

# Derivation of Softmax, Support Vector Machines

Milan Straka

 November 18, 2019



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



unless otherwise stated

# Lagrange Multipliers – Equality Constraints

Given a function  $J(\mathbf{x})$ , we can find a maximum with respect to a vector  $\mathbf{x} \in \mathbb{R}^d$ , by investigating the critical points  $\nabla_{\mathbf{x}} J(\mathbf{x}) = 0$ .

Consider now finding maximum subject to a constraint  $g(\mathbf{x}) = 0$ .

- Note that  $\nabla_{\mathbf{x}} g(\mathbf{x})$  is orthogonal to the surface of the constraint, because if  $\mathbf{x}$  and a nearby point  $\mathbf{x} + \boldsymbol{\varepsilon}$  lie on the surface, from the Taylor expansion  $g(\mathbf{x} + \boldsymbol{\varepsilon}) \approx g(\mathbf{x}) + \boldsymbol{\varepsilon}^T \nabla_{\mathbf{x}} g(\mathbf{x})$  we get  $\boldsymbol{\varepsilon}^T \nabla_{\mathbf{x}} g(\mathbf{x}) \approx 0$ .
- In the sought maximum,  $\nabla_{\mathbf{x}} f(\mathbf{x})$  must also be orthogonal to the constraint surface (or else moving in the direction of the derivative would increase the value).
- Therefore, there must exist  $\lambda$  such that  $\nabla_{\mathbf{x}} f + \lambda \nabla_{\mathbf{x}} g = 0$ .

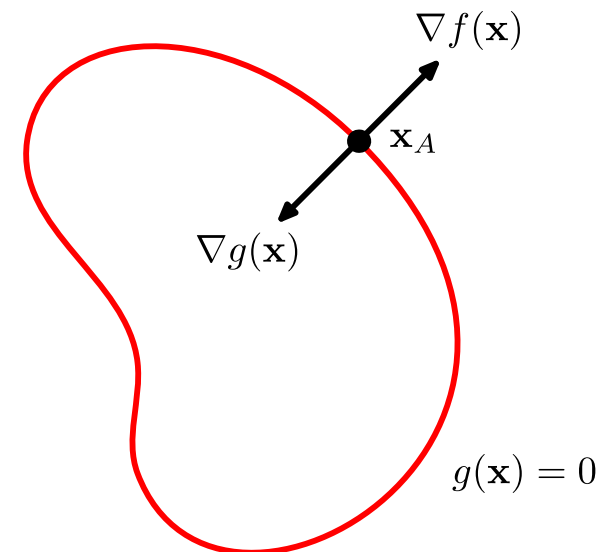


Figure E.1 of Pattern Recognition and Machine Learning.

# Lagrange Multipliers – Equality Constraints

We therefore introduce the *Lagrangian function*

$$L(\mathbf{x}, \lambda) \stackrel{\text{def}}{=} f(\mathbf{x}) + \lambda g(\mathbf{x}).$$

We can then find the maximum under the constraint by inspecting critical points of  $L(\mathbf{x}, \lambda)$  with respect to both  $\mathbf{x}$  and  $\lambda$ :

- $\frac{\partial L}{\partial \lambda} = 0$  leads to  $g(\mathbf{x}) = 0$ ;
- $\frac{\partial L}{\partial \mathbf{x}} = 0$  is the previously derived  $\nabla_{\mathbf{x}} f + \lambda \nabla_{\mathbf{x}} g = 0$ .

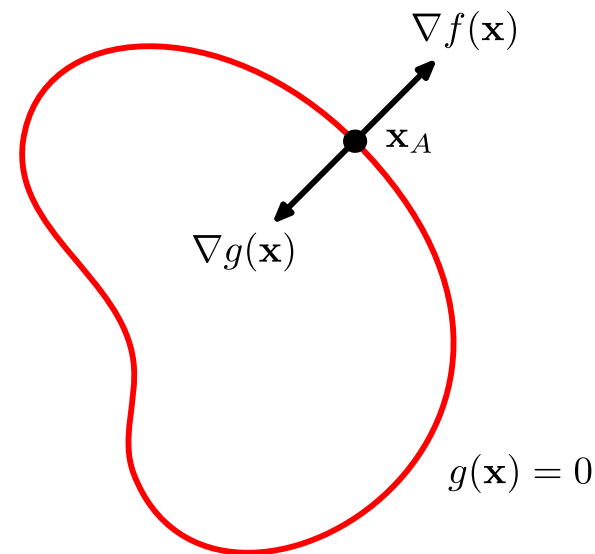
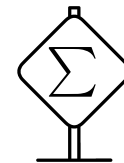


Figure E.1 of *Pattern Recognition and Machine Learning*.

Many optimization techniques depend on minimizing a function  $J(\mathbf{w})$  with respect to a vector  $\mathbf{w} \in \mathbb{R}^d$ , by investigating the critical points  $\nabla_{\mathbf{w}} J(\mathbf{w}) = 0$ .



A function of a function,  $J[f]$ , is known as a **functional**, for example entropy  $H[\cdot]$ .

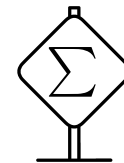
Similarly to partial derivatives, we can take **functional derivatives** of a functional  $J[f]$  with respect to individual values  $f(\mathbf{x})$  for all points  $\mathbf{x}$ . The functional derivative of  $J$  with respect to a function  $f$  in a point  $\mathbf{x}$  is denoted as

$$\frac{\partial}{\partial f(\mathbf{x})} J.$$

For this class, we will use only the following theorem, which states that for all differentiable functions  $f$  and differentiable functions  $g(f(\mathbf{x}), \mathbf{x})$  with continuous derivatives, it holds that

$$\frac{\partial}{\partial f(\mathbf{x})} \int g(f(\mathbf{x}), \mathbf{x}) d\mathbf{x} = \frac{\partial}{\partial y} g(y, \mathbf{x}).$$

An intuitive view is to think about  $f(\mathbf{x})$  as a vector of uncountably many elements (for every value  $\mathbf{x}$ ). In this interpretation the result is analogous to computing partial derivatives of a vector  $\mathbf{w} \in \mathbb{R}^d$ :



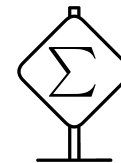
$$\frac{\partial}{\partial w_i} \sum_j g(w_j, \mathbf{x}) = \frac{\partial}{\partial w_i} g(w_i, \mathbf{x}).$$

$$\frac{\partial}{\partial f(\mathbf{x})} \int g(f(\mathbf{x}), \mathbf{x}) d\mathbf{x} = \frac{\partial}{\partial y} g(y, \mathbf{x}).$$

# Continuous Distribution with Maximum Entropy

What distribution over  $\mathbb{R}$  maximizes entropy  $H[p] = -\mathbb{E}_x \log p(x)$ ?

For continuous values, the entropy is an integral  $H[p] = -\int p(x) \log p(x) dx$ .



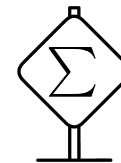
We cannot just maximize  $H$  with respect to a function  $p$ , because:

- the result might not be a probability distribution – we need to add a constraint that  $\int p(x) dx = 1$ ;
- the problem is unspecified because a distribution can be shifted without changing entropy – we add a constraint  $\mathbb{E}[x] = \mu$ ;
- because entropy increases as variance increases, we ask which distribution with a *fixed* variance  $\sigma^2$  has maximum entropy – adding a constraint  $\text{Var}(x) = \sigma^2$ .

# Function with Maximum Entropy

Lagrangian of all the constraints and the entropy function is

$$L(p; \mu, \sigma^2) = \lambda_1 \left( \int p(x) dx - 1 \right) + \lambda_2 (\mathbb{E}[x] - \mu) + \lambda_3 (\text{Var}(x) - \sigma^2) + H[p].$$



By expanding all definitions to integrals, we get

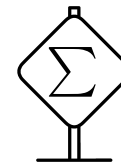
$$L(p; \mu, \sigma^2) = \int \left( \lambda_1 p(x) + \lambda_2 p(x)x + \lambda_3 p(x)(x - \mu)^2 - p(x) \log p(x) \right) dx - \lambda_1 - \mu\lambda_2 - \sigma^2\lambda_3.$$

The functional derivative of  $L$  is:

$$\frac{\partial}{\partial p(x)} L(p; \mu, \sigma^2) = \lambda_1 + \lambda_2 x + \lambda_3 (x - \mu)^2 - 1 - \log p(x) = 0.$$

Rearrangint the functional derivative of  $L$ :

$$\frac{\partial}{\partial p(x)} L(p; \mu, \sigma^2) = \lambda_1 + \lambda_2 x + \lambda_3 (x - \mu)^2 - 1 - \log p(x) = 0.$$



we obtain

$$p(x) = \exp \left( \lambda_1 + \lambda_2 x + \lambda_3 (x - \mu)^2 - 1 \right).$$

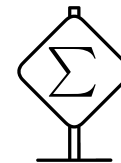
We can verify that setting  $\lambda_1 = 1 - \log \sigma \sqrt{2\pi}$ ,  $\lambda_2 = 0$  and  $\lambda_3 = -1/(2\sigma^2)$  fulfils all the constraints, arriving at

$$p(x) = \mathcal{N}(x; \mu, \sigma^2).$$



# Derivation of Softmax using Maximum Entropy

Let  $\mathbb{X} = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$  be training data of a  $K$ -class classification, with  $\mathbf{x}_i \in \mathbb{R}^D$  and  $t_i \in \{1, 2, \dots, K\}$ .



We want to model it using a function  $\pi : \mathbb{R}^D \rightarrow \mathbb{R}^K$  so that  $\pi(\mathbf{x})$  gives a distribution of classes for input  $\mathbf{x}$ .

We impose the following conditions on  $\pi$ :

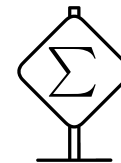
- $$\pi(\mathbf{x})_j \geq 0$$

- $$\sum_{j=1}^K \pi(\mathbf{x})_j = 1$$

- $$\forall_{k \in \{1, 2, \dots, D\}}, \forall_{j \in \{1, 2, \dots, K\}} : \sum_{i=1}^N \pi(\mathbf{x}_i)_j \mathbf{x}_{i,k} = \sum_{i=1}^N [t_i == j] \mathbf{x}_{i,k}$$

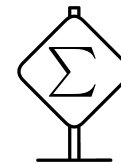
There are many such  $\pi$ , one particularly bad is

$$\pi(\mathbf{x}) = \begin{cases} t_i & \text{if there exists } i : \mathbf{x}_i = \mathbf{x}, \\ 0 & \text{otherwise.} \end{cases}$$



Therefore, we want to find a more general  $\pi$  – we will aim for one with maximum entropy.

# Derivation of Softmax using Maximum Entropy



We therefore want to maximize  $-\sum_{i=1}^N \sum_{j=1}^K \pi(\mathbf{x}_i)_j \log(\pi(\mathbf{x}_i)_j)$  given

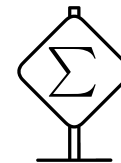
- $\pi(\mathbf{x})_j \geq 0$ ,
- $\sum_{i=j}^K \pi(\mathbf{x})_j = 1$ ,
- $\forall k \in \{1, \dots, D\}, \forall j \in \{1, \dots, K\} : \sum_{i=1}^N \pi(\mathbf{x}_i)_j \mathbf{x}_{i,k} = \sum_{i=1}^N [t_i == j] \mathbf{x}_{i,k}$ .

We therefore form a Lagrangian

$$\begin{aligned} L = & \sum_{k=1}^D \sum_{j=1}^K \lambda_{k,j} \left( \sum_{i=1}^N \pi(\mathbf{x}_i)_j \mathbf{x}_{i,k} - [t_i == j] \mathbf{x}_{i,k} \right) \\ & - \sum_{i=1}^N \beta_i \left( \sum_{j=1}^K \pi(\mathbf{x}_i)_j - 1 \right) \\ & - \sum_{i=1}^N \sum_{j=1}^K \pi(\mathbf{x}_i)_j \log(\pi(\mathbf{x}_i)_j) \end{aligned}$$

We now compute partial derivatives of the Lagrangian, notably the values

$$\frac{\partial}{\partial \pi(\mathbf{x}_i)_j} L.$$



We arrive at

$$\frac{\partial}{\partial \pi(\mathbf{x}_i)_j} L = \lambda_{*,j} \mathbf{x}_i + \beta_i - \log(\pi(\mathbf{x}_i)_j) - 1.$$

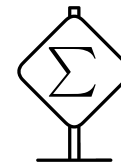
Setting the Lagrangian to zero, we get  $\lambda_{*,j} \mathbf{x}_i + \beta_i - \log(\pi(\mathbf{x}_i)_j) - 1 = 0$ , which we rewrite to

$$\pi(\mathbf{x}_i)_j = e^{\lambda_{*,j} \mathbf{x}_i + \beta_i - 1}.$$

Such a form guarantees  $\pi(\mathbf{x}_i)_j > 0$ , which we did not include in the conditions.

# Derivation of Softmax using Maximum Entropy

In order to find out the  $\beta_i$  values, we turn to the constraint



$$\sum_j \pi(\mathbf{x}_i)_j = \sum_j e^{\lambda_{*,j} \mathbf{x}_i + \beta_i - 1} = 1,$$

from which we get

$$e^{\beta_i} = \frac{1}{\sum_j e^{\lambda_{*,j} \mathbf{x}_i - 1}},$$

yielding

$$\pi(\mathbf{x}_i)_j = \frac{e^{\lambda_{*,j} \mathbf{x}_i}}{\sum_k e^{\lambda_{*,k} \mathbf{x}_i}}.$$

Consider linear regression with cubic features

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

The SGD update for linear regression is then

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha (t - \mathbf{w}^T \varphi(\mathbf{x})) \varphi(\mathbf{x}).$$

# Kernel Linear Regression

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

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

By induction,  $\mathbf{w} = \mathbf{0} = \sum_i 0 \cdot \varphi(\mathbf{x}_i)$ , and assuming  $\mathbf{w} = \sum_i \beta_i \cdot \varphi(\mathbf{x}_i)$ , after a SGD update we get

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} + \alpha \sum_i (t_i - \mathbf{w}^T \varphi(\mathbf{x}_i)) \varphi(\mathbf{x}_i) \\ &= \sum_i \left( \beta_i + \alpha (t_i - \mathbf{w}^T \varphi(\mathbf{x}_i)) \right) \varphi(\mathbf{x}_i). \end{aligned}$$

A individual update is  $\beta_i \leftarrow \beta_i + \alpha (t_i - \mathbf{w}^T \varphi(\mathbf{x}_i))$ , and substituting for  $\mathbf{w}$  we get

$$\beta_i \leftarrow \beta_i + \alpha \left( t_i - \sum_j \beta_j \varphi(\mathbf{x}_j)^T \varphi(\mathbf{x}_i) \right).$$

We can formulate the alternative linear regression algorithm (it would be called a *dual formulation*):

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

- Set  $\beta_i \leftarrow 0$
- Compute all values  $K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j)$
- Repeat until convergence
  - Update the coordinates, either according to a full gradient update:
    - $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \alpha(\mathbf{t} - \mathbf{K}\boldsymbol{\beta})$
  - or alternatively use single-batch SGD, arriving at:
    - for  $i$  in random permutation of  $\{1, \dots, N\}$ :
      - $\beta_i \leftarrow \beta_i + \alpha \left( t_i - \sum_j \beta_j K(\mathbf{x}_i, \mathbf{x}_j) \right)$

In vector notation, we can write  $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \alpha(\mathbf{t} - \mathbf{K}\boldsymbol{\beta})$ .

The predictions are then performed by computing  $y(\mathbf{x}) = \mathbf{w}^T \varphi(\mathbf{x}) = \sum_i \beta_i \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x})$ .



# Kernel Trick

A single SGD update of all  $\beta_i$  then takes  $\mathcal{O}(N^2)$ , given that we can evaluate scalar dot product of  $\varphi(\mathbf{x}_j)^T \varphi(\mathbf{x}_i)$  quickly.

$$\begin{aligned}
 \varphi(\mathbf{x})^T \varphi(\mathbf{z}) &= 1 + \sum_i \mathbf{x}_i z_i + \sum_{i,j} \mathbf{x}_i \mathbf{x}_j z_i z_j + \sum_{i,j,k} \mathbf{x}_i \mathbf{x}_j \mathbf{x}_k z_i z_j z_k \\
 &= 1 + \sum_i \mathbf{x}_i z_i + \left( \sum_i \mathbf{x}_i z_i \right)^2 + \left( \sum_i \mathbf{x}_i z_i \right)^3 \\
 &= 1 + \mathbf{x}^T \mathbf{z} + (\mathbf{x}^T \mathbf{z})^2 + (\mathbf{x}^T \mathbf{z})^3.
 \end{aligned}$$

We define a *kernel* corresponding to a feature map  $\varphi$  as a function

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

There is quite a lot of theory behind kernel construction. The most often used kernels are:

- polynomial kernel or degree  $d$

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

which corresponds to a feature map generating all combinations of up to  $d$  input features;

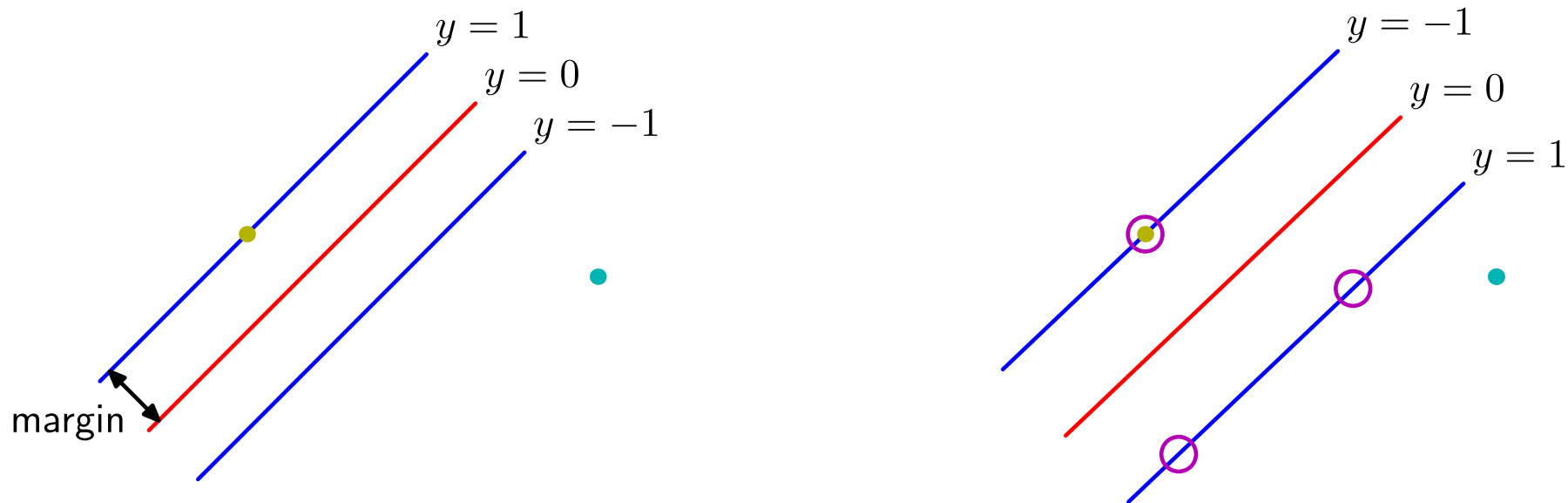
- Gaussian (or RBF) kernel

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

corresponding to a scalar product in an infinite-dimensional space (it is in a sense a combination of polynomial kernels of all degrees).

# Support Vector Machines

Let us return to a binary classification task. The perceptron algorithm guaranteed finding some separating hyperplane if it existed; 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  $\mathbf{X} \in \mathbb{R}^{N \times D}$ ,  $\mathbf{t} \in \{-1, 1\}^N$ , feature map  $\varphi$  and model

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

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

$$\frac{|y(\mathbf{x}_i)|}{\|\mathbf{w}\|} = \frac{t_i y(\mathbf{x}_i)}{\|\mathbf{w}\|}.$$

We therefore want to maximize

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

However, this problem is difficult to optimize directly.

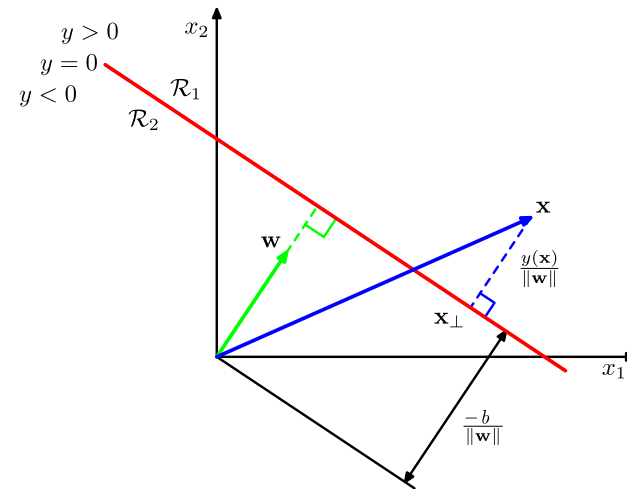


Figure 4.1 of Pattern Recognition and Machine Learning.

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

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

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

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

to

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

# Lagrange Multipliers – Inequality Constraints

Given a function  $J(\mathbf{x})$ , we can find a maximum with respect to a vector  $\mathbf{x} \in \mathbb{R}^d$ , by investigating the critical points  $\nabla_{\mathbf{x}} J(\mathbf{x}) = \mathbf{0}$ .

We even know how to incorporate constraints of form  $g(\mathbf{x}) = 0$ .

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

The optimum can either be attained for  $g(\mathbf{x}) > 0$ , when the constraint is said to be *inactive*, or for  $g(\mathbf{x}) = 0$ , when the constraint is said to be *active*.

In the inactive case, the maximum is again a critical point of the Lagrangian, with  $\lambda = 0$ . When maximum is on boundary, it corresponds to a critical point with  $\lambda \neq 0$  – but note that this time the sign of the multiplier matters, because maximum is attained only when gradient of  $f(\mathbf{x})$  is oriented away from the region  $g(\mathbf{x}) \geq 0$ . We therefore require  $\nabla f(\mathbf{x}) = -\lambda \nabla g(\mathbf{x})$  for  $\lambda > 0$ .

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

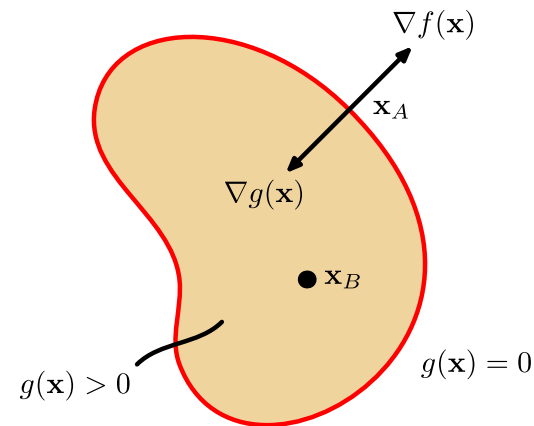


Figure E.3 of Pattern Recognition and Machine Learning.

# Karush-Khun-Tucker Conditions

Therefore, the solution to a maximization problem of  $f(\mathbf{x})$  subject to  $g(\mathbf{x}) \geq 0$  can be found by inspecting all points where the derivation of the Lagrangian is zero, subject to the following conditions:

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

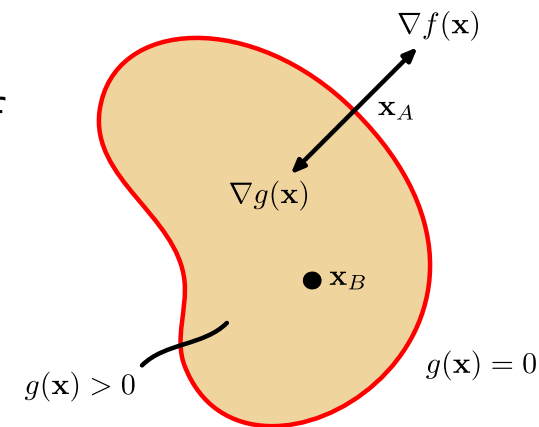


Figure E.3 of Pattern Recognition and Machine Learning.

## Necessary and Sufficient KKT Conditions

The above conditions are necessary conditions for a minimum. However, it can be proven that in the following settings, the conditions are also **sufficient**:

- if the objective to optimize is a *convex* function,
- the inequality constraints are continuously differentiable convex functions,
- the equality constraints are affine functions (linear functions with an offset).

In order to solve the constrained problem of

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

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

$$L = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_i a_i [t_i y(\mathbf{x}_i) - 1].$$

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

$$\begin{aligned} \mathbf{w} &= \sum_i a_i t_i \varphi(\mathbf{x}_i) \\ 0 &= \sum_i a_i t_i \end{aligned}$$



Substituting these to the Lagrangian, we get

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

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

The solution of this Lagrangian will fulfil the KKT conditions, meaning that

$$\begin{aligned} a_i &\geq 0 \\ t_i y(\mathbf{x}_i) - 1 &\geq 0 \\ a_i (t_i y(\mathbf{x}_i) - 1) &= 0. \end{aligned}$$

Therefore, either a point is on a boundary, or  $a_i = 0$ . Given that the predictions for point  $\mathbf{x}$  are given by  $y(\mathbf{x}) = \sum a_i t_i K(\mathbf{x}, \mathbf{x}_i) + b$ , we need to keep only the points on the boundary, the so-called **support vectors**.

The dual formulation allows us to use non-linear 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.

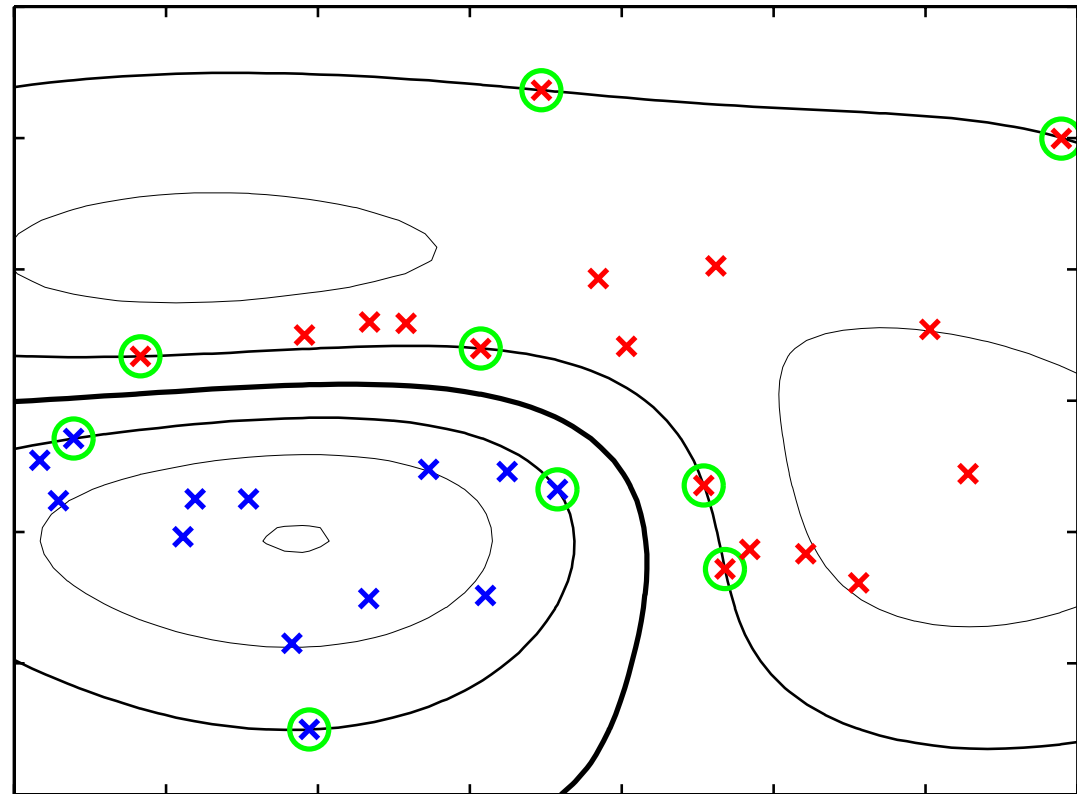


Figure 7.2 of Pattern Recognition and Machine Learning.