Kernel Methods, SVM

Milan Straka

November 07, 2022

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Kernel Linear Regression

Consider linear regression with linear, quadratic and cubic features (for simplicity we consider x_i, $x_i x_j$ and $x_i x_j x_k$ for any indices) computed by feature mapping $\varphi : \mathbb{R}^D \to \mathbb{R}^{1+D+D^2+D^3}$:

$$\varphi(\boldsymbol{x}) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \dots \\ x_1^2 \\ x_1 x_2 \\ \dots \\ x_2 x_1 \\ \dots \\ x_1^3 \\ x_1^2 x_2 \\ \dots \end{bmatrix}.$$

The SGD update of a linear regression using a minibatch of examples $\mathbb{B}$ is then

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{\alpha}{|\mathbb{B}|} \sum_{i \in \mathbb{B}} \big(\varphi(\boldsymbol{x}_i)^T \boldsymbol{w} - t_i\big) \varphi(\boldsymbol{x}_i).$$

Kernel Linear Regression

When the dimensionality of the input is D, one step of SGD takes $\mathcal{O}(D^3)$ per example.

Surprisingly, we can do better under some circumstances. We start by noting that we can write the parameters $\boldsymbol{w}$ as a linear combination of the N input feature vectors $\varphi(\boldsymbol{x}_i)$.

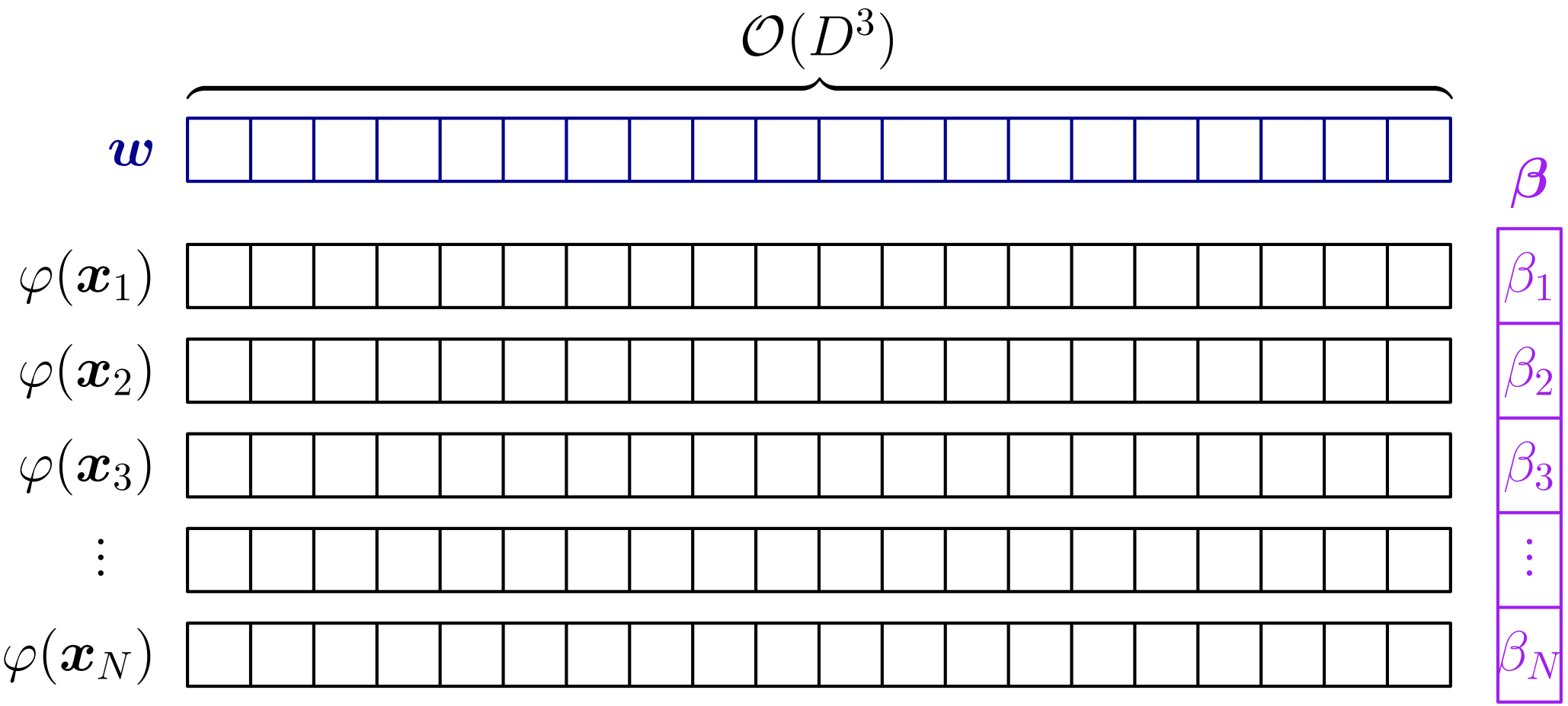
By induction, we can start with $\boldsymbol{w} = \mathbf{0} = \sum_{i=1}^{N} 0 \cdot \varphi(\boldsymbol{x}_i)$. Assuming $\boldsymbol{w} = \sum_{i=1}^{N} \beta_i \cdot \varphi(\boldsymbol{x}_i)$, after an SGD update we get

$$\begin{aligned}\boldsymbol{w} &\leftarrow \boldsymbol{w} - \frac{\alpha}{|\mathbb{B}|} \sum_{i \in \mathbb{B}} \big(\varphi(\boldsymbol{x}_i)^T \boldsymbol{w} - t_i\big)\varphi(\boldsymbol{x}_i) \\ &\leftarrow \sum_{i=1}^{N} \left(\beta_i - \big[i \in \mathbb{B}\big] \cdot \frac{\alpha}{|\mathbb{B}|}\big(\varphi(\boldsymbol{x}_i)^T \boldsymbol{w} - t_i\big)\right)\varphi(\boldsymbol{x}_i).\end{aligned}$$

Every β_i for $i \in \mathbb{B}$ changes to $\beta_i - \frac{\alpha}{|\mathbb{B}|}\big(\varphi(\boldsymbol{x}_i)^T \boldsymbol{w} - t_i\big)$, so after substituting for $\boldsymbol{w}$ we get

$$\beta_i \leftarrow \beta_i - \frac{\alpha}{|\mathbb{B}|}\left(\left(\sum_{j=1}^{N} \beta_j \varphi(\boldsymbol{x}_i)^T \varphi(\boldsymbol{x}_j)\right) - t_i\right).$$

Visualization of the Primal and Dual Representations



Kernel Linear Regression

We can formulate an alternative linear regression algorithm (a so-called **dual formulation**):

Input: Dataset ($\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\} \in \mathbb{R}^{N \times D}$, $\boldsymbol{t} \in \mathbb{R}^N$), learning rate $\alpha \in \mathbb{R}^+$.

- $\boldsymbol{\beta} \leftarrow \mathbf{0}$
- compute all values $K_{i,j} = \varphi(\boldsymbol{x}_i)^T \varphi(\boldsymbol{x}_j)$
- until convergence (or patience runs out), process a minibatch of examples $\mathbb{B}$:
  - simultaneously for all $i \in \mathbb{B}$ (the β_j on the right side must not be modified during the batch update):
    - $\beta_i \leftarrow \beta_i - \frac{\alpha}{|\mathbb{B}|}\left(\left(\sum_{j=1}^{N} \beta_j K_{i,j}\right) - t_i\right)$

The predictions for input $\boldsymbol{z}$ are then performed by computing

$$y(\boldsymbol{z}) = \varphi(\boldsymbol{z})^T \boldsymbol{w} = \sum_{i=1}^{N} \beta_i \boldsymbol{\varphi}(\boldsymbol{z})^T \boldsymbol{\varphi}(\boldsymbol{x}_i).$$

Bias in Kernel Linear Regression

Until now we did not consider *bias*. Unlike the usual formulation, where we can "hide" it in the weights, we usually handle it manually in the dual formulation.

Specifically, if we want to include bias in kernel linear regression, we modify the predictions to

$$y(\boldsymbol{z}) = \varphi(\boldsymbol{z})^T \boldsymbol{w} + b = \left(\sum_{i=1}^{N} \beta_i \boldsymbol{\varphi}(\boldsymbol{z})^T \boldsymbol{\varphi}(\boldsymbol{x}_i) \right) + b$$

and update the bias b separately.

To update the bias, we use SGD, and the resulting update is

$$b \leftarrow b - \frac{\alpha}{|\mathbb{B}|} \sum_{i \in \mathbb{B}} \left(\left(\sum_{j=1}^{N} \beta_j K_{i,j} \right) + b - t_i \right).$$

Kernel Trick

A single SGD update of a dual-formulation kernel linear regression takes $\mathcal{O}(N)$ per example, if we pre-compute all the $\mathcal{O}(N^2)$ dot products $\varphi(\boldsymbol{x}_i)^T \varphi(\boldsymbol{x}_j)$. Furthermore, inference requires evaluating $\mathcal{O}(N)$ dot products $\varphi(\boldsymbol{z})^T \varphi(\boldsymbol{x}_i)$.

Therefore, we need to compute the dot product $\varphi(\boldsymbol{x})^T \varphi(\boldsymbol{z})$ quickly.

Using the previously-defined φ, we get

$$\begin{aligned}
\varphi(\boldsymbol{x})^T \varphi(\boldsymbol{z}) &= 1 + \sum_i x_i z_i + \sum_{i,j} x_i x_j z_i z_j + \sum_{i,j,k} x_i x_j x_k z_i z_j z_k \\
&= 1 + \sum_i x_i z_i + \Big(\sum_i x_i z_i\Big)^2 + \Big(\sum_i x_i z_i\Big)^3 \\
&= 1 + \boldsymbol{x}^T \boldsymbol{z} + \big(\boldsymbol{x}^T \boldsymbol{z}\big)^2 + \big(\boldsymbol{x}^T \boldsymbol{z}\big)^3.
\end{aligned}$$

Therefore, we can compute the dot product $\varphi(\boldsymbol{x})^T \varphi(\boldsymbol{z})$ in $\mathcal{O}(D)$ instead of $\mathcal{O}(D^3)$.

We define a **kernel** corresponding to a feature map φ as a function

$$K(\boldsymbol{x}, \boldsymbol{z}) \stackrel{\text{def}}{=} \varphi(\boldsymbol{x})^T \varphi(\boldsymbol{z}).$$

There exist quite a lot of kernels, but the most commonly used are the following:

- **Polynomial kernel of degree** d, also called *homogenous polynomial kernel*,

$$K(\boldsymbol{x}, \boldsymbol{z}) = (\gamma \boldsymbol{x}^T \boldsymbol{z})^d,$$

  corresponds to a feature map returning all combinations of exactly d input features. Using $(a_1 + \ldots + a_k)^d = \sum_{n_i \geq 0, \sum n_i = d} \binom{d}{n_1, \ldots, n_k} a_1^{n_1} \cdots a_k^{n_k}$, we can verify that

$$\varphi(\boldsymbol{x}) = \left(\sqrt{\gamma^d \binom{d}{n_1, \ldots, n_D}} x_1^{n_1} \cdots x_D^{n_D} \right)_{n_i \geq 0, \sum n_i = d}.$$

  For example, for $d = 2$, $\varphi(x_1, x_2) = \gamma(x_1^2, \sqrt{2} x_1 x_2, x_2^2)$.

- **Polynomial kernel of degree at most** d, also called *nonhomogenous polynomial kernel*,

$$K(\boldsymbol{x}, \boldsymbol{z}) = (\gamma \boldsymbol{x}^T \boldsymbol{z} + 1)^d,$$

corresponds to a feature map generating all combinations of up to d input features.

Given that $(\gamma \boldsymbol{x}^T \boldsymbol{z} + 1)^d = \sum_i \binom{d}{i} (\gamma \boldsymbol{x}^T \boldsymbol{z})^i$, it is not difficult to derive that

$$\varphi(\boldsymbol{x}) = \left(\sqrt{\gamma^{d-n_{D+1}} \binom{d}{n_1, \ldots, n_{D+1}}} x_1^{n_1} \cdots x_D^{n_D} \right)_{n_i \geq 0,\, \sum_{i=1}^{D+1} n_i = d}.$$

For example, for $d = 2$, $\varphi(x_1, x_2) = (1, \sqrt{2\gamma} x_1, \sqrt{2\gamma} x_2, \gamma x_1^2, \sqrt{2}\gamma x_1 x_2, \gamma x_2^2)$.

- **Gaussian Radial basis function (RBF)** kernel

$$K(\boldsymbol{x}, \boldsymbol{z}) = e^{-\gamma\|\boldsymbol{x}-\boldsymbol{z}\|^2},$$

corresponds to a dot product in an infinite-dimensional space; it is a combination of polynomial kernels of all degrees. Assuming $\gamma = 1$ for simplicity, we get

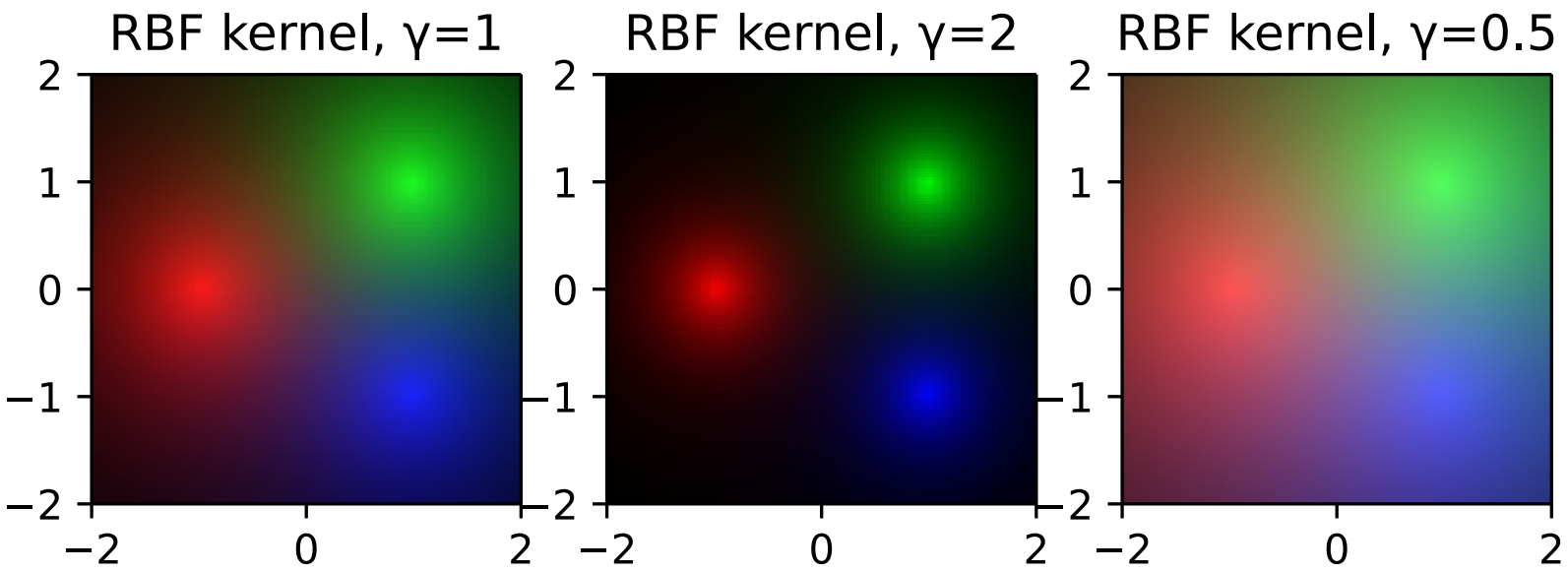
$$e^{-\|\boldsymbol{x}-\boldsymbol{z}\|^2} = e^{-\|\boldsymbol{x}\|^2+2\boldsymbol{x}^T\boldsymbol{z}-\|\boldsymbol{z}\|^2} = \sum_{d=0}^{\infty}\frac{(2\boldsymbol{x}^T\boldsymbol{z})^d}{d!}e^{-\|\boldsymbol{x}\|^2-\|\boldsymbol{z}\|^2} = \sum_{d=0}^{\infty}\frac{2^d e^{-\|\boldsymbol{x}\|^2-\|\boldsymbol{z}\|^2}}{d!}\left(\boldsymbol{x}^T\boldsymbol{z}\right)^d,$$

which is a combination of polynomial kernels; therefore, the feature map corresponding to the RBF kernel is

$$\varphi(\boldsymbol{x}) = \left(e^{-\gamma\|\boldsymbol{x}\|^2}\sqrt{\frac{(2\gamma)^d}{d!}\binom{d}{n_1,\ldots,n_D}}x_1^{n_1}\cdots x_D^{n_D}\right)_{d\in\{0,1,2,\ldots\},\, n_i\geq 0,\, \sum_{i=1}^{D} n_i = d}.$$

Note that the RBF kernel is a function of distance – it "weighs" more similar examples more strongly. We could interpret it as an extended version of the k-nearest neighbor algorithm, one which considers all examples, each weighted by similarity.

For illustration, we plot RBF kernel values of three points $(0, -1)$, $(1, 1)$ and $(1, -1)$ with different values of γ:



Support Vector Machines

Let us return to a binary classification task. The perceptron algorithm guaranteed finding some separating hyperplane if it existed (but it could find quite a bad one).

We now consider finding the one with **maximum margin**.

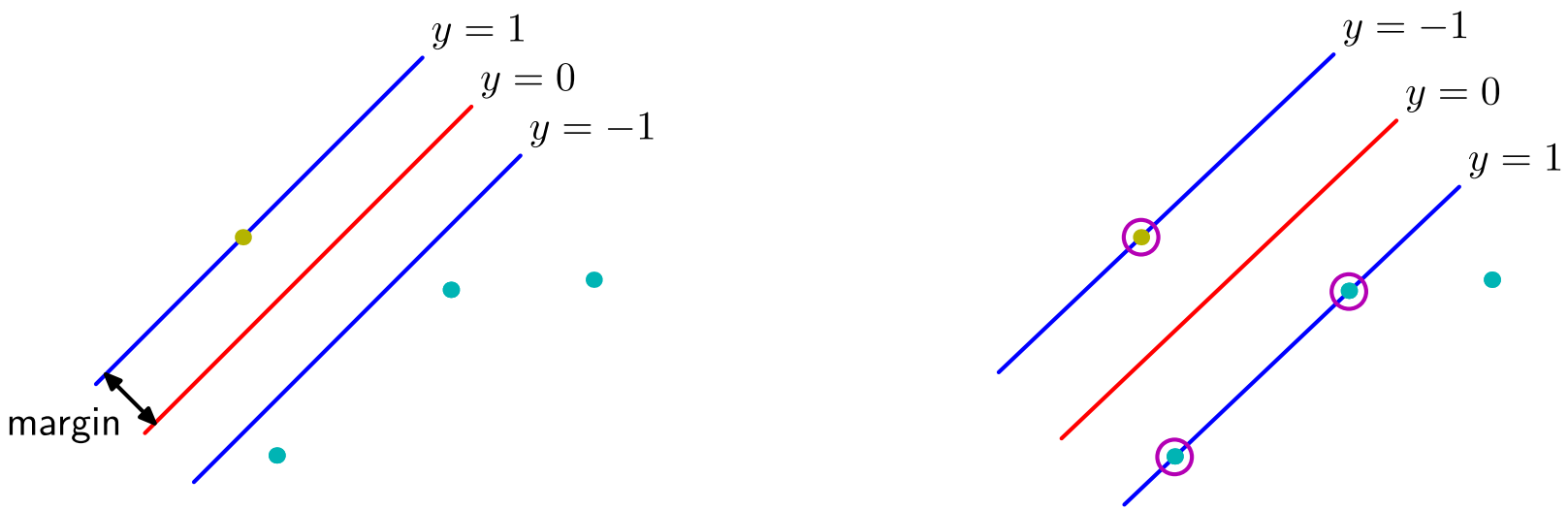


Figure 7.1 The margin is defined as the perpendicular distance between the decision boundary and the closest of the data points, as shown on the left figure. Maximizing the margin leads to a particular choice of decision boundary, as shown on the right. The location of this boundary is determined by a subset of the data points, known as support vectors, which are indicated by the circles.

Figure 7.1 of Pattern Recognition and Machine Learning.

Support Vector Machines

Assume we have a dataset $\boldsymbol{X} \in \mathbb{R}^{N \times D}$, $\boldsymbol{t} \in \{-1, 1\}^N$, a feature map φ and a model

$$y(\boldsymbol{x}) \stackrel{\text{def}}{=} \boldsymbol{\varphi}(\boldsymbol{x})^T \boldsymbol{w} + b.$$

We already know that the distance of a point $\boldsymbol{x}_i$ to the decision boundary is

$$\frac{|y(\boldsymbol{x}_i)|}{\|\boldsymbol{w}\|} \underset{\text{all } \boldsymbol{x}_i \text{ correctly}}{\overset{\text{assuming } y \text{ classifies}}{=\!=\!=}} \frac{t_i y(\boldsymbol{x}_i)}{\|\boldsymbol{w}\|}.$$

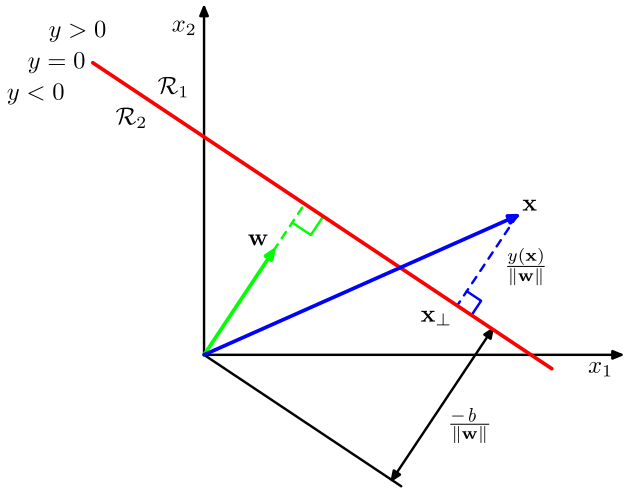


Figure 4.1 of Pattern Recognition and Machine Learning.

We therefore want to maximize

$$\arg\max_{\boldsymbol{w}, b} \frac{1}{\|\boldsymbol{w}\|} \min_i \left[t_i \left(\boldsymbol{\varphi}(\boldsymbol{x}_i)^T \boldsymbol{w} + b \right) \right].$$

However, this problem is difficult to optimize directly.

Because the model is invariant to multiplying $\boldsymbol{w}$ and b by a constant, we can decide that for the points closest to the decision boundary, it will hold that

$$t_i y(\boldsymbol{x}_i) = 1.$$

Then for all the points we will have $t_i y(\boldsymbol{x}_i) \geq 1$, and we can simplify

$$\arg\max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|} \min_i \left[t_i \left(\boldsymbol{\varphi}(\boldsymbol{x}_i)^T \boldsymbol{w} + b \right) \right].$$

to

$$\arg\min_{\boldsymbol{w},b} \frac{1}{2} \|\boldsymbol{w}\|^2 \text{ given that } t_i y(\boldsymbol{x}_i) \geq 1.$$

Constrained Optimization – Inequality Constraints

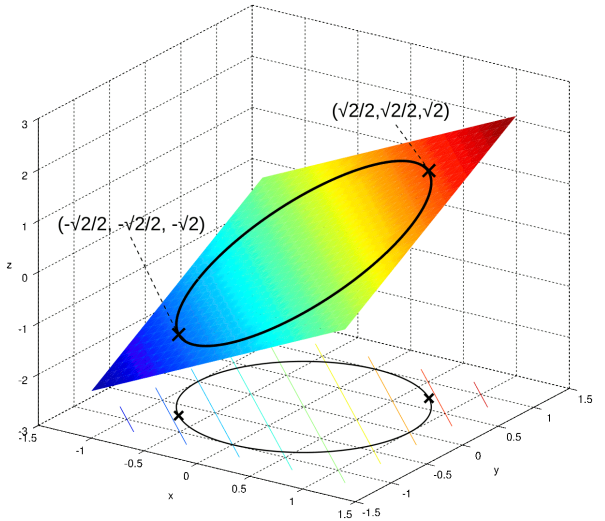
Given a function $f(\boldsymbol{x})$, we can find its minimum with respect to a vector $\boldsymbol{x} \in \mathbb{R}^D$, by investigating the critical points $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = 0$.

We can even incorporate constraints of form $g(\boldsymbol{x}) = 0$ by forming a Lagrangian

$$\mathcal{L}(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) - \lambda g(\boldsymbol{x})$$

and again investigating the critical points $\nabla_{\boldsymbol{x},\lambda} \mathcal{L}(\boldsymbol{x}, \lambda) = 0$.

We now describe how to include inequality constraints $g(\boldsymbol{x}) \geq 0$.



https://upload.wikimedia.org/wikipedia/commons/e/ed/Lagrange

Constrained Optimization – Inequality Constraints

Our goal is to find a minimum of $f(\boldsymbol{x})$ subject to a constraint $g(\boldsymbol{x}) \geq 0$.

We start by again forming a Lagrangian $f(\boldsymbol{x}) - \lambda g(\boldsymbol{x})$.

The optimum can either be attained for $g(\boldsymbol{x}) > 0$, when the constraint is said to be **inactive**, or for $g(\boldsymbol{x}) = 0$, when the constraint is said to be **active**. In the inactive case, the minimum is again a critical point of the Lagrangian with the condition $\lambda = 0$.

When the minimum is on a boundary, it corresponds to a critical point with $\lambda \neq 0$ – but note that this time the sign of the multiplier matters, because the minimum is attained only when the gradient of $f(\boldsymbol{x})$ is oriented **into** the region $g(\boldsymbol{x}) \geq 0$. We therefore require $\nabla f(\boldsymbol{x}) = \lambda \nabla g(\boldsymbol{x})$ for $\lambda > 0$.

In both cases, $\lambda g(\boldsymbol{x}) = 0$.

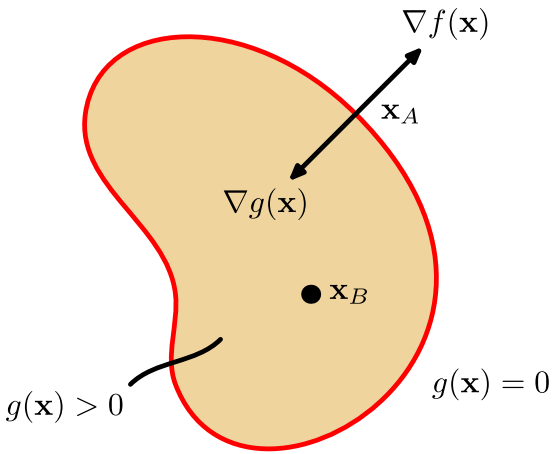
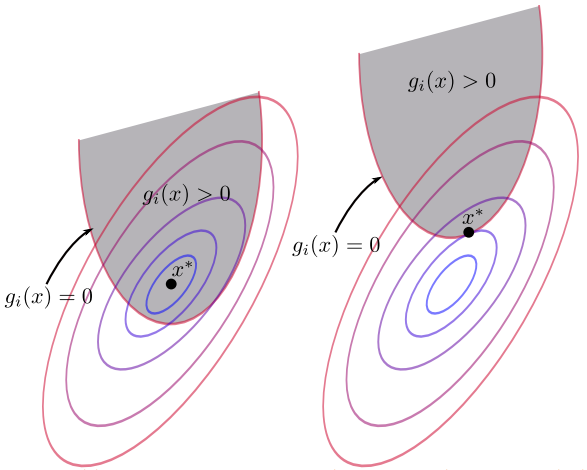


Figure E.3 of Pattern Recognition and Machine Learning.



https://upload.wikimedia.org/wikipedia/commons/5/5d/Inequ

Minimization – Inequality Constraint

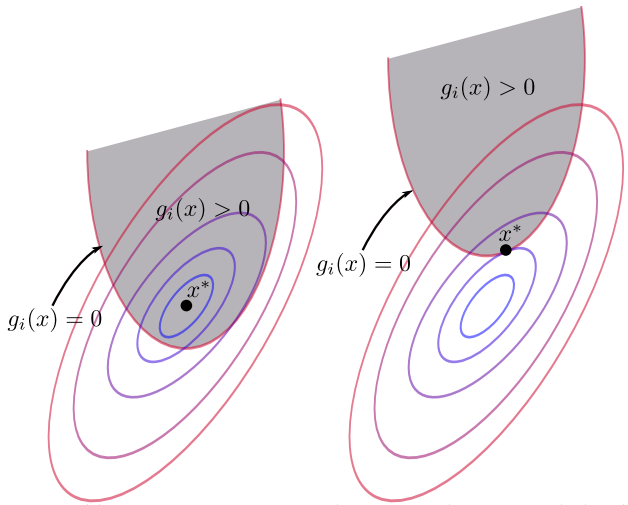
Let $f(\boldsymbol{x}) : \mathbb{R}^D \to \mathbb{R}$ be a function, which has a minimum in $\boldsymbol{x}$ subject to an inequality constraint $g(\boldsymbol{x}) \geq 0$. Assume that both f and g have continuous partial derivatives and that $\nabla_{\boldsymbol{x}} g(\boldsymbol{x}) \neq 0$.

Then there exists a $\lambda \in \mathbb{R}$, such that the **Lagrangian function**

$$\mathcal{L}(\boldsymbol{x}, \lambda) \stackrel{\text{def}}{=} f(\boldsymbol{x}) - \lambda g(\boldsymbol{x})$$

has zero gradient in $\boldsymbol{x}$ and the following conditions hold:

$$\begin{aligned} g(\boldsymbol{x}) &\geq 0, \\ \lambda &\geq 0, \\ \lambda g(\boldsymbol{x}) &= 0. \end{aligned}$$



https://upload.wikimedia.org/wikipedia/commons/5/5d/Inequalit

These conditions are known as **Karush-Kuhn-Tucker (KKT)** conditions.

Minimization – Inequality Constraint

It is easy to verify that if $\mathcal{L}(\boldsymbol{x}, \lambda)$ has a minimum in $\boldsymbol{x}$ and λ fulfills the KKT conditions $g(\boldsymbol{x}) \geq 0, \lambda \geq 0, \lambda g(\boldsymbol{x}) = 0$, the Lagrangian $\mathcal{L}$ has a **maximum** in λ subject to $\lambda \geq 0$:

- if $g(\boldsymbol{x}) = 0$, then $\mathcal{L}$ does not change when changing λ,
- if $g(\boldsymbol{x}) > 0$, then $\lambda = 0$ from the KKT conditions, which is a maximum of $\mathcal{L}$.

On the other hand, if $\mathcal{L}(\boldsymbol{x}, \lambda)$ has a minimum in $\boldsymbol{x}$, and a maximum in λ subject to $\lambda \geq 0$, all the KKT conditions must hold:

- if $g(\boldsymbol{x}) < 0$, then increasing λ would increase $\mathcal{L}$,
- if $g(\boldsymbol{x}) > 0$, then decreasing λ increases $\mathcal{L}$, so $\lambda = 0$.

Maximizing Given $f(\boldsymbol{x})$

If we instead want to find the constrained maximum of $f(\boldsymbol{x})$, we can search for the minimum of $-f(\boldsymbol{x})$, which results in the Lagrangian $f(\boldsymbol{x}) + \lambda g(\boldsymbol{x})$, which we *minimize* with respect to λ.

Necessary and Sufficient KKT Conditions

The KKT conditions are necessary conditions for a minimum (resp. a maximum).

However, it can be proven that in the following settings, the conditions are also **sufficient**:

- if the objective to optimize is a *convex* function (resp. *concave* for maximization) with respect to $\boldsymbol{x}$;
- the inequality constraints are continuously differentiable convex functions;
- the equality constraints are affine functions (linear functions with an offset).

Therefore, if the above holds and if we find $\boldsymbol{x}$ and λ such that:

- $\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}) = 0$,
- either
  - $g(\boldsymbol{x}) \geq 0$, $\lambda \geq 0$, $\lambda g(\boldsymbol{x}) = 0$,
  - or $\lambda \geq 0$ and $\mathcal{L}$ has a *maximum* in λ,

then $\boldsymbol{x}$ is a minimum of the function $f(\boldsymbol{x})$ subject to an inequality constraint $g(\boldsymbol{x}) \geq 0$.

It is easy to verify that these conditions hold for the SVM optimization problem.

In order to solve the constrained problem of

$$\arg\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 \text{ given that } t_i y(\boldsymbol{x}_i) \geq 1,$$

we write the Lagrangian with multipliers $\boldsymbol{a} = (a_1, \ldots, a_N)$ as

$$\mathcal{L} = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_i a_i \big(t_i y(\boldsymbol{x}_i) - 1\big).$$

Setting the derivatives with respect to $\boldsymbol{w}$ and b to zero, we get

$$\boldsymbol{w} = \sum_i a_i t_i \varphi(\boldsymbol{x}_i),$$
$$0 = \sum_i a_i t_i.$$

Substituting these to the Lagrangian, we want to maximize

$$\mathcal{L} = \sum_i a_i - \frac{1}{2} \sum_i \sum_j a_i a_j t_i t_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

with respect to a_i subject to the constraints $a_i \geq 0$ and $\sum_i a_i t_i = 0$, using the kernel $K(\boldsymbol{x}, \boldsymbol{z}) = \varphi(\boldsymbol{x})^T \varphi(\boldsymbol{z})$.

The solution will fulfill the KKT conditions, meaning that

$$a_i \geq 0, \qquad t_i y(\boldsymbol{x}_i) - 1 \geq 0, \qquad a_i \big(t_i y(\boldsymbol{x}_i) - 1\big) = 0.$$

Therefore, either a point $\boldsymbol{x}_i$ is on a boundary, or $a_i = 0$. Given that the prediction for $\boldsymbol{x}$ is $y(\boldsymbol{x}) = \sum_i a_i t_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b$, we only need to keep the training points $\boldsymbol{x}_i$ that are on the boundary, the so-called **support vectors**. Therefore, even though SVM is a nonparametric model, it needs to store only a subset of the training data.

Support Vector Machines

The dual formulation allows us to use nonlinear kernels.

Figure 7.2 Example of synthetic data from two classes in two dimensions showing contours of constant $y(\mathbf{x})$ obtained from a support vector machine having a Gaussian kernel function. Also shown are the decision boundary, the margin boundaries, and the support vectors.

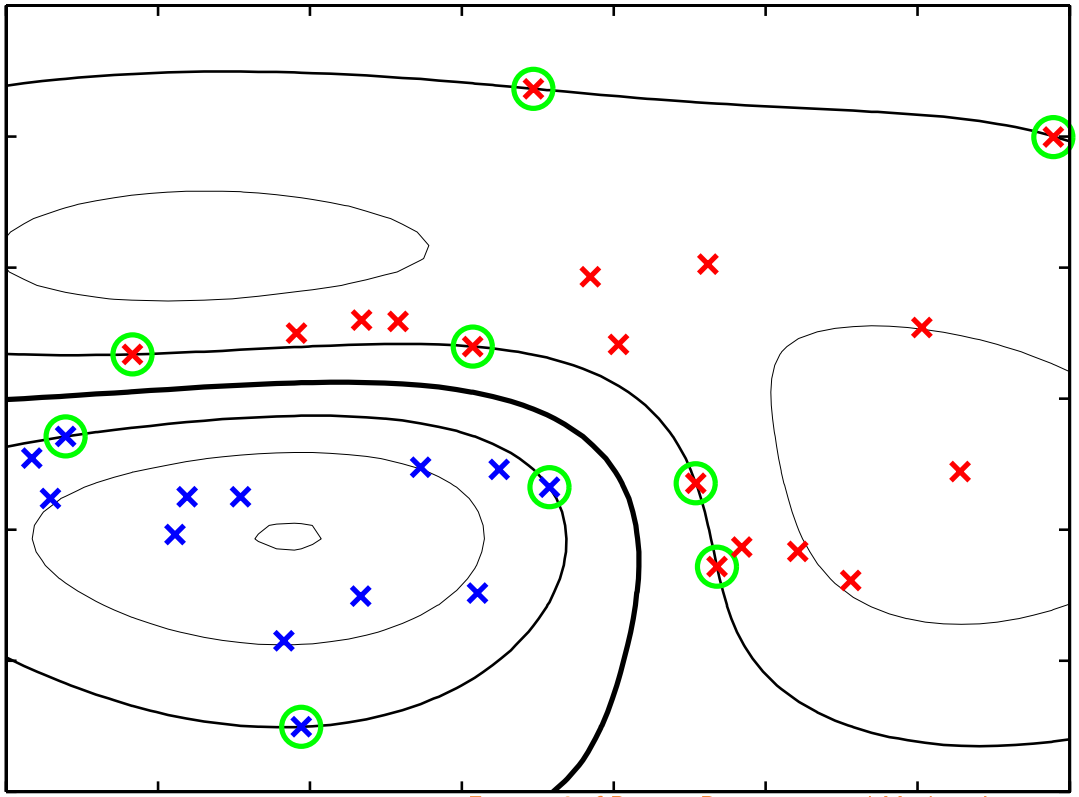


Figure 7.2 of Pattern Recognition and Machine Learning.