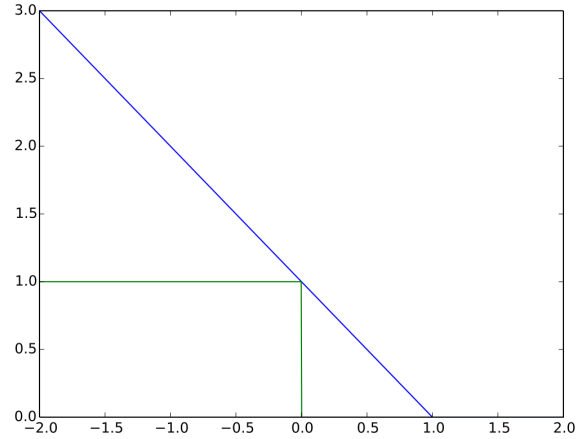
Hinge loss formulation

Thus, we can rewrite the objective function as an unconstrained optimization problem known as the hinge loss formulation

$$\min_{\boldsymbol{w},b} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{N} \max\{0, 1 - y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b)\}$$

The hinge loss function $E_{\text{hinge}}(z) = \max\{0, 1 - z\}$ penalizes the points that lie within the margin.

The name comes from the shape from the function, as can be seen in the figure to the right.



We can optimize this hinge loss objective directly, using standard gradient-based methods.

Hinge loss (blue) can be viewed as an approximation to zero-one loss (green).

Section 5

Kernels

Feature space

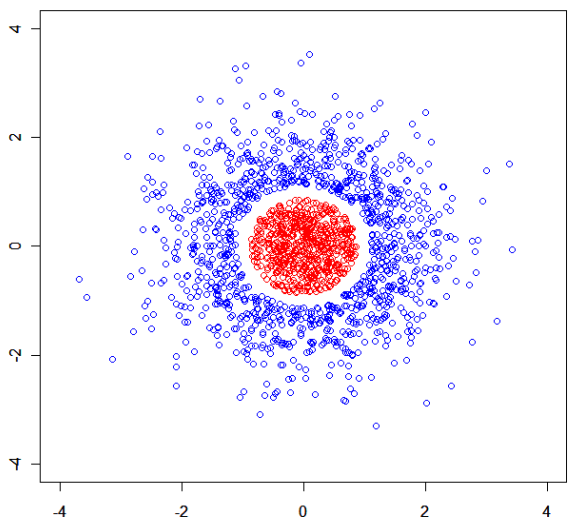
So far we can only construct linear classifiers.

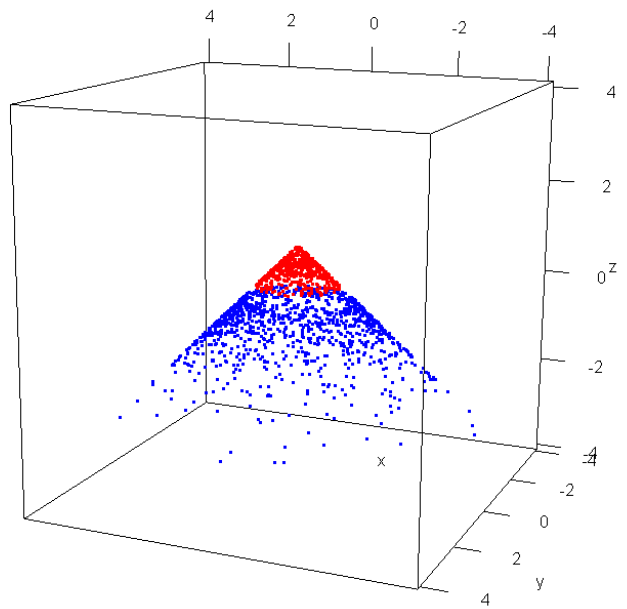
Before, we used basis functions $\phi(\cdot)$ to make the models nonlinear

$$\boldsymbol{\phi} : \mathbb{R}^D \to \mathbb{R}^M \qquad \boldsymbol{x_i} \mapsto \boldsymbol{\phi}(\boldsymbol{x_i})$$

For example, with the following mapping the data becomes linearly separable

$$\boldsymbol{\phi}(x, y) = \begin{pmatrix} x \\ y \\ -\sqrt{x^2 + y^2} \end{pmatrix}$$





Kernel trick

In the dual formulation of SVM, the samples $\boldsymbol{x}_i$ only enter the dual objective as inner products $\boldsymbol{x_i^T x_j}$

$$g(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \boldsymbol{x_i^T x_j} \,,$$

For basis functions this means that

$$g(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \boldsymbol{\phi(x_i)^T \phi(x_j)}$$

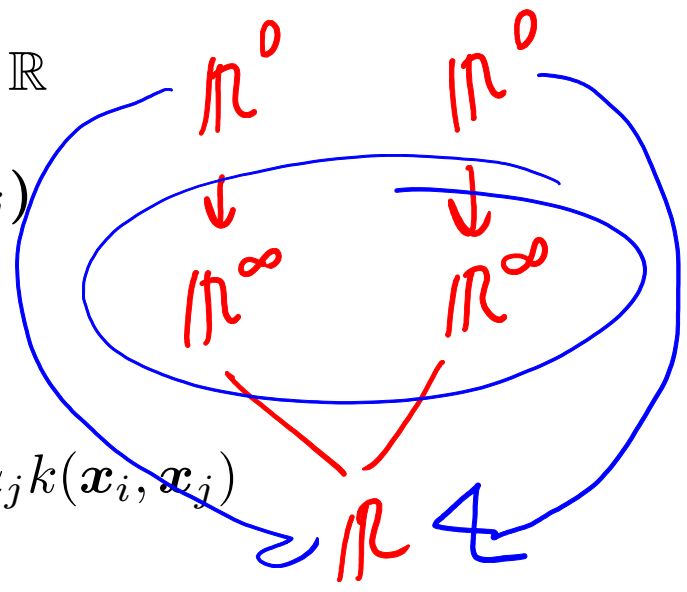
Kernel trick

We can define a kernel function $k : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) := \boldsymbol{\phi}(\boldsymbol{x_i})^T \boldsymbol{\phi}(\boldsymbol{x_j})$$

and rewrite the dual as

$$g(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$$



This operation is referred to as the kernel trick.

It can be used not only for SVM. Kernel trick can be used in any model that can be formulated such that it only depends on the inner products $\boldsymbol{x}_i^T \boldsymbol{x}_j$. (e.g. linear regression, k-nearest neighbors)

Kernel trick

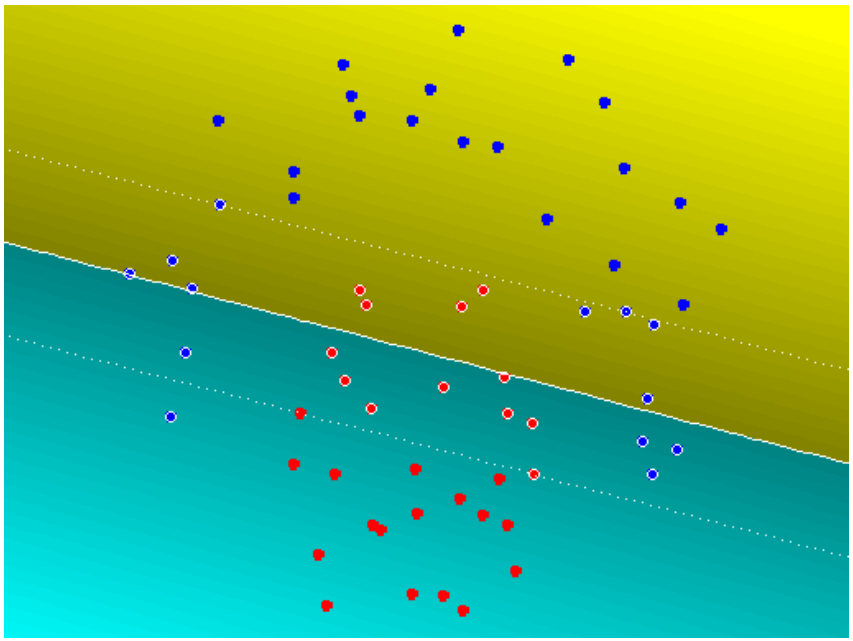
The kernel represent an inner product in the feature space spanned by ϕ.
Like before, this makes our models non-linear w.r.t. the data space.

What's the point of using kernels if we can simply use the basis functions?

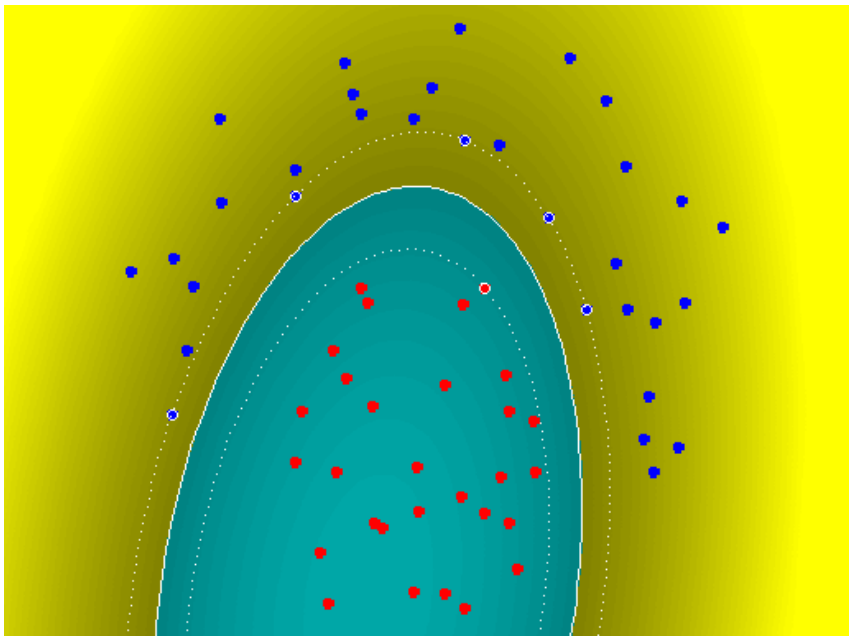
- Some kernels are equivalent to using infinite-dimensional basis functions. While computing these feature transformations would be impossible, directly evaluating the kernels is often easy.
- Kernels can be used to encode similarity between arbitrary non-numerical data, from strings to graphs.
  For example, we could define

$$k(\texttt{lemon}, \texttt{orange}) = 10 \quad \text{and} \quad k(\texttt{apple}, \texttt{orange}) = -5$$

SVM using kernels example



Linear kernel (no kernel)
$\boldsymbol{a}^T\boldsymbol{b}$



2nd order polynomial kernel
$(\boldsymbol{a}^T\boldsymbol{b})^2$

What makes a valid kernel?

A kernel is valid if it corresponds to an inner product in some feature space. An equivalent formulation is given by Mercer's theorem.

Mercer's theorem

A kernel is valid if it gives rise to a symmetric, positive (semi)-definite kernel matrix $\boldsymbol{K}$ for any input data $\boldsymbol{X}$.

Kernel matrix (also known as Gram matrix) $\boldsymbol{K} \in \mathbb{R}^{N \times N}$ is defined as

$$\boldsymbol{K} = \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & k(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ k(\boldsymbol{x}_2, \boldsymbol{x}_1) & k(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_2, \boldsymbol{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_1) & k(\boldsymbol{x}_N, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{pmatrix}$$

What happens if we use a non-valid kernel?

Our optimization problem might become non-convex, so we may not get a globally optimal solution.

Kernel preserving operations

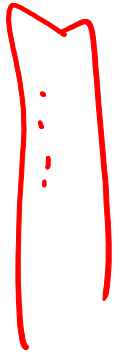
Let $k_1 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and $k_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be kernels, with $\mathcal{X} \subseteq \mathbb{R}^N$. Then the following functions k are kernels as well:

- $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = k_1(\boldsymbol{x}_1, \boldsymbol{x}_2) + k_2(\boldsymbol{x}_1, \boldsymbol{x}_2)$
- $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = c \cdot k_1(\boldsymbol{x}_1, \boldsymbol{x}_2)$, with $c > 0$
- $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = k_1(\boldsymbol{x}_1, \boldsymbol{x}_2) \cdot k_2(\boldsymbol{x}_1, \boldsymbol{x}_2)$
- $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = k_3(\phi(\boldsymbol{x}_1), \phi(\boldsymbol{x}_2))$, with the kernel k_3 on $\mathcal{X}' \subseteq \mathbb{R}^M$ and $\phi : \mathcal{X} \to \mathcal{X}'$
- $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{x}_1 \boldsymbol{A} \boldsymbol{x}_2$, with $\boldsymbol{A} \in \mathbb{R}^N \times \mathbb{R}^N$ symmetric and positive semidefinite

Examples of kernels

- Polynomial:
$$k(\boldsymbol{a}, \boldsymbol{b}) = (\boldsymbol{a}^T \boldsymbol{b})^p \quad \text{or} \quad (\boldsymbol{a}^T \boldsymbol{b} + 1)^p$$
- Gaussian kernel:
$$k(\boldsymbol{a}, \boldsymbol{b}) = \exp\left(-\frac{\|\boldsymbol{a} - \boldsymbol{b}\|^2}{2\sigma^2}\right)$$
- Sigmoid:
$$k(\boldsymbol{a}, \boldsymbol{b}) = \tanh(\kappa\, \boldsymbol{a}^T \boldsymbol{b} - \delta) \quad \text{for } \kappa, \delta > 0$$

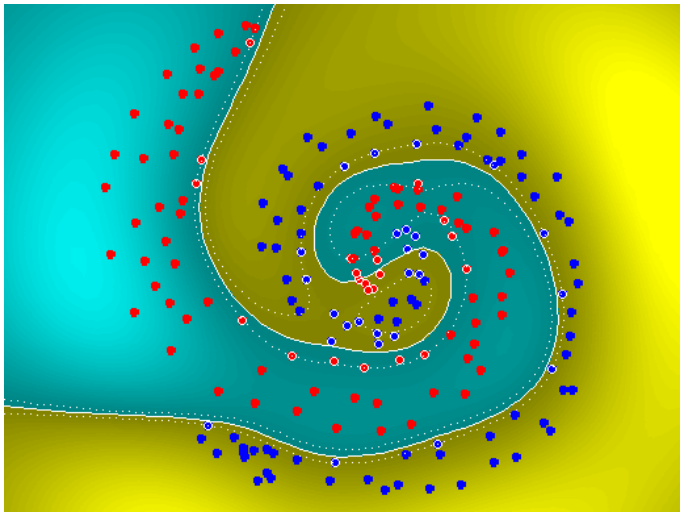
  In fact, the sigmoid kernel is not PSD, but still works well in practice.

$\phi(a) =$ 

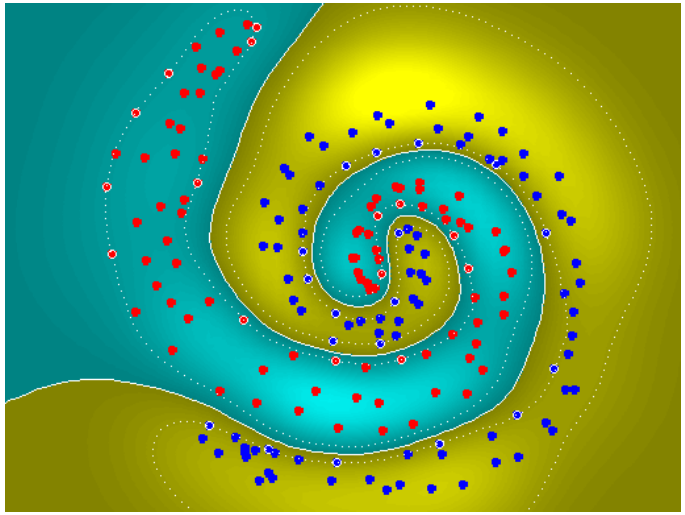
Some kernels introduce additional hyperparameters, that affect the behavior of the algorithm.

Note, that the sigmoid kernel is different from the sigmoid function from *Linear Classification*.

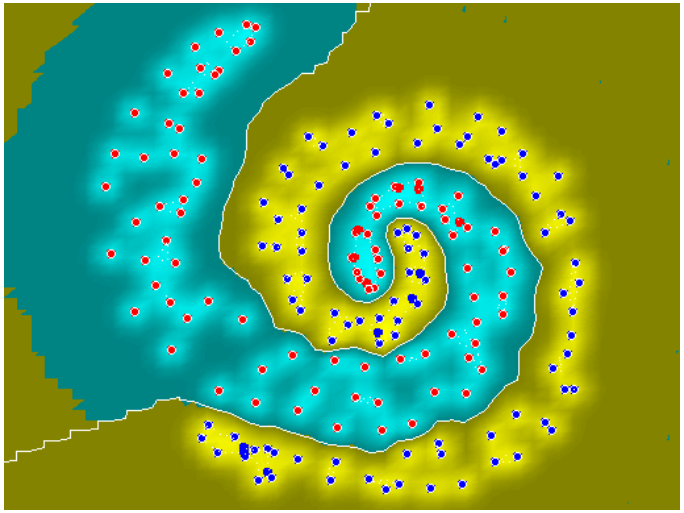
Gaussian kernel (C=1000)



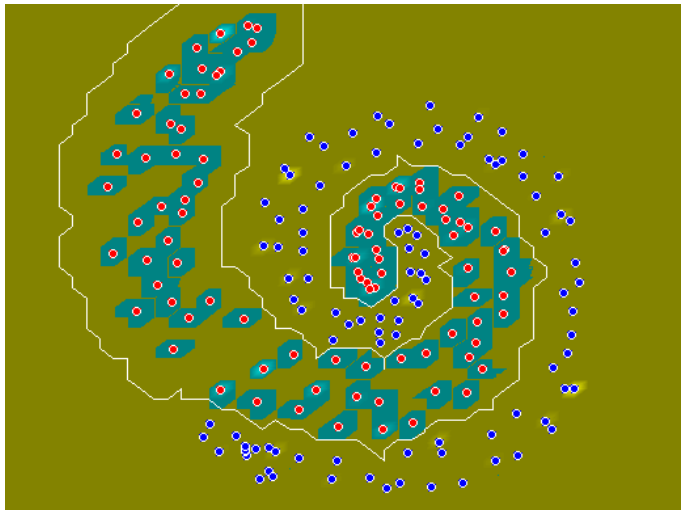
$\sigma = 0.5$



$\sigma = 0.25$



$\sigma = 0.05$



$\sigma = 0.005$

Classifying a new point with kernelized SVM

We denote the set of support vectors as $\mathcal{S}$ (points $\boldsymbol{x}_i$ for which holds $0 < \alpha_i \leq C$). Note: If $0 < \alpha_i < C$ then $\xi_i = 0$; if $\alpha_i = C$ then $\xi_i > 0$.

From the complementary slackness condition, points $\boldsymbol{x}_i \in \mathcal{S}$ with $\xi_i = 0$ must satisfy

$$y_i \left(\sum_{\{j | \boldsymbol{x}_j \in \mathcal{S}\}} \alpha_j y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) + b \right) = 1$$

Like for the regular SVM, the bias can be recovered as

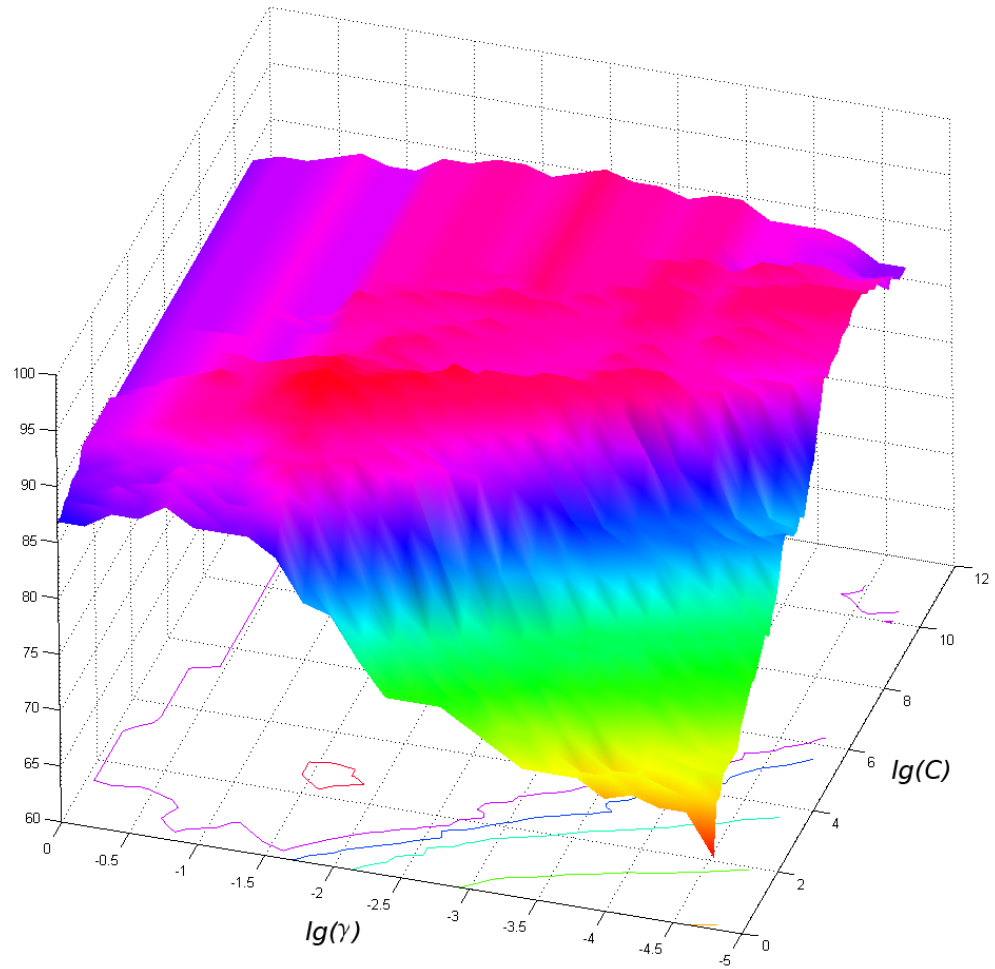
$$b = y_i - \left(\sum_{\{j | \boldsymbol{x}_j \in \mathcal{S}\}} \alpha_j y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \right)$$

Thus, a new point $\boldsymbol{x}$ can be classified as

$$h(\boldsymbol{x}) = \operatorname{sgn} \left(\sum_{\{j | \boldsymbol{x}_j \in \mathcal{S}\}} \alpha_j y_j k(\boldsymbol{x}_j, \boldsymbol{x}) + b \right)$$

How to choose the hyperparameters?

The best setting of the penalty parameter C and the kernel hyperparameters (e.g., γ or σ) can be determined by performing cross validation with random search over the parameter space.



Multiple classes

The standard SVM model cannot handle multiclass data.
Two approaches to address this issue are:

One-vs-rest: Train C SVM models for C classes, where each SVM is being trained for classification of one class against all the remaining ones. The winner is then the class, where the distance from the hyperplane is maximal.

One-vs-one: Train $\binom{C}{2}$ classifiers (all possible pairings) and evaluate all. The winner is the class with the majority vote (votes are weighted according to the distance from the margin).

Summary

- Support Vector Machine is a linear binary classifier that, motivated by learning theory, maximizes the margin between the two classes.
- The SVM objective is convex, so a globally optimal solution can be obtained.
- The dual formulation is a quadratic programming problem, that can be solved efficiently, e.g. using standard QP libraries.
- Soft-margin SVM with slack variables can deal with noisy data. Smaller values for the penalty C lead to a larger margin at the cost of misclassifying more samples.
- Linear soft-margin SVM ($=$ hinge loss formulation) can be solved directly using gradient descent.
- We can obtain a nonlinear decision boundary by moving to an implicit high-dimensional feature space by using the kernel trick. This only works in the dual formulation.