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

Lagrange Multipliers – Equality Constraints

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

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

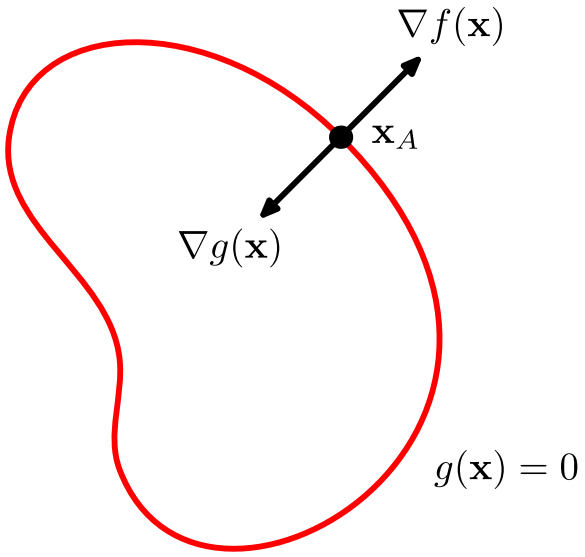

Figure E.1 of Pattern Recognition and Machine Learning.

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

Lagrange Multipliers – Equality Constraints

We therefore introduce the *Lagrangian function*

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

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

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

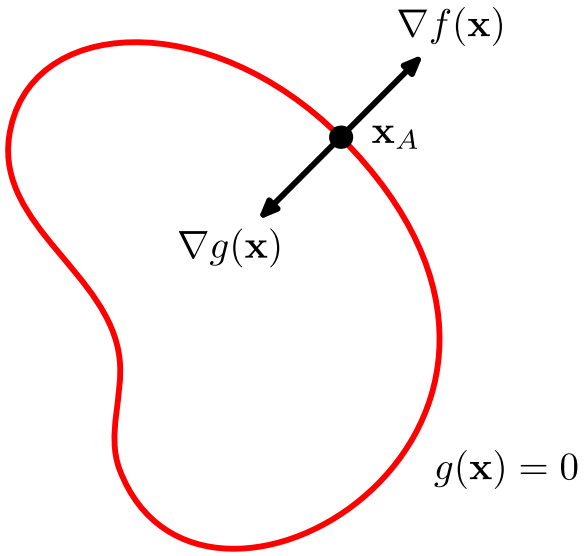

Figure E.1 of Pattern Recognition and Machine Learning.

Calculus of Variations

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

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 $f(\boldsymbol{x})$ for all points $\boldsymbol{x}$. The functional derivative of J with respect to a function f in a point $\boldsymbol{x}$ is denoted as

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

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

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

Calculus of Variations

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

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

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

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 $H[p] = -\int p(x) \log p(x) \, \mathrm{d}x$.

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 $\int p(x) \, \mathrm{d}x = 1$;
- the problem is unspecified because a distribution can be shifted without changing entropy – we add a constraing $\mathbb{E}[x] = \mu$;
- because entropy increases as variance increases, we ask which distribution with a *fixed* variance σ^2 has maximum entropy – adding a constraing $\mathrm{Var}(x) = \sigma^2$.

Function with Maximum Entropy

Lagrangian of all the constraings and the entropy function is

$$L(p;\mu,\sigma^2) = \lambda_1\Big(\int p(x)\,\mathrm{d}x - 1\Big) + \lambda_2\big(\mathbb{E}[x]-\mu\big) + \lambda_3\big(\operatorname{Var}(x)-\sigma^2\big) + H[p].$$

By expanding all definitions to integrals, we get

$$L(p;\mu,\sigma^2) = \int \Big(\lambda_1 p(x) + \lambda_2 p(x) x \lambda_3 p(x)(x-\mu)^2 - p(x)\log p(x)\Big)\,\mathrm{d}x - \\ - \lambda_1 - \mu\lambda_2 - \sigma^2\lambda_3.$$

The functional derivative of L is:

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

Function with Maximum Entropy

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

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

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

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

We impose the following conditions on π:

- $$\pi(\boldsymbol{x})_j \geq 0$$
- $$\sum_{j=1}^{K} \pi(\boldsymbol{x})_j = 1$$
- $$\forall_{k \in \{1,2,\ldots,D\}}, \forall_{j \in \{1,2,\ldots,K\}} : \sum_{i=1}^{N} \pi(\boldsymbol{x}_i)_j x_{i,k} = \sum_{i=1}^{N} \Big[t_i == j \Big] x_{i,k}$$

Derivation of Softmax using Maximum Entropy

There are many such π, one particularly bad is

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

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

Derivation of Softmax using Maximum Entropy

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

- $\pi(\boldsymbol{x})_j \geq 0$,
- $\sum_{i=j}^{K}\pi(\boldsymbol{x})_j = 1$,
- $\forall_{k\in\{1,\ldots,D\}}, \forall_{j\in\{1,\ldots,K\}} : \sum_{i=1}^{N}\pi(\boldsymbol{x}_i)_j x_{i,k} = \sum_{i=1}^{N}\left[t_i == j\right]x_{i,k}$.

We therefore form a Lagrangian

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

Derivation of Softmax using Maximum Entropy

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

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

We arrive at

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

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

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

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

Derivation of Softmax using Maximum Entropy

In order to find out the β_i values, we turn to the constraint

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

from which we get

$$e^{\beta_i} = \frac{1}{\sum_j e^{\boldsymbol{\lambda}_{*,j}\boldsymbol{x}_i - 1}},$$

yielding

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

Kernel Linear Regression

Consider linear regression with cubic features

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

The SGD update for linear regression is then

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

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 $\boldsymbol{w}$ as a linear combination of the input features $\varphi(\boldsymbol{x}_i)$.

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

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

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

$$\beta_i \leftarrow \beta_i + \alpha\Big(t_i - \sum\nolimits_j \beta_j \varphi(\boldsymbol{x}_j)^T \varphi(\boldsymbol{x}_i)\Big).$$

Kernel Linear Regression

We can formulate the alternative linear regression algorithm (it would be called a *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 $\alpha \in \mathbb{R}^+$.

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

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

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

Kernel Trick

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

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

Kernels

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

- polynomial kernel or degree d

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

  which corresponds to a feature map generating all combinations of up to d input features;
- Gaussian (or RBF) kernel

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

  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*.

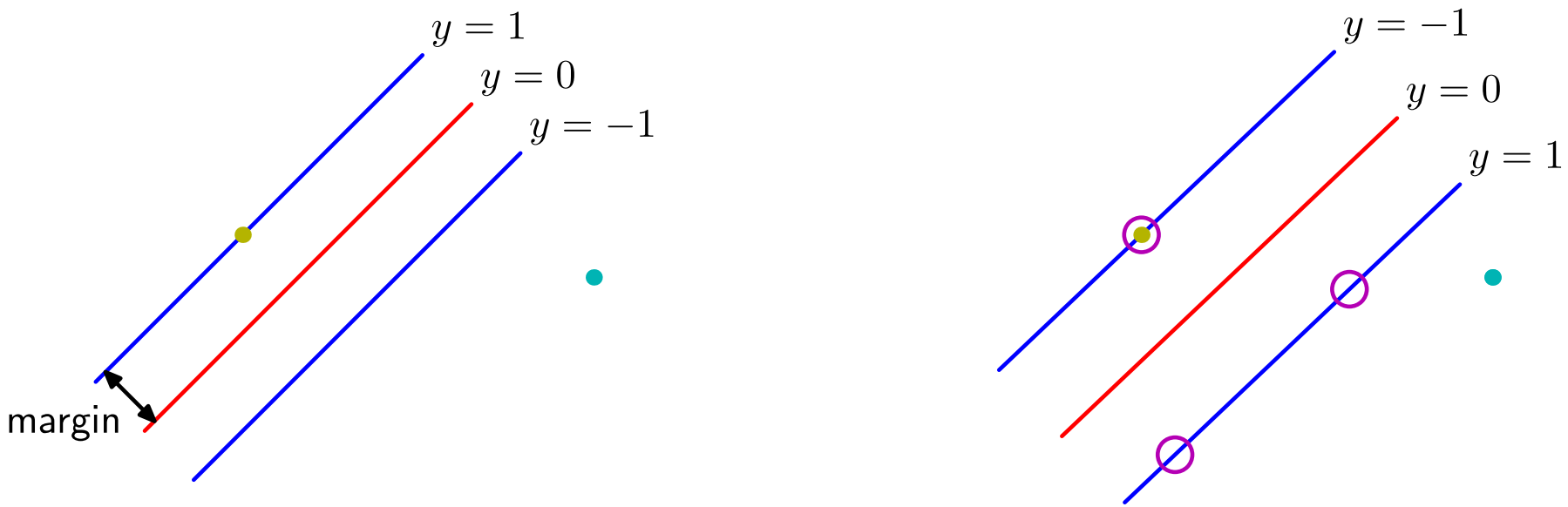

Figure 7.1 The margin is defined as the perpendicular distance between the decision boundary and the closest of the data points, as shown on the left figure. Maximizing the margin leads to a particular choice of decision boundary, as shown on the right. The location of this boundary is determined by 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 $\boldsymbol{X} \in \mathbb{R}^{N \times D}$, $\boldsymbol{t} \in \{-1, 1\}^N$, feature map φ and 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

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

We therefore want to maximize

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

However, this problem is difficult to optimize directly.

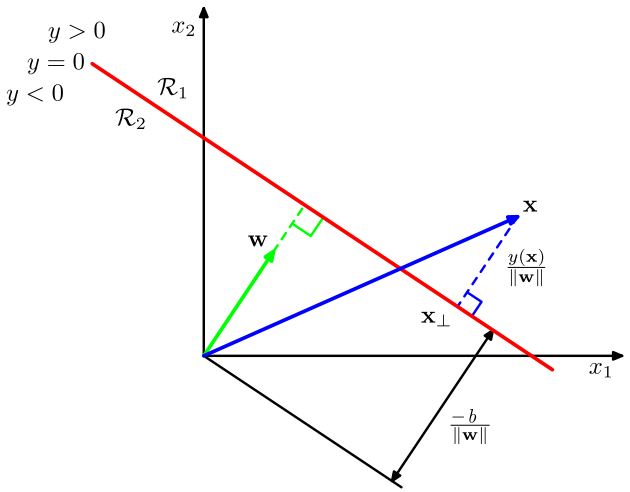

Figure 4.1 of Pattern Recognition and Machine Learning.

Support Vector Machines

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

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

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

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

to

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

Lagrange Multipliers – Inequality Constraints

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

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

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

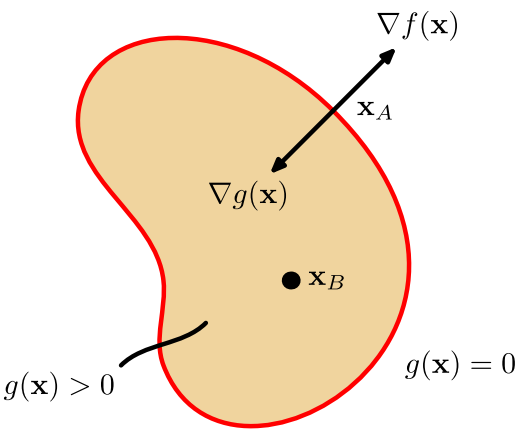

Figure E.3 of Pattern Recognition and Machine Learning.

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 sayd to be *active*.

In the inactive case, the maximum is again a critical point of the Langrangian, 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 $f(\boldsymbol{x})$ is oriented away from 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, $\lambda g(\boldsymbol{x}) = 0$.

Karush-Khun-Tucker Conditions

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

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

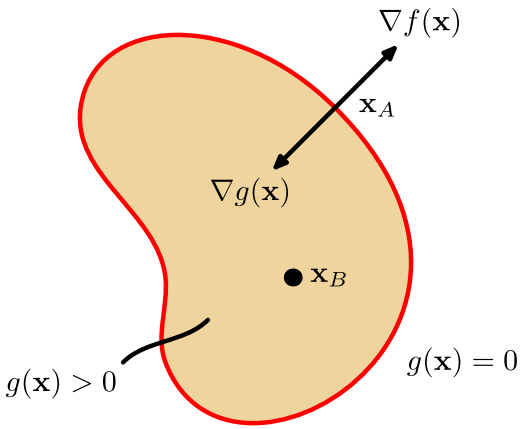

Figure E.3 of Pattern Recognition and Machine Learning.

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 constraings are continuously differentiable convex functions,
- the equality constraints are affine functions (linear functions with an offset).

Support Vector Machines

In order to solve the constrained problem of

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

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

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

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

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

$$0 = \sum_i a_i t_i$$

Support Vector Machines

Substituting these to the Lagrangian, we get

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

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

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

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

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

Support Vector Machines

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.

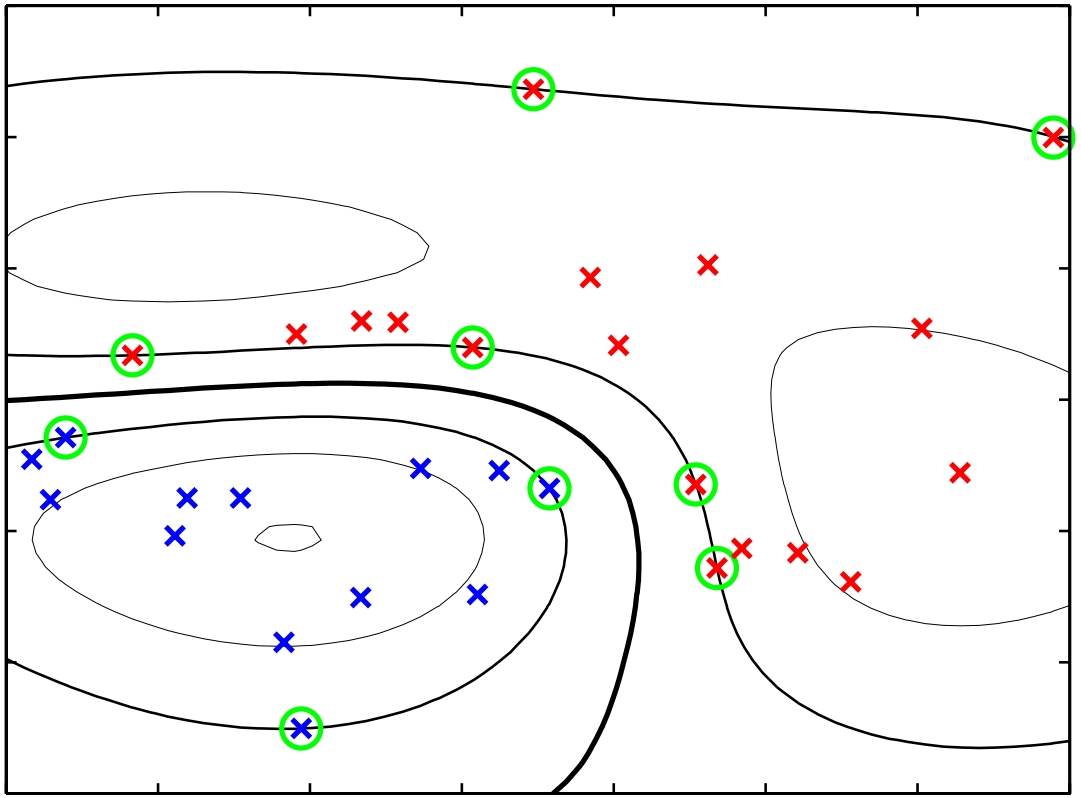

Figure 7.2 of Pattern Recognition and Machine Learning.