Support Vector Machines

Sriram Sankararaman

The instructor gratefully acknowledges Fei Sha, Ameet Talwalkar, Eric Eaton, and Jessica Wu whose slides are heavily used, and the many others who made their course material freely available online.

Announcement

- Problem set 3 released.
- Due March 1, 11:59pm. (Note the date!)

Outline

Kernelized ridge regression

Setup

- Input: $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$ (covariates, predictors, features)
- Transformation: $\boldsymbol{\phi}(\boldsymbol{x}) \in \mathbb{R}^{\mathsf{M}}$.
- Output: $y \in \mathbb{R}$ (responses, targets, outcomes, outputs)
- Hypotheses/Model: $h_{\boldsymbol{w}} : \boldsymbol{x} \to y$, with $h_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x})$
- Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, \mathsf{N}\}$

Non-linear basis functions

$$\phi(\boldsymbol{x}) = \begin{pmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \vdots \\ \sqrt{2}x_D \\ {x_1}^2 \\ x_1 x_2 \\ \vdots \\ x_1 x_D \\ x_2 x_1 \\ {x_2}^2 \\ \vdots \\ x_2 x_D \\ \vdots \\ x_D x_1 \\ \vdots \\ {x_D}^2 \end{pmatrix}$$

Two ways to do ridge regression

Primal

Learn $\boldsymbol{w}$ by minimizing the cost function $J(\boldsymbol{w})$:

$$\widehat{\boldsymbol{w}} = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}$$

Predict y for a new input $\boldsymbol{x}$

$$\widehat{\boldsymbol{w}}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$$

Here:

$$\boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathrm{T}} \\ \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}_N)^{\mathrm{T}} \end{pmatrix} \in \mathbb{R}^{N \times M = \mathrm{O}(\mathrm{D}^2)} \qquad \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} \in \mathbb{R}^N$$

Two ways to do ridge regression

Primal

Learn $\boldsymbol{w}$ by minimizing the cost function $J(\boldsymbol{w})$:

$$\widehat{\boldsymbol{w}} = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}$$

Predict y for a new input $\boldsymbol{x}$

$$\widehat{\boldsymbol{w}}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$$

- Cost of computing new features : $\mathrm{O}(\mathrm{ND}^2)$.
- Cost of learning (matrix multiplication and matrix inversion): $\mathrm{O}((\mathrm{D}^2)^2\mathrm{N} + (\mathrm{D}^2)^3)$.
- Cost of prediction $\mathrm{O}(\mathrm{D}^2)$.

Two ways to do ridge regression

Dual

Learn $\boldsymbol{\alpha}$ by minimizing the cost function $J(\boldsymbol{\alpha})$:

$$\widehat{\boldsymbol{\alpha}} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \boldsymbol{y}$$

Predict y for a new input $\boldsymbol{x}$

$$\boldsymbol{y}^{\mathrm{T}} (\boldsymbol{K} + \lambda I)^{-1} \boldsymbol{k}_{\boldsymbol{x}}$$

Here:

$$\boldsymbol{K} = \boldsymbol{\Phi} \boldsymbol{\Phi}^{\mathrm{T}}$$

$$\boldsymbol{k}_{\boldsymbol{x}} = \boldsymbol{\Phi} \boldsymbol{\phi}(\boldsymbol{x})$$

For the feature vector ϕ that we chose, we can compute them as:

$$K_{m,n} = \left(1 + \boldsymbol{x}_m^{\mathrm{T}} \boldsymbol{x}_n\right)^2$$

$$k_{x,n} = \left(1 + \boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{x}\right)^2$$

Two ways to do ridge regression

Dual

Learn $\boldsymbol{\alpha}$ by minimizing the cost function $J(\boldsymbol{\alpha})$:

$$\widehat{\boldsymbol{\alpha}} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \boldsymbol{y}$$

Predict y for a new input $\boldsymbol{x}$

$$\boldsymbol{y}^{\mathrm{T}} (\boldsymbol{K} + \lambda I)^{-1} \boldsymbol{k}_{\boldsymbol{x}}$$

- Computing kernel matrix $\boldsymbol{K}$: O(N^{2}D).
- Cost of learning: O(N^3).
- Cost of prediction: O(N^2).

Kernel functions

- To use kernelized ridge regression, we need to be able to compute an inner product between a test point and any training point.
- Since we need this for any possible pair of points, we need a function that takes a pair of points and computes an inner product.
- This is the kernel function $k(\cdot, \cdot)$.
- Given two inputs, $k(\cdot, \cdot)$ tells us how "similar" or "close" these inputs are in the space defined by the function ϕ.

Kernel functions

Definition: a kernel function $k(\cdot, \cdot)$ is a bivariate function that satisfies the following properties. For any $\boldsymbol{x}_m$ and $\boldsymbol{x}_n$,

$$k(\boldsymbol{x}_m, \boldsymbol{x}_n) = k(\boldsymbol{x}_n, \boldsymbol{x}_m) \text{ and } k(\boldsymbol{x}_m, \boldsymbol{x}_n) = \boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n)$$

for *some* function $\boldsymbol{\phi}(\cdot)$.

Kernel functions

Mercer theorem (loosely), a bivariate function $k(\cdot, \cdot)$ is a kernel function, if and only if, for *any N and any $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, ..., and $\boldsymbol{x}_N$*, the matrix

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

is positive semidefinite.

There are infinite numbers of kernels to use!

Rules of composing kernels (this is just a partial list)

- if $k(\boldsymbol{x}_m, \boldsymbol{x}_n)$ is a kernel, then $ck(\boldsymbol{x}_m, \boldsymbol{x}_n)$ is also if $c > 0$.
- if both $k_1(\boldsymbol{x}_m, \boldsymbol{x}_n)$ and $k_2(\boldsymbol{x}_m, \boldsymbol{x}_n)$ are kernels, then $\alpha k_1(\boldsymbol{x}_m, \boldsymbol{x}_n) + \beta k_2(\boldsymbol{x}_m, \boldsymbol{x}_n)$ are also if $\alpha, \beta \geq 0$
- if both $k_1(\boldsymbol{x}_m, \boldsymbol{x}_n)$ and $k_2(\boldsymbol{x}_m, \boldsymbol{x}_n)$ are kernels, then $k_1(\boldsymbol{x}_m, \boldsymbol{x}_n)k_2(\boldsymbol{x}_m, \boldsymbol{x}_n)$ are also.
- if $k(\boldsymbol{x}_m, \boldsymbol{x}_n)$ is a kernel, then $e^{k(\boldsymbol{x}_m, \boldsymbol{x}_n)}$ is also.
- $\ldots$

In practice, choosing an appropriate kernel is an "art"

People typically start with polynomial and Gaussian RBF kernels or incorporate domain knowledge.

Mini-Summary

- Kernels allow us to design algorithms that use rich set of features while being computationally efficient.
- Many machine learning algorithms can be "kernelized".
- Picking kernels is an art.
- We still need to tune hyperparameters.

Outline

Support vector machines

- One of the most commonly used classification algorithms
- Good generalization in theory and practice
- Can be kernelized

Perceptron (linear classification)

Setup

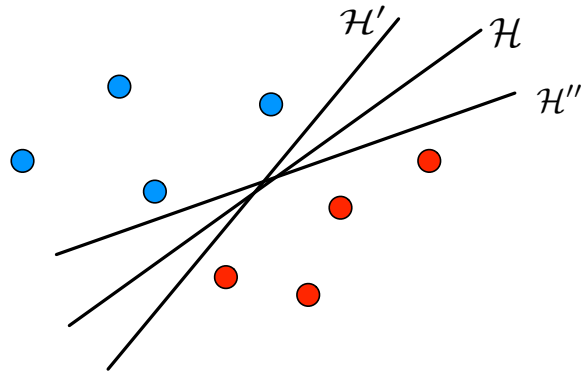
- Input: $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$
- Output: $y \in \{-1, +1\}$
- Hypotheses/Model: $h_{\boldsymbol{w},b} : \boldsymbol{x} \to y$, with $h_{\boldsymbol{w},b}(\boldsymbol{x}) = \text{SIGN}(b + \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}))$

  $\boldsymbol{w} = [w_1\ w_2\ \cdots\ w_{\mathsf{M}}]^{\mathrm{T}}$: *weights*, *parameters*, or *parameter vector*

  b is called *bias*.
- Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, \mathsf{N}\}$

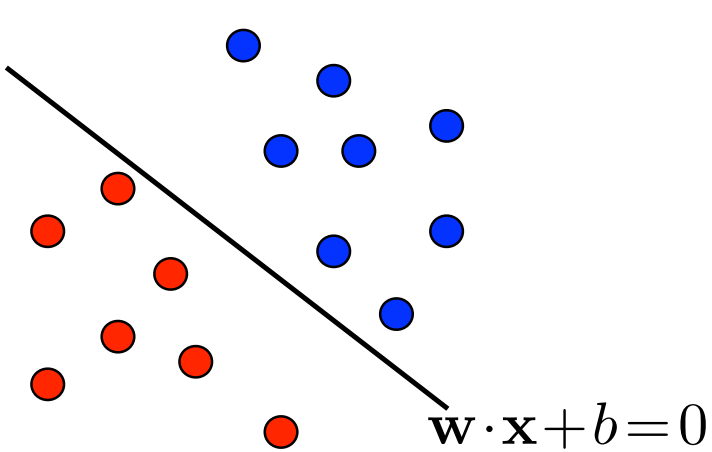
Intuition: where to put the decision boundary?

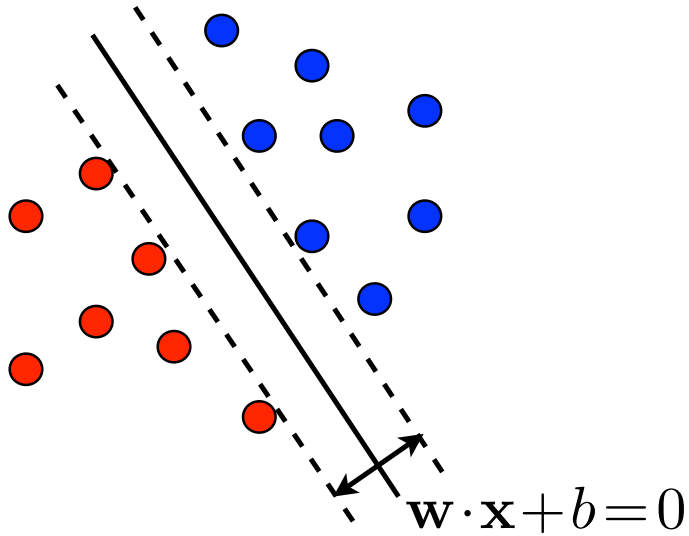
Consider the following *separable* training dataset, i.e., we assume there exists a decision boundary that separates the two classes perfectly. There are an *infinite* number of decision boundaries $\mathcal{H} : \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b = 0$!



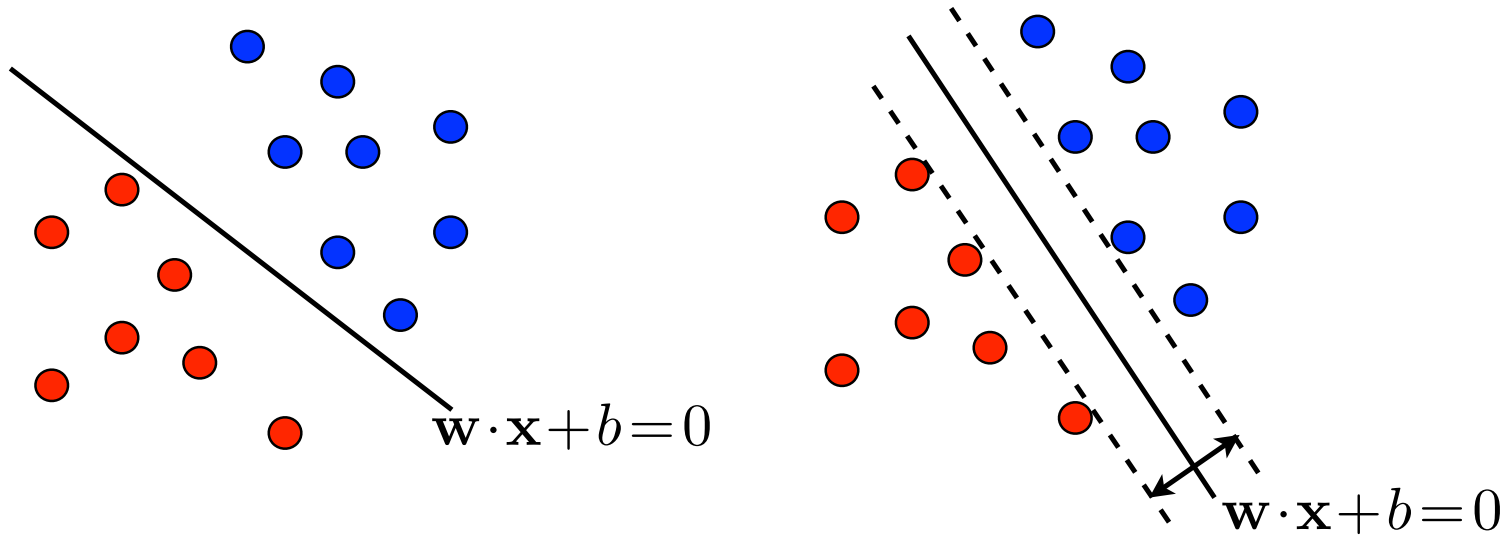
Which one should we pick?

Intuition: where to put the decision boundary?





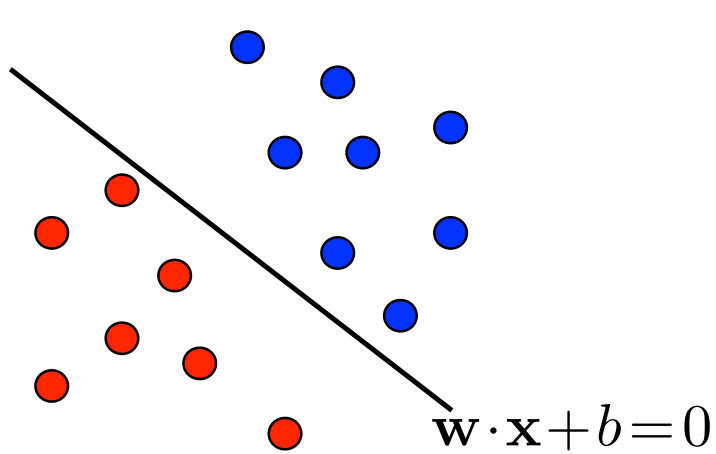
Intuition: where to put the decision boundary?

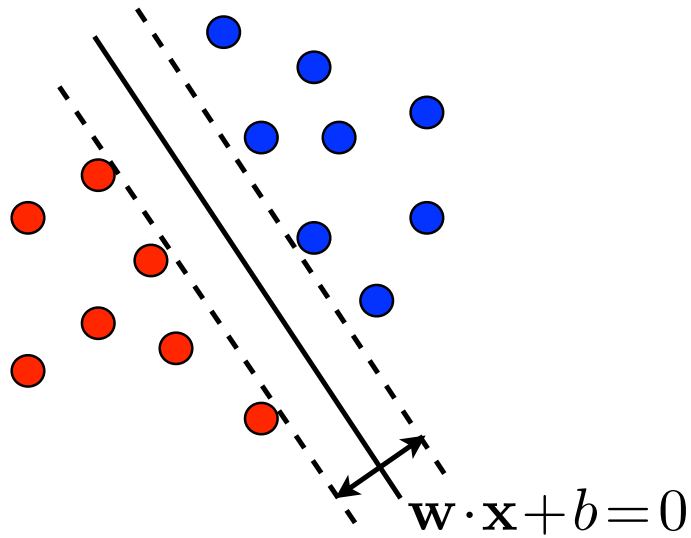


Idea: Find a decision boundary in the '*middle*' of the two classes. In other words, we want a decision boundary that:

- Perfectly classifies the training data
- Is as far away from every training point as possible

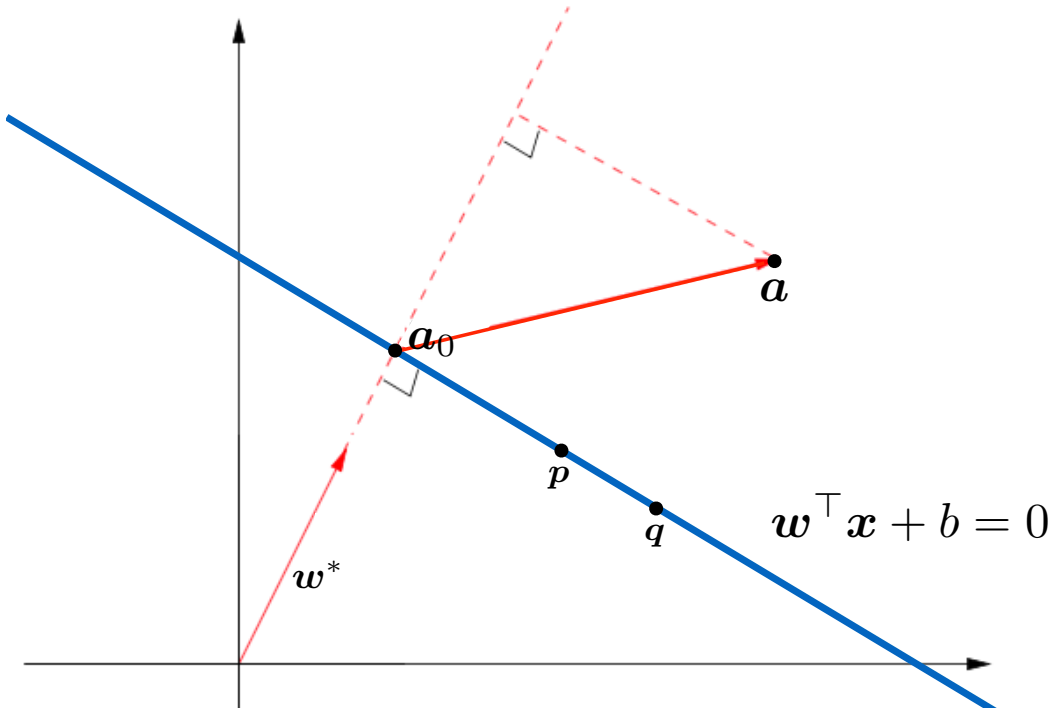
Review of hyperplane





- General equation is $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$
- $\boldsymbol{w} \in \mathbb{R}^d$ is a non-zero normal vector, b is a scalar intercept
- Divides the space in half, i.e., $\boldsymbol{w}^\top \boldsymbol{x} + b > 0$ and $\boldsymbol{w}^\top \boldsymbol{x} + b < 0$
- A hyperplane is a line in 2D and a plane in 3D

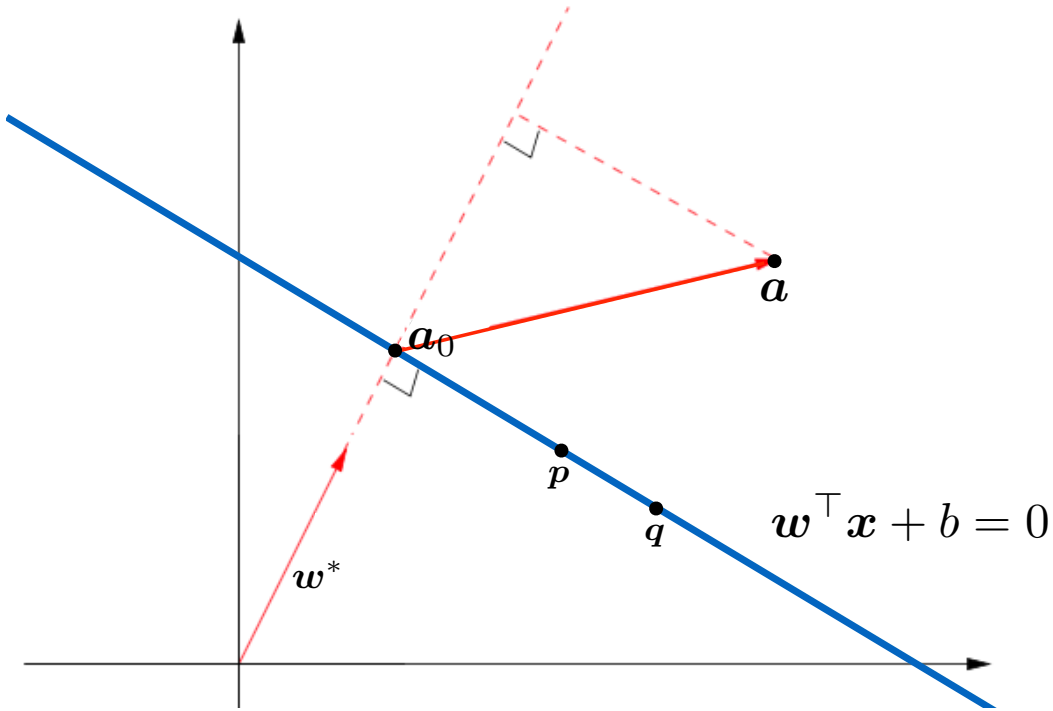
Properties of hyperplanes



Why is w normal to this hyperplane?

- If two points, $\boldsymbol{p}$ and $\boldsymbol{q}$ are both on the hyperplane, then $\boldsymbol{w}^\top(\boldsymbol{p}-\boldsymbol{q}) = \boldsymbol{w}^\top\boldsymbol{p} - \boldsymbol{w}^\top\boldsymbol{q} = b - b = 0$

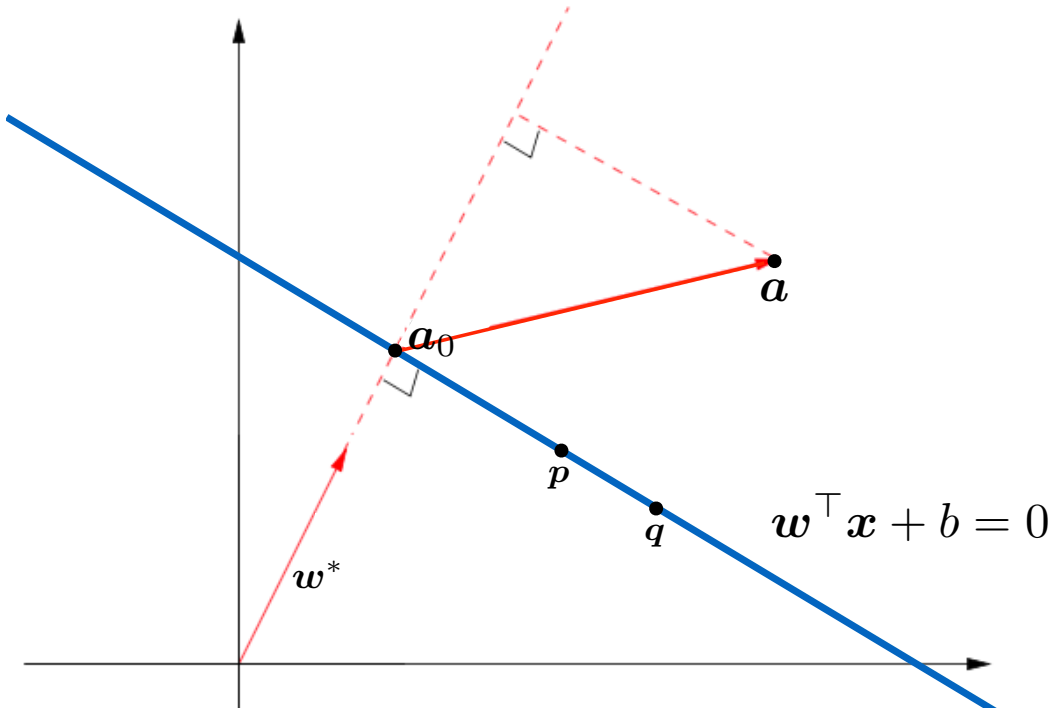
Properties of hyperplanes



Why is w normal to this hyperplane?

- If two points, $\boldsymbol{p}$ and $\boldsymbol{q}$ are both on the hyperplane, then $\boldsymbol{w}^\top(\boldsymbol{p} - \boldsymbol{q}) = \boldsymbol{w}^\top \boldsymbol{p} - \boldsymbol{w}^\top \boldsymbol{q} = b - b = 0$
- $\boldsymbol{p} - \boldsymbol{q}$ is an arbitrary vector parallel to the hyperplane, thus $\boldsymbol{w}$ is orthogonal

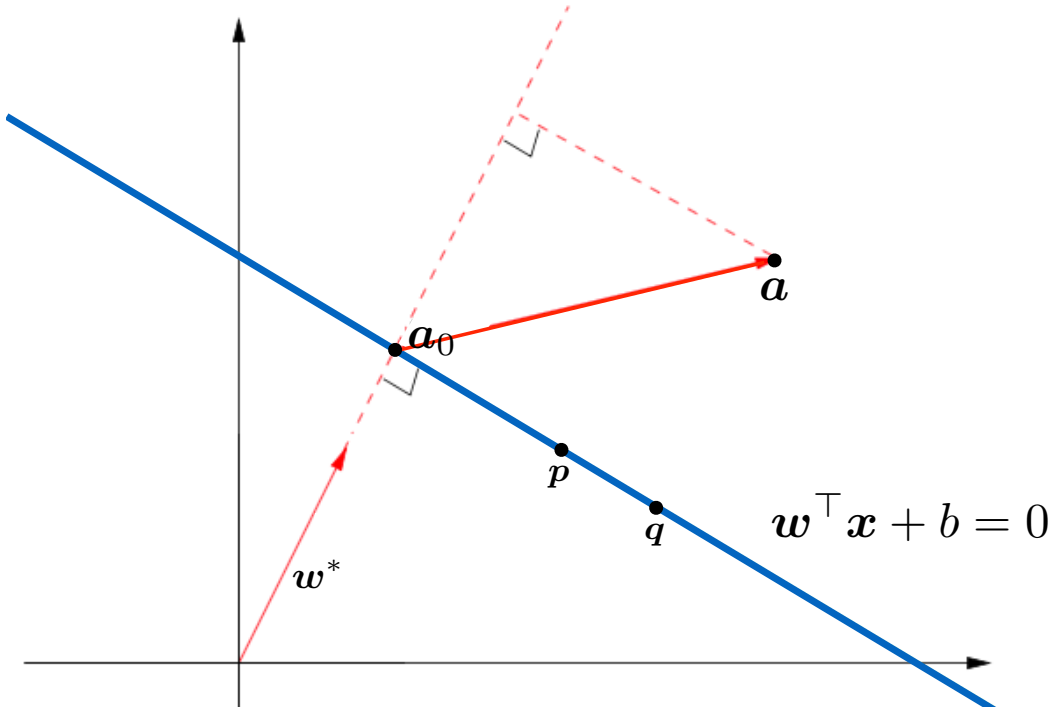
Properties of hyperplanes



Why is w normal to this hyperplane?

- If two points, $\boldsymbol{p}$ and $\boldsymbol{q}$ are both on the hyperplane, then $\boldsymbol{w}^\top(\boldsymbol{p} - \boldsymbol{q}) = \boldsymbol{w}^\top \boldsymbol{p} - \boldsymbol{w}^\top \boldsymbol{q} = b - b = 0$
- $\boldsymbol{p} - \boldsymbol{q}$ is an arbitrary vector parallel to the hyperplane, thus $\boldsymbol{w}$ is orthogonal
- $\boldsymbol{w}^* = \frac{\boldsymbol{w}}{||\boldsymbol{w}||}$ is the unit normal vector

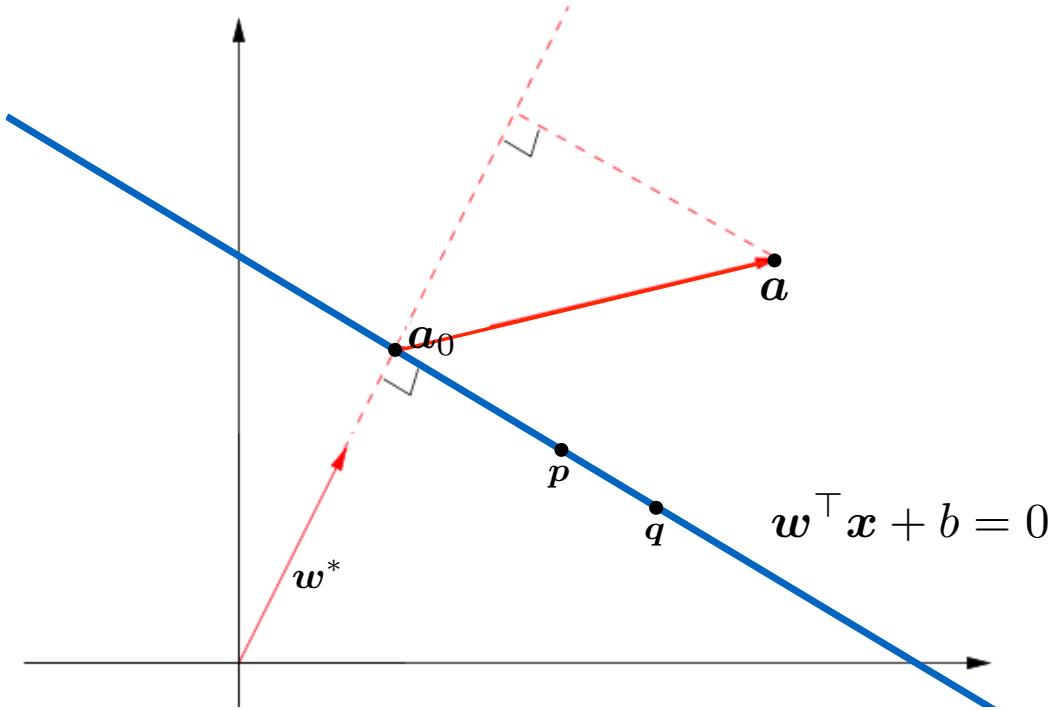
Properties of hyperplanes



How to compute signed distance from a to the hyperplane?

- If we define point a_0 on the hyperplane, then this distance corresponds to length of $a - a_0$ in direction of w^*, which equals $w^{*\top}(a - a_0) = \frac{w}{||w||}^\top (a - a_0) = \frac{w^\top a - w^\top a_0}{||w||}$

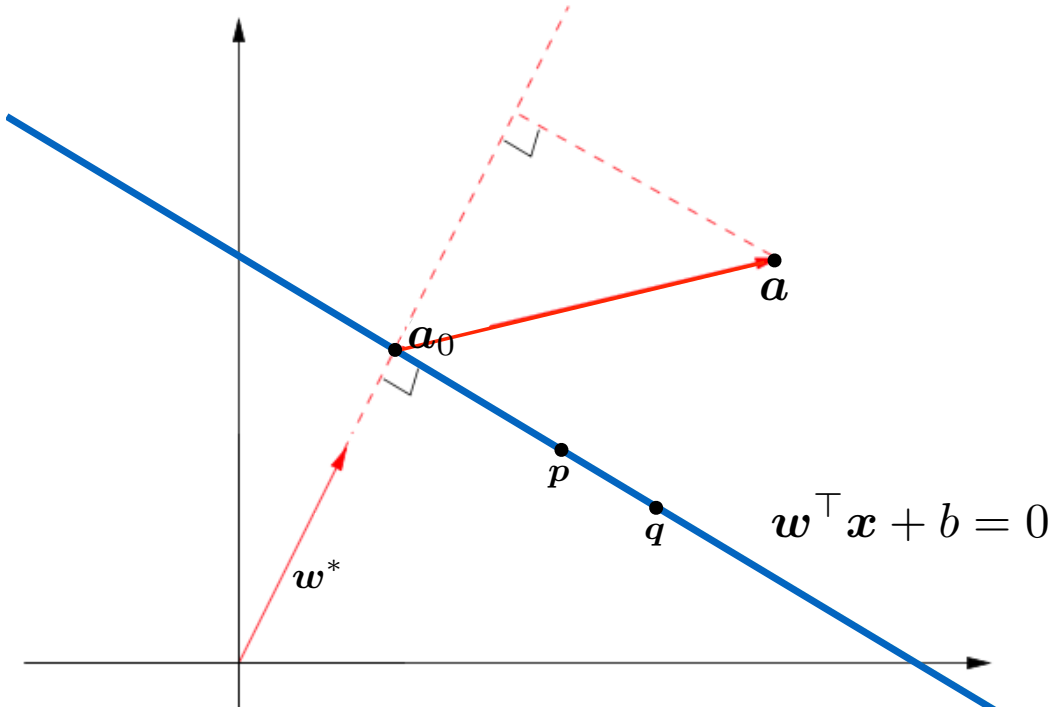
Properties of hyperplanes



How to compute signed distance from a to the hyperplane?

- If we define point $\boldsymbol{a}_0$ on the hyperplane, then this distance corresponds to length of $\boldsymbol{a} - \boldsymbol{a}_0$ in direction of $\boldsymbol{w}^*$, which equals $\boldsymbol{w}^{*\top}(\boldsymbol{a} - \boldsymbol{a}_0) = \frac{\boldsymbol{w}}{||\boldsymbol{w}||}^\top (\boldsymbol{a} - \boldsymbol{a}_0) = \frac{\boldsymbol{w}^\top \boldsymbol{a} - \boldsymbol{w}^\top \boldsymbol{a}_0}{||\boldsymbol{w}||}$
- Since $\boldsymbol{w}^\top \boldsymbol{a}_0 = -b$,

Properties of hyperplanes



How to compute signed distance from a to the hyperplane?

- If we define point $\boldsymbol{a}_0$ on the hyperplane, then this distance corresponds to length of $\boldsymbol{a} - \boldsymbol{a}_0$ in direction of $\boldsymbol{w}^*$, which equals $\boldsymbol{w}^{*\top}(\boldsymbol{a} - \boldsymbol{a}_0) = \frac{\boldsymbol{w}}{||\boldsymbol{w}||}^\top (\boldsymbol{a} - \boldsymbol{a}_0) = \frac{\boldsymbol{w}^\top \boldsymbol{a} - \boldsymbol{w}^\top \boldsymbol{a}_0}{||\boldsymbol{w}||}$
- Since $\boldsymbol{w}^\top \boldsymbol{a}_0 = -b$, the distance equals $\frac{1}{||\boldsymbol{w}||}(\boldsymbol{w}^\top \boldsymbol{a} + b)$

Distance from a point to decision boundary

The *unsigned* distance from a point $\boldsymbol{\phi}(\boldsymbol{x})$ to decision boundary (hyperplane) $\mathcal{H}$ is

$$d_{\mathcal{H}}(\boldsymbol{\phi}(\boldsymbol{x})) = \frac{|\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b|}{\|\boldsymbol{w}\|_2}$$

Distance from a point to decision boundary

The *unsigned* distance from a point $\boldsymbol{\phi}(\boldsymbol{x})$ to decision boundary (hyperplane) $\mathcal{H}$ is

$$d_{\mathcal{H}}(\boldsymbol{\phi}(\boldsymbol{x})) = \frac{|\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b|}{\|\boldsymbol{w}\|_2}$$

We can remove the absolute value $|\cdot|$ by exploiting the fact that the decision boundary classifies every point in the training dataset correctly.

Distance from a point to decision boundary

The *unsigned* distance from a point $\boldsymbol{\phi}(\boldsymbol{x})$ to decision boundary (hyperplane) $\mathcal{H}$ is

$$d_{\mathcal{H}}(\boldsymbol{\phi}(\boldsymbol{x})) = \frac{|\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b|}{\|\boldsymbol{w}\|_2}$$

We can remove the absolute value $|\cdot|$ by exploiting the fact that the decision boundary classifies every point in the training dataset correctly.

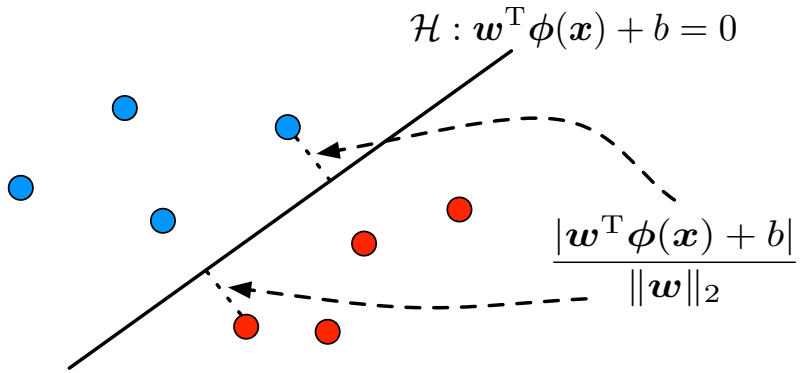
Namely, $(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b)$ and $\boldsymbol{x}$'s label y must have the same sign, so:

$$d_{\mathcal{H}}(\boldsymbol{\phi}(\boldsymbol{x})) = \frac{y[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b]}{\|\boldsymbol{w}\|_2}$$

Optimizing the Margin

Margin Smallest distance between the hyperplane and all training points

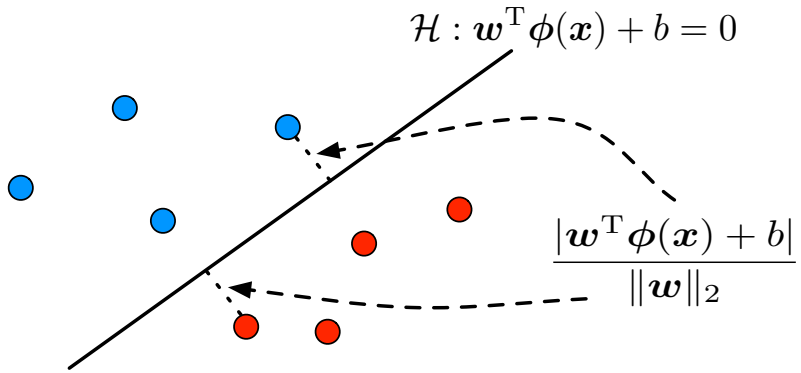
$$\text{MARGIN}(\boldsymbol{w}, b) = \min_n \frac{y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]}{\|\boldsymbol{w}\|_2}$$



Optimizing the Margin

Margin Smallest distance between the hyperplane and all training points

$$\text{MARGIN}(\boldsymbol{w}, b) = \min_n \frac{y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]}{\|\boldsymbol{w}\|_2}$$

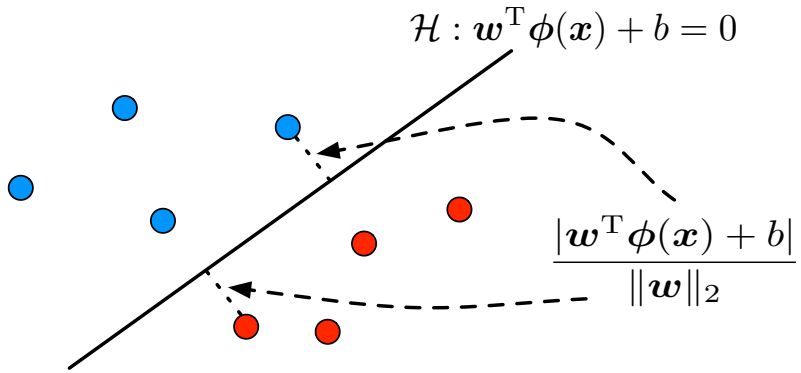


How should we pick (w, b) based on its margin?

Optimizing the Margin

Margin Smallest distance between the hyperplane and all training points

$$\text{MARGIN}(\boldsymbol{w}, b) = \min_n \frac{y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]}{\|\boldsymbol{w}\|_2}$$



How should we pick (w, b) based on its margin?

We want a decision boundary that is as far away from all training points as possible, so we want to *maximize* the margin!

$$\max_{\boldsymbol{w},b} \min_n \frac{y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]}{\|\boldsymbol{w}\|_2} = \max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|_2} \min_n y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]$$

Scale of $\boldsymbol{w}$

Margin Smallest distance between the hyperplane and all training points

$$\text{MARGIN}(\boldsymbol{w}, b) = \min_n \frac{y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]}{\|\boldsymbol{w}\|_2}$$

Consider three hyperplanes

- $(\boldsymbol{w}, b)$
- $(2\boldsymbol{w}, 2b)$
- $(.5\boldsymbol{w}, .5b)$

Which one has the largest margin?

Scale of $\boldsymbol{w}$

Margin Smallest distance between the hyperplane and all training points

$$\text{MARGIN}(\boldsymbol{w}, b) = \min_n \frac{y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]}{\|\boldsymbol{w}\|_2}$$

Consider three hyperplanes

- $(\boldsymbol{w}, b)$
- $(2\boldsymbol{w}, 2b)$
- $(.5\boldsymbol{w}, .5b)$

Which one has the largest margin?

- The MARGIN doesn't change if we scale $(\boldsymbol{w}, b)$ by a constant factor c
- $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b = 0$ and $(c\boldsymbol{w})^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + (cb) = 0$: same decision boundary!
- Can we further constrain the problem?

Rescaled Margin

We can further constrain the problem by scaling $(\boldsymbol{w}, b)$ such that

$$\min_n y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] = 1$$

Rescaled Margin

We can further constrain the problem by scaling $(\boldsymbol{w}, b)$ such that

$$\min_n y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] = 1$$

We've fixed the numerator in the $\textsc{margin}(\boldsymbol{w}, b)$ equation, and we have:

$$\textsc{margin}(\boldsymbol{w}, b) = \frac{1}{\|\boldsymbol{w}\|_2}$$

Rescaled Margin

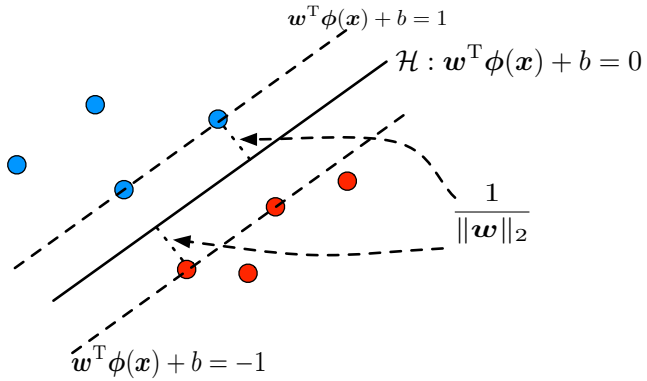
We can further constrain the problem by scaling $(\boldsymbol{w}, b)$ such that

$$\min_n y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] = 1$$

We've fixed the numerator in the $\textsc{margin}(\boldsymbol{w}, b)$ equation, and we have:

$$\textsc{margin}(\boldsymbol{w}, b) = \frac{1}{\|\boldsymbol{w}\|_2}$$

Hence the points closest to the decision boundary are at distance $\frac{1}{||\boldsymbol{w}||_2}$!



SVM: max margin formulation for separable data

Assuming separable training data, we thus want to solve:

$$\max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|_2} \quad \text{such that} \quad \min_n y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] = 1$$

This is equivalent to

$$\max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|_2} \quad \text{such that} \quad y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1, \quad n \in \{1, \ldots, N\}$$

SVM: max margin formulation for separable data

Assuming separable training data, we thus want to solve:

$$\max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|_2} \quad \text{such that} \quad \min_n y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] = 1$$

This is equivalent to

$$\max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|_2} \quad \text{such that} \quad y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1, \quad n \in \{1, \ldots, N\}$$

This is equivalent to

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1, \quad n \in \{1, \ldots, N\} \end{aligned}$$

Given our geometric intuition, SVM is called a *max margin* (or large margin) classifier. The constraints are called *large margin constraints*.

SVM for separable data

Constrained optimization problem

$$\min_{\boldsymbol{w},b} \quad \underbrace{\frac{1}{2}\|\boldsymbol{w}\|_2^2}_{\text{Objective function}}$$

$$\text{s.t.} \quad \underbrace{y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n)+b] \geq 1}_{\text{Constraint}}, \quad n \in \{1,\ldots,N\}$$

- Variables $(\boldsymbol{w}, b)$
- Objective function
- Constraints

Convex optimization problem (also called convex program)

$$\begin{aligned} \min_{\boldsymbol{x}} \; & f_0(\boldsymbol{x}) \\ \text{s.t.} \; & f_i(\boldsymbol{x}) \leq 0, \quad i = 1, \ldots, m \\ & \boldsymbol{a}_i^{\mathrm{T}} \boldsymbol{x} = b_i, \quad i = 1, \ldots, p \end{aligned}$$

where $\boldsymbol{x} \in \mathbb{R}^n$, $f_0, \ldots, f_m$ are convex functions.

- A point $\boldsymbol{x}$ that satisfies all constraints is feasible.
- If there exists at least one feasible point, the problem is feasible. Otherwise, it is infeasible.

SVM for separable data

Constrained optimization problem

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & -y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n)+b]+1 \leq 0, \quad n \in \{1,\ldots,N\} \end{aligned}$$

- Objective function: convex in $(\boldsymbol{w}, b)$
- Inequality constraints are linear in $(\boldsymbol{w}, b)$. Hence, convex.

SVM for non-separable data

SVM formulation for separable data

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1, \quad n \in \{1, \ldots, N\} \end{aligned}$$

Non-separable setting In practice our training data will not be separable. What issues arise with the optimization problem above when data is not separable?

SVM for non-separable data

SVM formulation for separable data

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1, \quad n \in \{1, \ldots, N\} \end{aligned}$$

Non-separable setting In practice our training data will not be separable. What issues arise with the optimization problem above when data is not separable?

- For every $\boldsymbol{w}$ there exists a training point $\boldsymbol{x}_i$ such that

$$y_i[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b] \leq 0$$

- There is no feasible $(\boldsymbol{w}, b)$ as at least one of our constraints is violated!

SVM for non-separable data

Constraints in separable setting

$$y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1, \quad n \in \{1, \ldots, N\}$$

Constraints in non-separable setting

Idea: modify our constraints to account for non-separability! Specifically, we introduce slack variables $\xi_n \geq 0$:

$$y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1 - \xi_n, \quad n \in \{1, \ldots, N\}$$

SVM for non-separable data

Constraints in separable setting

$$y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1, \quad n \in \{1, \ldots, N\}$$

Constraints in non-separable setting

Idea: modify our constraints to account for non-separability! Specifically, we introduce slack variables $\xi_n \geq 0$:

$$y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1 - \xi_n, \quad n \in \{1, \ldots, N\}$$

- For "hard" training points, we can increase ξ_n until the above inequalities are met
- What does it mean when ξ_n is very large?

Soft-margin SVM formulation

We do not want ξ_n to grow too large, and we can control their size by incorporating them into our optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_n \xi_n \\ \text{s.t.} \quad & y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1 - \xi_n, \quad n \in \{1,\ldots,N\} \\ & \xi_n \geq 0, \quad n \in \{1,\ldots,N\} \end{aligned}$$

Soft-margin SVM formulation

We do not want ξ_n to grow too large, and we can control their size by incorporating them into our optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_n \xi_n \\ \text{s.t.} \quad & y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1 - \xi_n, \quad n \in \{1,\ldots,N\} \\ & \xi_n \geq 0, \quad n \in \{1,\ldots,N\} \end{aligned}$$

What is the role of C?

Soft-margin SVM formulation

We do not want ξ_n to grow too large, and we can control their size by incorporating them into our optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_n \xi_n \\ \text{s.t.} \quad & y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1 - \xi_n, \quad n \in \{1,\ldots,N\} \\ & \xi_n \geq 0, \quad n \in \{1,\ldots,N\} \end{aligned}$$

What is the role of C?

- User-defined hyperparameter
- Trades off between the two terms in our objective
- Same idea as the regularization term in ridge regression, i.e., $C = \frac{1}{\lambda}$

How to solve this problem?

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_n \xi_n \\ \text{s.t.} \quad & y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1 - \xi_n, \quad n \in \{1,\ldots,N\} \\ & \xi_n \geq 0, \quad n \in \{1,\ldots,N\} \end{aligned}$$

- This is a *convex problem*: the objective function is quadratic in $\boldsymbol{w}$ and linear in $\boldsymbol{\xi}$ and the constraints are linear (inequality) constraints in $\boldsymbol{w}$, b and ξ_n.

How to solve this problem?

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_n \xi_n \\ \text{s.t.} \quad & y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1 - \xi_n, \quad n \in \{1,\ldots,N\} \\ & \xi_n \geq 0, \quad n \in \{1,\ldots,N\} \end{aligned}$$

- This is a *convex problem*: the objective function is quadratic in $\boldsymbol{w}$ and linear in $\boldsymbol{\xi}$ and the constraints are linear (inequality) constraints in $\boldsymbol{w}$, b and ξ_n.
- Given $\boldsymbol{\phi}(\cdot)$, we can solve the optimization problem using general-purpose solvers.

How to solve this problem?

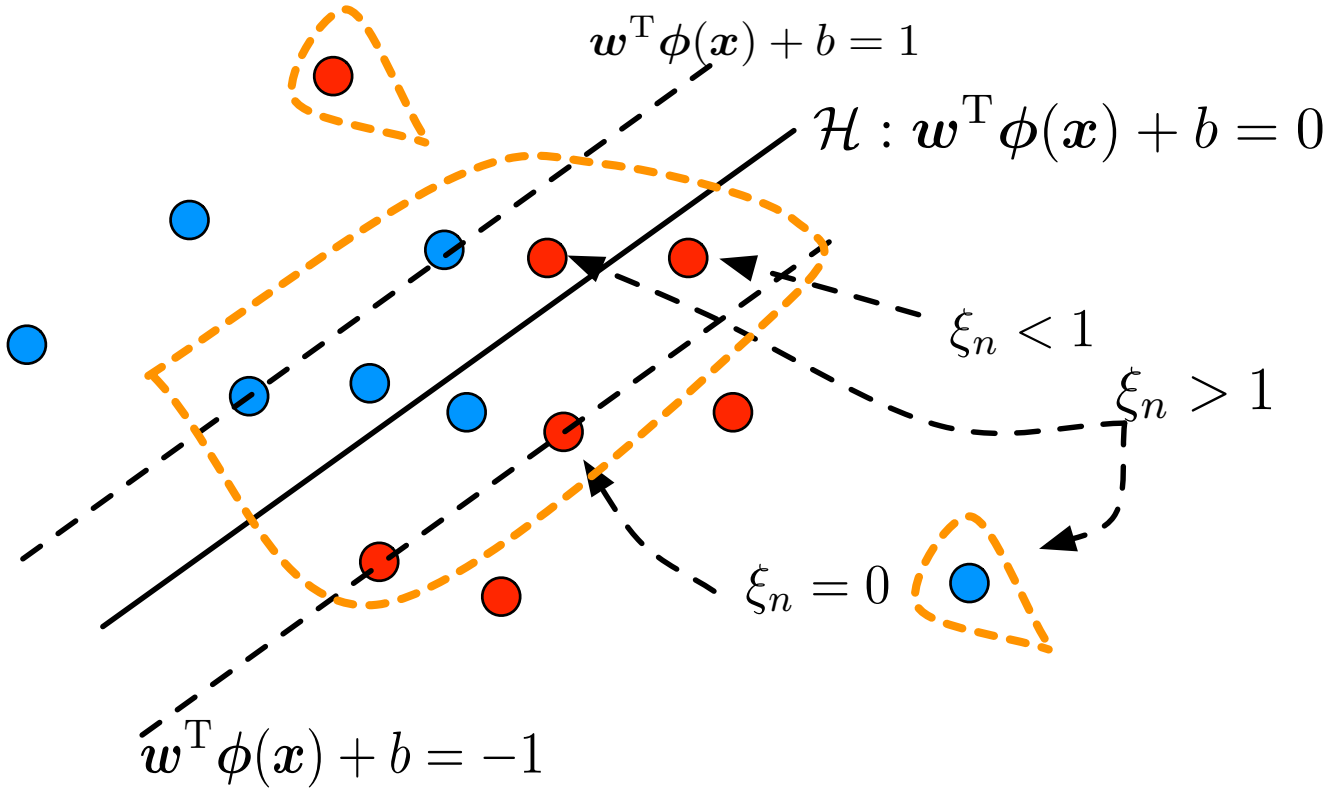
$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_n \xi_n \\ \text{s.t.} \quad & y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1 - \xi_n, \quad n \in \{1,\ldots,N\} \\ & \xi_n \geq 0, \quad n \in \{1,\ldots,N\} \end{aligned}$$

- This is a *convex problem*: the objective function is quadratic in $\boldsymbol{w}$ and linear in $\boldsymbol{\xi}$ and the constraints are linear (inequality) constraints in $\boldsymbol{w}$, b and ξ_n.
- Given $\boldsymbol{\phi}(\cdot)$, we can solve the optimization problem using general-purpose solvers.
- There are several specialized methods for solving this problem, taking advantage of the special structure of the objective function and the constraints (we will not discuss them). Most existing SVM implementation/packages leverage these methods.

Meaning of "support vectors" in SVMs

- The SVM solution is only determined by a subset of the training instances.
- These instances are called *support vectors*
- All other training points do not affect the optimal solution, i.e.,if remove the other points and construct another SVM classifier on the reduced dataset, the optimal solution will be the same

Visualization of how training data points are categorized



Support vectors are highlighted by the dotted orange lines

Outline

A general view of supervised learning

Definition Assume $y \in \{-1, 1\}$ and the decision rule is $h(\boldsymbol{x}) = \text{SIGN}(a(\boldsymbol{x}))$ with $a(\boldsymbol{x}) = \boldsymbol{w}^{\text{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b$

For classification: $0/1$ loss

$$\ell^{0/1}(y, a(\boldsymbol{x})) = \begin{cases} 0 & \text{if } ya(\boldsymbol{x}) \geq 0 \\ 1 & \text{otherwise} \end{cases}$$

Minimize weighted sum of empirical risk and regularizer

$$\begin{aligned} &\arg\min_{\boldsymbol{w},b} R^{\mathrm{EMP}}[h_{\boldsymbol{w},b}(x)] + \lambda R(\boldsymbol{w}, b) \\ &= \arg\min_{\boldsymbol{w},b} \frac{1}{N} \sum_n \ell(y_n, a(\boldsymbol{x}_n)) + \lambda R(\boldsymbol{w}, b) \end{aligned} \tag{1}$$

Minimize weighted sum of empirical risk and regularizer

$$\begin{aligned} &\arg\min_{\boldsymbol{w},b} R^{\mathrm{EMP}}[h_{\boldsymbol{w},b}(x)] + \lambda R(\boldsymbol{w}, b) \\ &= \arg\min_{\boldsymbol{w},b} \frac{1}{N} \sum_{n} \ell(y_n, a(\boldsymbol{x}_n)) + \lambda R(\boldsymbol{w}, b) \end{aligned} \tag{1}$$

- Problem with minimizing the $0/1$ loss ?

Hinge loss

Definition Assume $y \in \{-1, 1\}$ and the decision rule is $h(\boldsymbol{x}) = \text{SIGN}(a(\boldsymbol{x}))$ with $a(\boldsymbol{x}) = \boldsymbol{w}^{\text{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b$,

$$\ell^{\text{HINGE}}(y, a(\boldsymbol{x})) = \begin{cases} 0 & \text{if } ya(\boldsymbol{x}) \geq 1 \\ 1 - ya(\boldsymbol{x}) & \text{otherwise} \end{cases}$$

Intuition

Hinge loss

Definition Assume $y \in \{-1, 1\}$ and the decision rule is $h(\boldsymbol{x}) = \text{SIGN}(a(\boldsymbol{x}))$ with $a(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b$,

$$\ell^{\text{HINGE}}(y, a(\boldsymbol{x})) = \begin{cases} 0 & \text{if } ya(\boldsymbol{x}) \geq 1 \\ 1 - ya(\boldsymbol{x}) & \text{otherwise} \end{cases}$$

Intuition

- No penalty if raw output, $a(\boldsymbol{x})$, has same sign and is far enough from decision boundary (i.e., if 'margin' is large enough)
- Otherwise pay a growing penalty, between 0 and 1 if signs match, and greater than one otherwise

Hinge loss

Definition Assume $y \in \{-1, 1\}$ and the decision rule is $h(\boldsymbol{x}) = \text{SIGN}(a(\boldsymbol{x}))$ with $a(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b$,

$$\ell^{\text{HINGE}}(y, a(\boldsymbol{x})) = \begin{cases} 0 & \text{if } ya(\boldsymbol{x}) \geq 1 \\ 1 - ya(\boldsymbol{x}) & \text{otherwise} \end{cases}$$

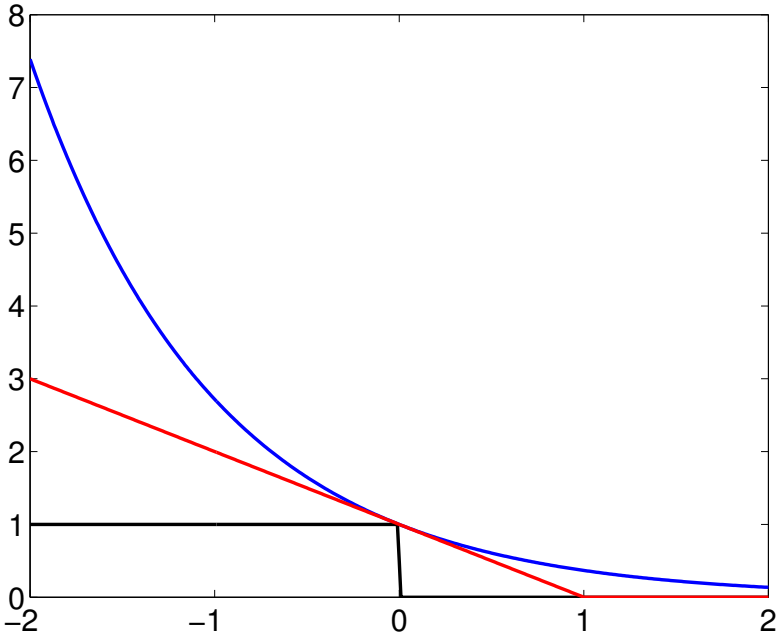
Intuition

- No penalty if raw output, $a(\boldsymbol{x})$, has same sign and is far enough from decision boundary (i.e., if 'margin' is large enough)
- Otherwise pay a growing penalty, between 0 and 1 if signs match, and greater than one otherwise

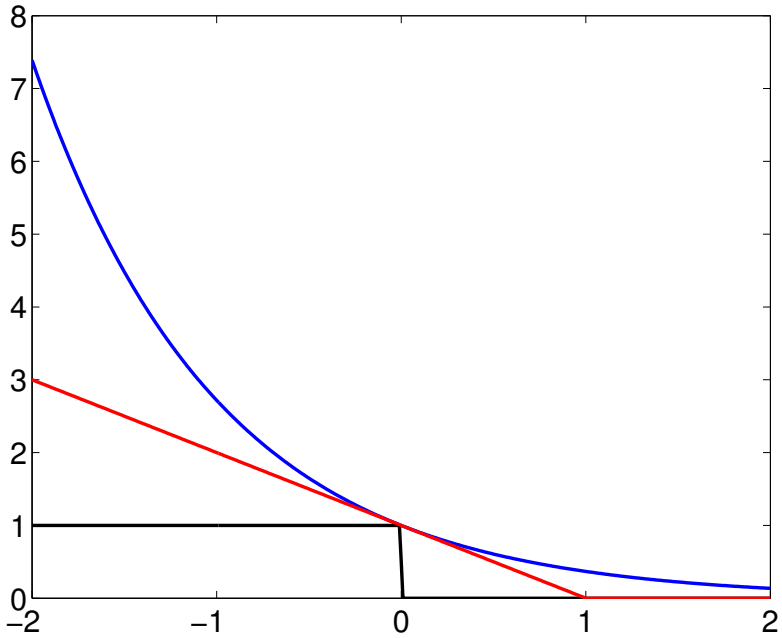
Convenient shorthand

$$\ell^{\text{HINGE}}(y, a(\boldsymbol{x})) = \max(0, 1 - ya(\boldsymbol{x})) = (1 - ya(\boldsymbol{x}))_+$$

Visualization and Properties

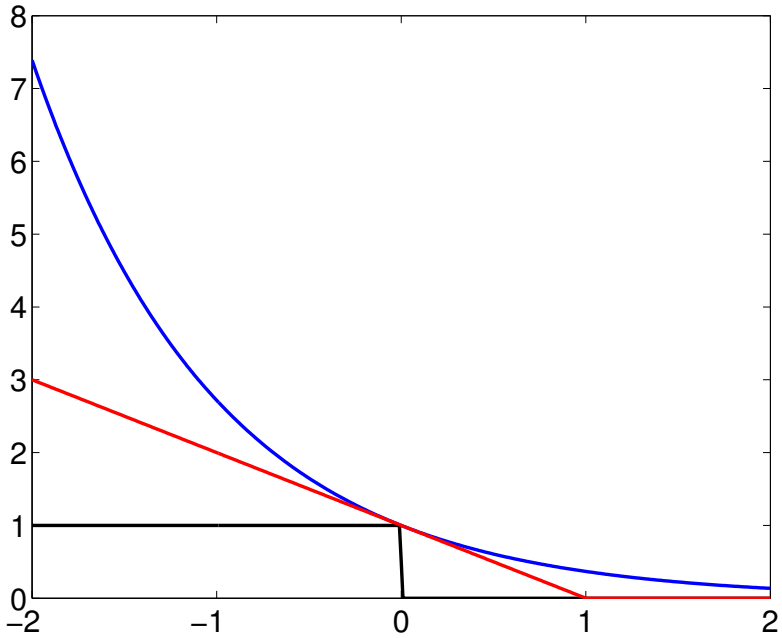


Visualization and Properties



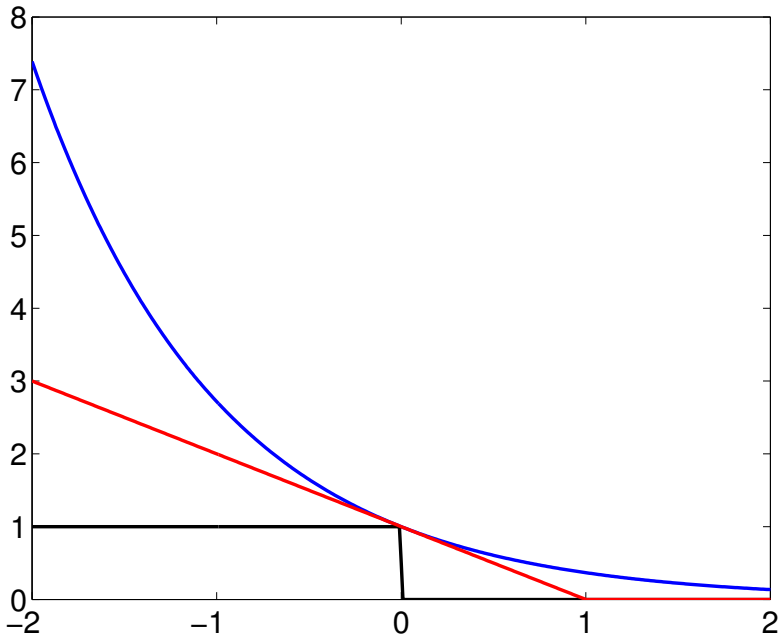
- Upper-bound for $0/1$ loss function (black line)
- We use hinge loss as a *surrogate* to $0/1$ loss – Why?

Visualization and Properties



- Upper-bound for $0/1$ loss function (black line)
- We use hinge loss as a *surrogate* to $0/1$ loss – Why?
- Hinge loss is convex, and thus easier to work with.

Visualization and Properties



- Other surrogate losses can be used, e.g., exponential loss for Adaboost (in blue), logistic loss (not shown) for logistic regression
- Hinge loss less sensitive to outliers than exponential (or logistic) loss

Primal formulation of support vector machines (SVM)

Minimizing the total hinge loss on all the training data with l_2 regularization

$$\min_{\boldsymbol{w},b} \sum_n \max(0, 1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

Analogous to l_2 regularized least squares, as we balance between two terms (the loss and the regularizer).

Primal formulation of support vector machines (SVM)

Minimizing the total hinge loss on all the training data with l_2 regularization

$$\min_{\boldsymbol{w},b} \sum_n \max(0, 1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

Analogous to l_2 regularized least squares, as we balance between two terms (the loss and the regularizer).

Previously, we used geometric arguments to derive:

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_n \xi_n \\ \text{s.t.} \quad & y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1 - \xi_n \quad \text{and} \quad \xi_n \geq 0, \quad n \in \{1, \ldots, N\} \end{aligned}$$

Do these yield the same solution?

Recovering our previous SVM formulation

Minimizing the total hinge loss on all the training data

$$\min_{\boldsymbol{w},b} \sum_n \max(0, 1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

Define $C = 1/\lambda$:

$$\min_{\boldsymbol{w},b} C\sum_n \max(0, 1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]) + \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

Recovering our previous SVM formulation

Minimizing the total hinge loss on all the training data

$$\min_{\boldsymbol{w},b} \sum_n \max(0, 1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

Define $C = 1/\lambda$:

$$\min_{\boldsymbol{w},b} C\sum_n \max(0, 1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]) + \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

Define $\xi_n = \max(0, 1 - y_n[\boldsymbol{w}^{\mathbf{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b])$

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & \max(0, 1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]) = \xi_n, \quad n \in \{1, \ldots, N\} \end{aligned}$$

Recovering our previous SVM formulation

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & \max(0, 1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]) = \xi_n, \quad n \in \{1, \ldots, N\} \end{aligned}$$

is equivalent to

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & 1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \leq \xi_n, \quad n \in \{1, \ldots, N\} \\ & 0 \leq \xi_n, \quad n \in \{1, \ldots, N\} \end{aligned}$$

At optimal solution constraints are active so we have equality! Why?

At optimal solution constraints are active so we have equality! Why?

- If $\xi_n^* > \max(0, 1 - y_n f(\boldsymbol{x}_n))$, we could choose $\bar{\xi}_n < \xi_n^*$ and still satisfy the constraint while reducing our objective function!
- Since $c \geq \max(a, b) \iff c \geq a,\ c \geq b$, we recover previous formulation