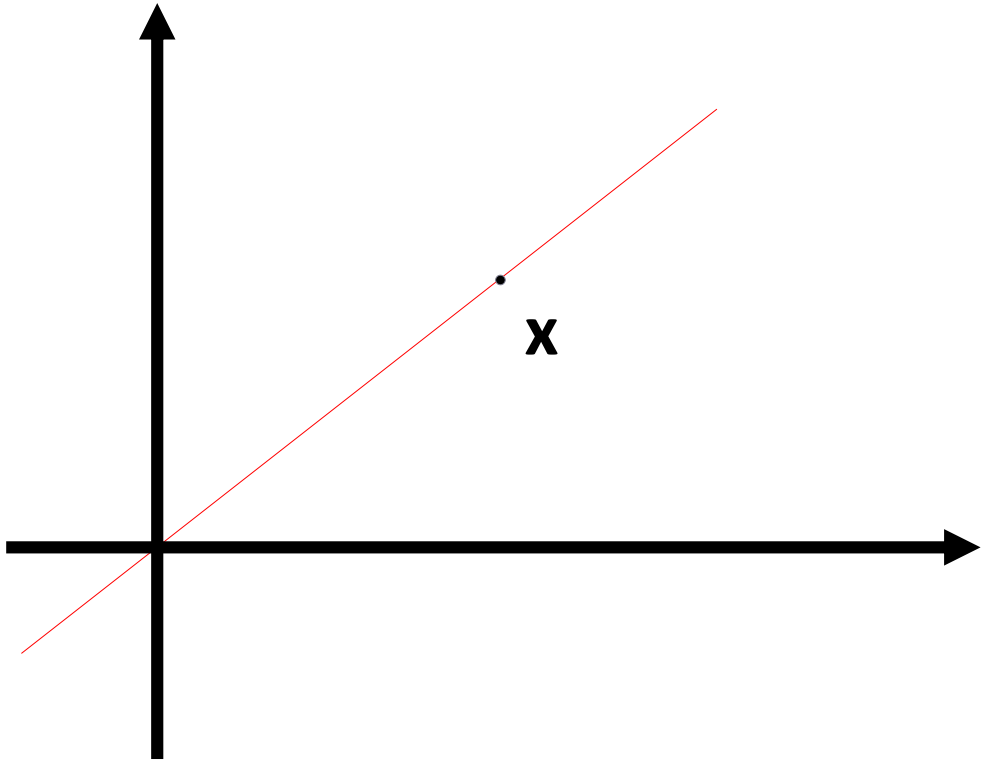
AARHUS
UNIVERSITET

Optimization and Data Analytics

Alexandros Iosifidis
@
Aarhus University, Department of Engineering

Linear Decision Functions

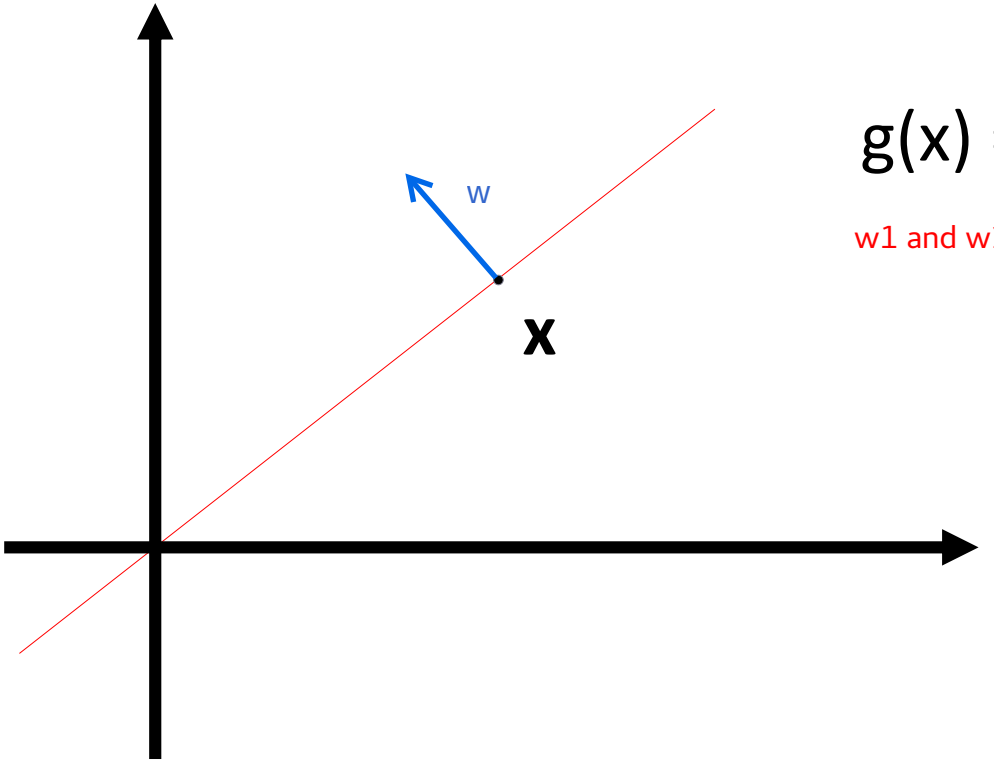
How can we define a linear decision function passing through the origin?



Demo

Linear Decision Functions

How can we define a linear decision function passing through the origin?

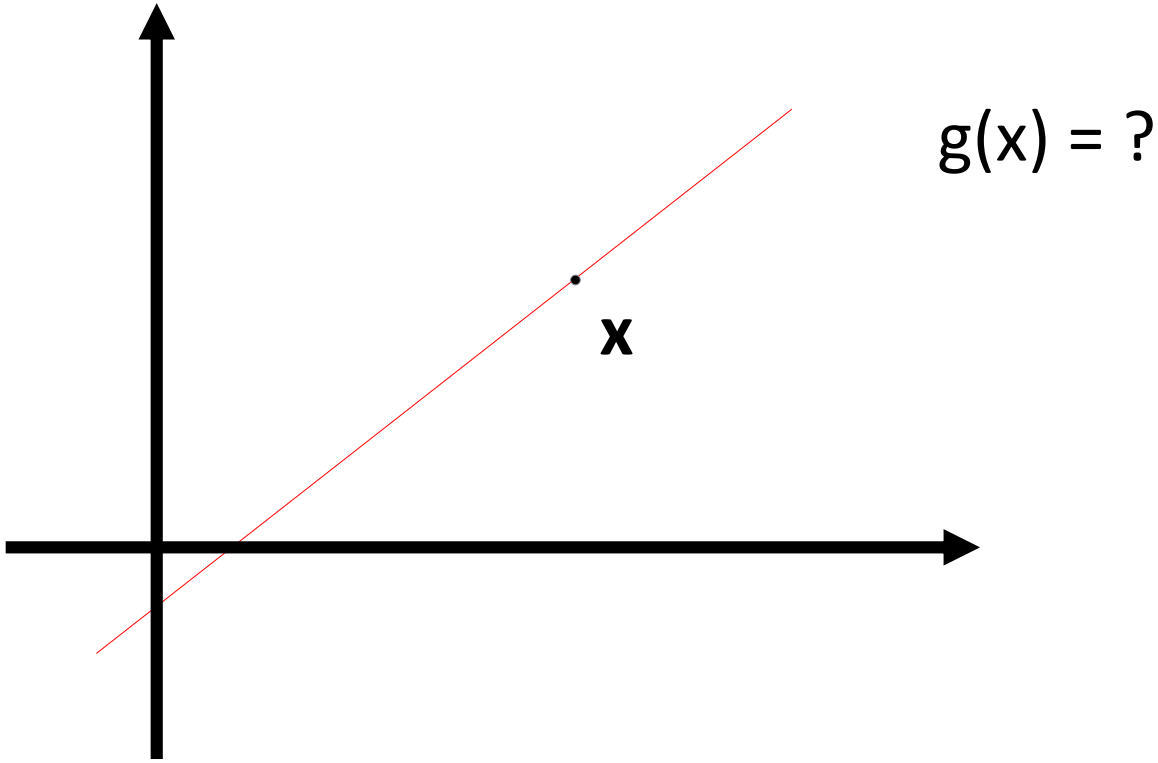


$$g(x) = w_1x_1 + w_2x_2 = 0$$

w1 and w2 describe the slope

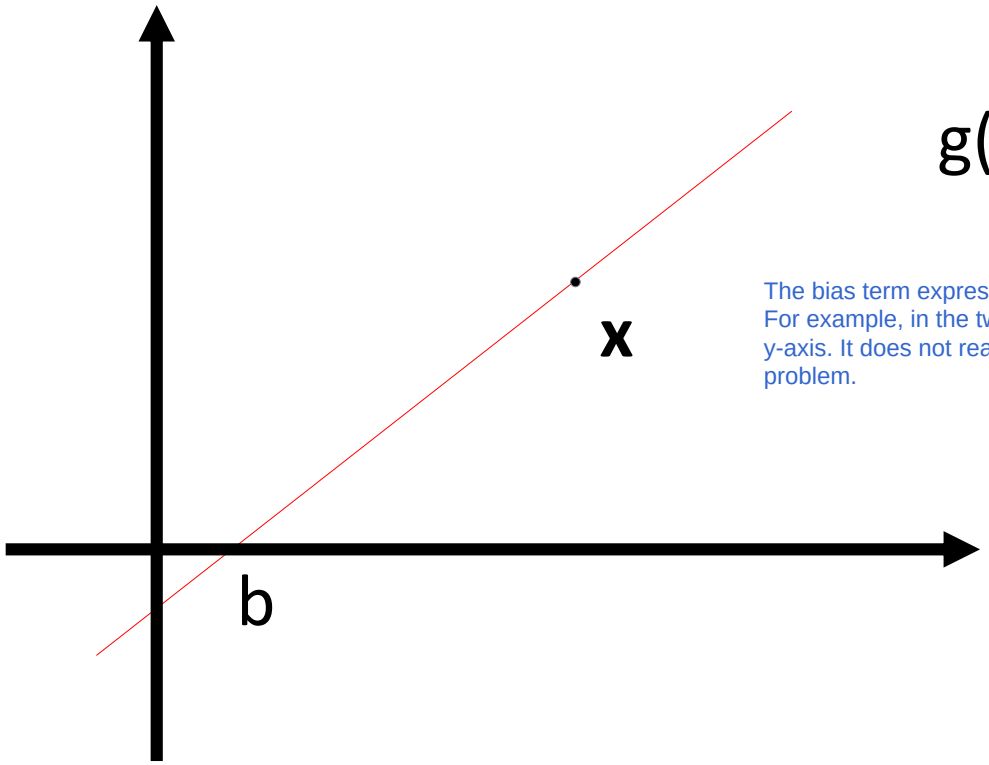
Linear Decision Functions

If the linear decision function is not passing through the origin?



Linear Decision Functions

If the linear decision function is not passing through the origin?



$$g(x) = w_1x_1 + w_2x_2 + b = 0$$

The bias term expresses the displacement of the hyperplane from the origin in on of the axes. For example, in the two-dimensional case, b could express the displace on the x-axis or the y-axis. It does not really matter. What matters is that the definition is consistent throughout the problem.

Linear Decision Functions

In the general case, a linear decision function in a D-dimensional feature space can be expressed as

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = \sum_{d=1}^{D} w_d x_d + w_0$$

where **w** $\epsilon\mathbb{R}^D$ is called weight vector and w_0 is called bias.

w expresses the orientation of the decision function (also called discriminant hyperplane) and w_0 expresses the displacement of the hyperplane from the origin.

Linear Decision Functions

The equation g(x) = 0 defines the decision function (hyperplane) between two classes.

If two vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ are points of the hyperplane, then

$$g(\mathbf{x_1}) = \mathbf{w}^T\mathbf{x}_1 + w_0 = 0 = \mathbf{w}^T\mathbf{x}_2 + w_0 = g(\mathbf{x_2})$$

which leads to $$\mathbf{w}^T(\mathbf{x}_1 - \mathbf{x}_2) = 0$$

The above means that **w** is orthogonal to any vector lying in the hyperplane.

Linear Decision Functions

The decision function g(**x**) divides the feature space in two regions:

1. If g(**x**) > 0, then we say that x belongs to class c1
2. If g(**x**) < 0, then we say that x belongs to class c2
3. If g(**x**) = 0, x is on the hyperplane. Usually, we decide which class it belongs to before hand,

Due to the above (binary) form of the discriminant function, two-class classification problems are usually called as binary classification problems.

We will see how to define multi-class classifiers later.

Linear Decision Functions

The value of g(·) for vector **x**, is a measurement of the distance of **x** from the hyperplane. Why?

Linear Decision Functions

In order to get a more compact definition of the decision function, we use an augmented definition of **w** and **x**

$$\tilde{\mathbf{w}} \leftarrow [w_0 \ \mathbf{w}^T]^T$$

$$\tilde{\mathbf{x}} \leftarrow [1 \ \mathbf{x}^T]^T$$

Then,

$$g(\mathbf{x}) = \tilde{\mathbf{w}}^T \tilde{\mathbf{x}}$$

We will use **w** and **x** to denote their augmented versions (if not stated otherwise)

Linear Decision Functions

The value of g(·) for vector **x**, is a measurement of the distance of **x** from the hyperplane.

We express **x** as a summation of two vectors

x_p is parallel to line

$$\mathbf{x} = \mathbf{x}_p + r\frac{1}{\|\mathbf{w}\|_2}\mathbf{w}$$

Since $\mathbf{w}^T\mathbf{x}_p = 0$, we have

$$\begin{aligned} g(\mathbf{x}) &= \mathbf{w}^T\left(\mathbf{x}_p + r\frac{1}{\|\mathbf{w}\|_2}\mathbf{w}\right) \\ &= r\frac{1}{\|\mathbf{w}\|_2}\mathbf{w}^T\mathbf{w} = r\frac{\|\mathbf{w}\|_2^2}{\|\mathbf{w}\|_2} = r\|\mathbf{w}\|_2 \end{aligned}$$

Since we typically set w to be unit vector then r is the distance

Linear Decision Functions

Given a set of N samples, each represented by a vector $\mathbf{x}_i \in \mathbb{R}^D$, and the corresponding labels $l_i = \{-1,1\}$ we want to optimize the parameters of $g(\cdot)$ in order to define a discriminant hyperplane discriminating the two classes.

We will do this optimization of the parameters of $g(\cdot)$ without setting assumptions on the distributions of each class.

The only assumption we will make is that the decision function corresponds to a linear discriminator (a hyperplane).

Linearly separable case

In order to simplify notations, we will use a trick related to the values of the discriminant function for samples of different classes:

- $g(\mathbf{w}, \mathbf{x}_i) > 0$, then we decide that $\mathbf{x}_i$ belongs to class c_1,
- $g(\mathbf{w}, \mathbf{x}_i) < 0$, then we decide that $\mathbf{x}_i$ belongs to class c_2,
- $g(\mathbf{w}, \mathbf{x}_i) = 0$, then we say that $\mathbf{x}_i$ belongs to ambiguous region.

Thus, for a set of parameters **w*** classifying correctly all training vectors will have:

Notice that we use one equation to express the three classifcation rules above!

$$f(\mathbf{w}^*, \mathbf{x}_i) = l_i \, g(\mathbf{w}^*, \mathbf{x}_i) = l_i \, \mathbf{w}^{*T} \mathbf{x}_i \geq 0, \; i = 1, \ldots, N$$

Linearly separable case

Let us assume that we have already a weight vector **w** and let us denote by X the set of training vectors that are miss-classified by using **w**.

If f(x) is positive then x is classified correctly.
If it is negative then x is misclassified

Then, we can define the error of the classifier using **w** as

$$\mathcal{J}_p(\mathbf{w}) = \sum_{x_i \in \mathcal{X}} -f(\mathbf{w}, \mathbf{x}_i) = \sum_{x_i \in \mathcal{X}} -l_i \, \mathbf{w}^T \mathbf{x}_i$$

The error of the classifier is the summation of the misclassified samples. Basically, we are summing the distances of the misclassified samples and the line. The error is always positive!

If $J_p(\mathbf{w}^*) = 0$, then $\mathbf{w}^*$ can classify all training samples and is a solution to an optimization problem minimizing J_p.

Either Chi is empty or there are some samples that are on the hyperplane

J_p is called Perceptron criterion function.

Linearly separable case

Linearly separable case: The is a gap between the two classe so there is a linear classifier that can classify all the samples perfectly.

In order to optimize $J_p(\mathbf{w})$ we calculate the derivative w.r.t. $\mathbf{w}$

$$\nabla \mathcal{J}_p = \sum_{x_i \in \mathcal{X}} (-l_i \, \mathbf{x}_i)$$

Only defined for the misclassified samples

Having the derivative, allows us to update $\mathbf{w}$ to a new one which (usually) gives a lower criterion value

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta(t) \nabla \mathcal{J}_p = \mathbf{w}(t) + \eta(t) \sum_{x_i \in \mathcal{X}} l_i \, \mathbf{x}_i$$

What this update corresponds to?

It corresponds to the gradient-based classifier like the steepest descent.

Linearly separable case

Algorithm 5: Batch Perceptron

1: Initialize the parameters $\mathbf{w}$, $\eta(\cdot)$, $t = 0$
2: **Do** $t \leftarrow t + 1$
3: $\quad \mathcal{X} = \{\}$
4: $\quad i = 0$
5: $\quad$ **Do** $i \leftarrow i + 1$
6: $\quad\quad$ **if** $f(\mathbf{w}, \mathbf{x}_i) < 0$, **then** $\mathbf{x}_i \rightarrow \mathcal{X}$
7: $\quad$ **until** $i = N$
8: $\quad \mathbf{w} \leftarrow \mathbf{w} + \eta(t) \sum_{x_i \in \mathcal{X}} l_i \mathbf{x}_i$
9: **until** $\mathcal{X} = \{\}$

Linearly separable case

Variant 1

Algorithm 6: Single-sample Perceptron

1: Initialize the parameters $\mathbf{w}$, $\eta(\cdot)$, θ, $t = 0$
2: **Do** $t \leftarrow t + 1$
3: Select (randomly) a vector $\mathbf{x}_i$
4: **if** $f(\mathbf{w}, \mathbf{x}_i) < 0$, **then** $\mathbf{w} \leftarrow \mathbf{w} + \eta(t) l_i \mathbf{x}_i$
5: **until** all samples are correctly classified

Line 3: We don't want w to change in a pattern!

Linearly separable case

Variant 2

Algorithm 7: Single-sample Perceptron with margin

1: Initialize the parameters $\mathbf{w}$, $\eta(\cdot)$, θ, $t = 0$
2: **Do** $t \leftarrow t + 1$
3: Select (randomly) a vector $\mathbf{x}_i$
4: **if** $f(\mathbf{w}, \mathbf{x}_i) + b < 0$, **then** $\mathbf{w} \leftarrow \mathbf{w} + \eta(t) l_i \mathbf{x}_i$
5: **until** $g(\mathbf{x}_i) + b \leq 0$ for all i

The difference is in line 4. Samples inside the margin are considered to be

Notice that the margin b is hyperparameter of the algorithm. We find a good b based on grid search.

Linearly separable case

Other criteria that can be used for calculating **w** is

$$\mathcal{J}_q(\mathbf{w}) = \sum_{\mathbf{x}\in\mathcal{X}} (l_i \mathbf{w}^T \mathbf{x})^2$$

quadratic error, always positive

We scale the contribution of each sample using

$$\mathcal{J}_r(\mathbf{w}) = \frac{1}{2} \sum_{\mathbf{x}\in\mathcal{X}} \frac{(l_i \mathbf{w}^T \mathbf{x} - b)^2}{\|\mathbf{x}\|_2^2}$$

What is the derivative of Jr? Solve it as an exercise!

Which are the differences between the two?

Linearly separable case

Using the criterion

$$\mathcal{J}_r(\mathbf{w}) = \frac{1}{2}\sum_{\mathbf{x}\in\mathcal{X}} \frac{(l_i \mathbf{w}^T \mathbf{x} - b)^2}{\|\mathbf{x}\|_2^2}$$

the derivative w.r.t. $\mathbf{w}$ is

$$\nabla \mathcal{J}_r = \sum_{\mathbf{x}_i\in\mathcal{X}} \frac{\mathbf{w}^T \mathbf{x}_i - l_i b}{\|\mathbf{x}_i\|_2^2}\mathbf{x}_i$$

Linearly separable case

Algorithm 8: Batch Relaxation with Margin

Initialize the parameters $\mathbf{w}$, $\eta(\cdot)$, $t = 0$
Do $t \leftarrow t + 1$
  $\mathcal{X} = \{\}$
  $i = 0$
  Do $i \leftarrow i + 1$
    if $f(\mathbf{w}, \mathbf{x}_i) + b < 0$, **then** $\mathbf{x}_i \rightarrow \mathcal{X}$
  until $i = N$
  $\mathbf{w} \leftarrow \mathbf{w} - \eta(t) \sum_{\mathbf{x}_i \in \mathcal{X}} \frac{\mathbf{w}^T \mathbf{x}_i - l_i b}{\|\mathbf{x}_i\|_2^2} \mathbf{x}_i$
until $\mathcal{X} = \{\}$

Linearly separable case

Algorithm 9: Single-sample Relaxation with Margin

1: Initialize the parameters $\mathbf{w}$, $\eta(\cdot)$, θ, $t = 0$
2: **Do** $t \leftarrow t + 1$
3: Select (randomly) a vector $\mathbf{x}_i$
4: **if** $f(\mathbf{w}, \mathbf{x}_i) + b < 0$, **then** $\mathbf{w} \leftarrow \mathbf{w} - \eta(t)\frac{\mathbf{w}^T\mathbf{x}_i - l_i b}{\|\mathbf{x}_i\|_2^2}\mathbf{x}_i$
5: **until** all samples are correctly classified

Linearly separable case

Discussion:

1. How to select the learning rate η?
   A high value of eta will classifier. If eta is 0.1 then the algorithm will converge.
2. What is better, batch or sample-based update?
   There is no good answer. Typically, we in-between method called mini-batche updates..
3. Which algorithm will give the best performance?
   Experience will give you which algorithms to use and you will test the algorithm on your data.

Linearly separable case

Use of Linear Programming

All algorithms above solve simultaneously linear inequalities. Such optimization problems can be solved by formulating suitable linear programming problems, where the linear inequalities are used as constraints.

Linearly separable case

Use of Linear Programming

All algorithms above solve simultaneously linear inequalities. Such optimization problems can be solved by formulating suitable linear programming problems, where the linear inequalities are used as constraints.

Linear programming problem that can be solved by the Simplex Method:

$$z = \mathbf{a}^T \mathbf{u},$$

with constraints

$$\begin{aligned} \mathbf{A}\mathbf{u} &\geq \beta \\ \mathbf{u} &\geq 0, \end{aligned}$$

where $\mathbf{a} \in \mathbb{R}^m$ is a *cost vector*, $\boldsymbol{\mu} \in \mathbb{R}^l$ and $\mathbf{A} \in \mathbb{R}^{l \times m}$

Linearly separable case

Linear programming problem that can be solved by the Simplex Method:

$$z = \mathbf{a}^T \mathbf{u},$$

with constraints

$$\begin{aligned} \mathbf{A}\mathbf{u} &\geq \beta \\ \mathbf{u} &\geq 0, \end{aligned}$$

where $\mathbf{a} \in \mathbb{R}^m$ is a *cost vector*, $\boldsymbol{\mu} \in \mathbb{R}^l$ and $\mathbf{A} \in \mathbb{R}^{l \times m}$

Thus, can we use $\mathbf{w} = \mathbf{u}$?

Linearly separable case

Linear programming problem that can be solved by the Simplex Method:

$$z = \mathbf{a}^T \mathbf{u},$$

with constraints

$$\begin{aligned} \mathbf{A}\mathbf{u} &\geq \beta \\ \mathbf{u} &\geq 0, \end{aligned}$$

where $\mathbf{a} \in \mathbb{R}^m$ is a *cost vector*, $\boldsymbol{\mu} \in \mathbb{R}^l$ and $\mathbf{A} \in \mathbb{R}^{l \times m}$

Thus, can we use **w** = **u?** No, because **w** is not a positive vector. We can form u using $\mathbf{w}^+$ and $\mathbf{w}^-$

$$\mathbf{w} = \mathbf{w}^+ - \mathbf{w}^- \rightarrow \begin{cases} \mathbf{w}^+ = \frac{1}{2}(|\mathbf{w}| + \mathbf{w}) \\ \mathbf{w}^- = \frac{1}{2}(|\mathbf{w}| - \mathbf{w}) \end{cases}$$

Linearly separable case

When the problem is linearly separable, w satisfies

$$l_i \mathbf{w}^T \mathbf{x}_i \geq b_i > 0, \; i = 1, \ldots, N$$

Using a variable $\tau \geq 0$, the above can be written as

$$l_i \mathbf{w}^T \mathbf{x}_i + \tau \geq b_i > 0, \; i = 1, \ldots, N$$

Thus, if $\tau = 0$, the two set of constraints are the same.

Linearly separable case

When the problem is linearly separable, w satisfies

$$l_i \mathbf{w}^T \mathbf{x}_i \geq b_i > 0, \; i = 1, \ldots, N$$

Using a variable $\tau \geq 0$, the above can be written as

$$l_i \mathbf{w}^T \mathbf{x}_i + \tau \geq b_i > 0, \; i = 1, \ldots, N$$

Thus, if $\tau = 0$, the two set of constraints are the same.

This can be expressed using the following optimization criterion

$$\text{Minimize } \tau \text{ such that } l_i \mathbf{w}^T \mathbf{x}_i + \tau \geq b_i \text{ and } \tau \geq 0$$

Linearly separable case

The optimization criterion

$$\text{Minimize } \tau \text{ such that } l_i \mathbf{w}^T \mathbf{x}_i + \tau \geq b_i \text{ and } \tau \geq 0$$

can be solved by the Simplex Method when it is formulated as follows

$$\text{Minimize: } z = \mathbf{a}^T \mathbf{u},$$

subject to the constraints:

$$\mathbf{A}\mathbf{u} \geq \beta$$

$$\mathbf{u} \geq 0$$

Linearly separable case

In the above we use the following variables

$$\mathbf{a} = \begin{bmatrix} 0_D \\ 0_D \\ 1 \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} \mathbf{w}^+ \\ \mathbf{w}^- \\ \tau \end{bmatrix}, \quad \beta = \mathbf{b}, \quad \mathbf{A} = \begin{bmatrix} l_1\mathbf{x}_1^T & -l_1\mathbf{x}_1^T & 1 \\ l_2\mathbf{x}_2^T & -l_2\mathbf{x}_2^T & 1 \\ \vdots & \vdots & \vdots \\ l_N\mathbf{x}_N^T & -l_N\mathbf{x}_N^T & 1 \end{bmatrix}$$

b=margin

In the case where the problem is linearly-separable, the optimal value of τ is zero, and the solution (optimal weight vector) is given by the values of $\mathbf{w}^+$ and $\mathbf{w}^-$

Linearly separable case

Another criterion that can be optimized using Linear Programming is

$$\mathcal{J}'_p(\mathbf{w}) = \sum_{x_i \in \mathcal{X}'} (b_i - l_i \, \mathbf{w}^T \mathbf{x}_i)$$

margin for each sample x_i

where now $\mathbf{x}_i \in \mathcal{X}'(\mathbf{w})$ if $l_i \mathbf{w}^T \mathbf{x}_i \leq b_i$

Which is a variant of the Perceptron criterion

$$\mathcal{J}_p(\mathbf{w}) = \sum_{x_i \in \mathcal{X}} -l_i \, \mathbf{w}^T \mathbf{x}_i$$

Linearly separable case

Since the criterion is not defined on the entire set of N samples, we introduce N artificial variables and we solve for

$$\text{Minimize: } \alpha = \sum_{i=1}^{N} \tau_i,$$

subject to the constraints:

$$\tau_i \geq b_i - l_i \mathbf{w}^T \mathbf{x}_i$$

$$\tau_i \geq 0.$$

For any fixed value of $\mathbf{w}$, the minimum value of z is equal to $\mathcal{J}'_p(\mathbf{w})$. This is due to the constraints leading to $\tau_i = \max(0, b_i - l_i \mathbf{w}^T \mathbf{x}_i)$.

Linearly separable case

Minimization of $\mathcal{J}'_p(\mathbf{w})$ is equivalent to the following linear programming optimization problem:

$$\text{Minimize: } z = \mathbf{a}^T\mathbf{u},$$

subject to the constraints:

$$\mathbf{A}\mathbf{u} \geq \beta$$

$$\mathbf{u} \geq 0$$

where

This block expresses the b_i's

$$\mathbf{a} = \begin{bmatrix} 0_D \\ 0_D \\ 1_N \end{bmatrix}, \ \mathbf{u} = \begin{bmatrix} \mathbf{w}^+ \\ \mathbf{w}^- \\ \boldsymbol{\tau} \end{bmatrix}, \ \beta = \mathbf{b}, \ \mathbf{A} = \begin{bmatrix} l_1\mathbf{x}_1^T & -l_1\mathbf{x}_1^T & 1 & 0 & \dots & 0 \\ l_2\mathbf{x}_2^T & -l_2\mathbf{x}_2^T & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ l_N\mathbf{x}_N^T & -l_N\mathbf{x}_N^T & 0 & 0 & \dots & 1 \end{bmatrix}$$

The values $\mathbf{w} = 0$ and $\tau_i = b_i$ is a feasible solution that can be used to start the simplex algorithm.

Demo

link