Support Vector Machines

Professor Ameet Talwalkar

Outline

Announcements

- HW4 online and due next Monday
- Midterms are still being graded

Outline

Basic and important machine learning concepts

Supervised learning

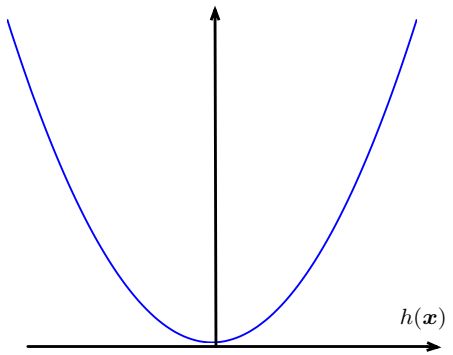
We aim to build a function $h(\boldsymbol{x})$ to predict the true value y associated with $\boldsymbol{x}$. If we make a mistake, we incur a *loss*

$$\ell(h(\boldsymbol{x}), y)$$

Example: quadratic loss function for regression when y is continuous

$$\ell(h(\boldsymbol{x}), y) = [h(\boldsymbol{x}) - y]^2$$

Ex: when $y = 0$



Measure how good our predictor is

Risk: assume we know the true distribution of data $p(\boldsymbol{x}, y)$, the *risk* is

$$R[h(\boldsymbol{x})] = \int_{\boldsymbol{x},y} \ell(h(\boldsymbol{x}), y) p(\boldsymbol{x}, y) d\boldsymbol{x} d\, y$$

However, we cannot compute $R[h(\boldsymbol{x})]$, so we use *empirical risk*, given a training dataset $\mathcal{D}$

$$R^{\text{EMP}}[h(\boldsymbol{x})] = \frac{1}{N} \sum_n \ell(h(\boldsymbol{x}_n), y_n)$$

Intuitively, as $N \to +\infty$,

$$R^{\text{EMP}}[h(\boldsymbol{x})] \to R[h(\boldsymbol{x})]$$

How this relates to what we have learned?

So far, we have been doing empirical risk minimization (ERM)

- For linear regression, $h(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$, and we use squared loss
- For logistic regression, $h(\boldsymbol{x}) = \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})$, and we use cross-entropy loss

How this relates to what we have learned?

So far, we have been doing empirical risk minimization (ERM)

- For linear regression, $h(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$, and we use squared loss
- For logistic regression, $h(\boldsymbol{x}) = \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})$, and we use cross-entropy loss

ERM can lead to overfitting

- If $h(\boldsymbol{x})$ is complicated enough, $R^{\mathrm{EMP}}[h(\boldsymbol{x})] \to 0$, but $h(\boldsymbol{x})$ may not generalize well to new data

Regularization can help prevent overfitting

- The bias-variance analysis in the context of squared loss helps us to understand why regularization can help

Bias/variance tradeoff for regression

Goal: to understand the sources of prediction errors

- $\mathcal{D}$: our training data
- $h_{\mathcal{D}}(\boldsymbol{x})$: our prediction function
  - subscript $\mathcal{D}$ indicates that function is learned on a specific training set
- $\ell(h(\boldsymbol{x}), y)$: squared loss function for regression

$$\ell(h_{\mathcal{D}}(\boldsymbol{x}), y) = [h_{\mathcal{D}}(\boldsymbol{x}) - y]^2$$

- Unknown joint distribution $p(\boldsymbol{x}, y)$

The effect of finite training samples

Risk of prediction function, $h_{\mathcal{D}}(x)$

$$R[h_{\mathcal{D}}(\boldsymbol{x})] = \mathbb{E}_{(\boldsymbol{x},y)\sim p(\boldsymbol{x},y)}\ell(h_{\mathcal{D}}(\boldsymbol{x}), y) = \int_{\boldsymbol{x}}\int_{y}[h_{\mathcal{D}}(\boldsymbol{x}) - y]^2 p(\boldsymbol{x}, y)d\boldsymbol{x}dy$$

The effect of finite training samples

Risk of prediction function, $h_{\mathcal{D}}(x)$

$$R[h_{\mathcal{D}}(\boldsymbol{x})] = \mathbb{E}_{(\boldsymbol{x},y)\sim p(\boldsymbol{x},y)}\ell(h_{\mathcal{D}}(\boldsymbol{x}), y) = \int_{\boldsymbol{x}}\int_{y}[h_{\mathcal{D}}(\boldsymbol{x}) - y]^2 p(\boldsymbol{x}, y)d\boldsymbol{x}dy$$

But, $\mathcal{D}$ is a random sample from the following joint distribution

$$\mathcal{D} \sim P(\mathcal{D}) = \prod_{n=1}^{N} p(\boldsymbol{x}_n, y_n)$$

The effect of finite training samples

Risk of prediction function, $h_{\mathcal{D}}(x)$

$$R[h_{\mathcal{D}}(\boldsymbol{x})] = \mathbb{E}_{(\boldsymbol{x},y)\sim p(\boldsymbol{x},y)}\ell(h_{\mathcal{D}}(\boldsymbol{x}), y) = \int_{\boldsymbol{x}}\int_{y}[h_{\mathcal{D}}(\boldsymbol{x}) - y]^2 p(\boldsymbol{x}, y)d\boldsymbol{x}dy$$

But, $\mathcal{D}$ is a random sample from the following joint distribution

$$\mathcal{D} \sim P(\mathcal{D}) = \prod_{n=1}^{N} p(\boldsymbol{x}_n, y_n)$$

So, $h_{\mathcal{D}}(x)$ and $R[h_{\mathcal{D}}(x)]$ are also random w.r.t. $P(\mathcal{D})$

How can we disentangle the impact of the random sample $\mathcal{D}$ when assessing the quality of $h_{\mathcal{D}}(\cdot)$?

Average over the distribution of the training data

Averaged risk

$$\mathbb{E}_{\mathcal{D}} R[h_{\mathcal{D}}(\boldsymbol{x})] = \int_{\mathcal{D}} \int_{\boldsymbol{x}} \int_{y} [h_{\mathcal{D}}(\boldsymbol{x}) - y]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy \; P(\mathcal{D}) d\mathcal{D}$$

Average over the distribution of the training data

Averaged risk

$$\mathbb{E}_{\mathcal{D}} R[h_{\mathcal{D}}(\boldsymbol{x})] = \int_{\mathcal{D}} \int_{\boldsymbol{x}} \int_{y} [h_{\mathcal{D}}(\boldsymbol{x}) - y]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy \; P(\mathcal{D}) d\mathcal{D}$$

Namely, the randomness with respect to $\mathcal{D}$ is marginalized out.

Average over the distribution of the training data

Averaged risk

$$\mathbb{E}_{\mathcal{D}} R[h_{\mathcal{D}}(\boldsymbol{x})] = \int_{\mathcal{D}} \int_{\boldsymbol{x}} \int_{y} [h_{\mathcal{D}}(\boldsymbol{x}) - y]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy \; P(\mathcal{D}) d\mathcal{D}$$

Namely, the randomness with respect to $\mathcal{D}$ is marginalized out.

Averaged prediction

$$\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) = \int_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) P(\mathcal{D}) d\mathcal{D}$$

Namely, if we have seen many training datasets, we predict with the average of our trained models learned on each training dataset.

Interpreting Averaged risk

We can add and subtract averaged prediction from averaged risk

$$\mathbb{E}_{\mathcal{D}} R[h_{\mathcal{D}}(\boldsymbol{x})] = \underbrace{\int_{\mathcal{D}} \int_{\boldsymbol{x}} \int_{y} [h_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x})]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy \; P(\mathcal{D}) d\mathcal{D}}_{\text{VARIANCE}}$$

Interpreting Averaged risk

We can add and subtract averaged prediction from averaged risk

$$\mathbb{E}_{\mathcal{D}} R[h_{\mathcal{D}}(\boldsymbol{x})] = \underbrace{\int_{\mathcal{D}} \int_{\boldsymbol{x}} \int_{y} [h_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x})]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy \; P(\mathcal{D}) d\mathcal{D}}_{\text{VARIANCE}}$$

$$+ \underbrace{\int_{\boldsymbol{x}} \int_{y} [\mathbb{E}_{\mathcal{D}} h_{\mathcal{D}}(\boldsymbol{x}) - \mathbb{E}_{y}[y]]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy}_{\text{BIAS}^2}$$

$$+ \underbrace{\int_{\boldsymbol{x}} \int_{y} [\mathbb{E}_{y}[y] - y]^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy}_{\text{NOISE}}$$

Understanding the 3 terms

$$\mathbb{E}_{\mathcal{D}} R[h_{\mathcal{D}}(\boldsymbol{x})] = \text{VARIANCE} + \text{BIAS}^2 + \text{NOISE}$$

How can we reduce noise?

Understanding the 3 terms

$$\mathbb{E}_{\mathcal{D}} R[h_{\mathcal{D}}(\boldsymbol{x})] = \text{VARIANCE} + \text{BIAS}^2 + \text{NOISE}$$

How can we reduce noise?

- There is *nothing* we can do as it depends only on $p(\boldsymbol{x}, y)$

How can we reduce the variance?

Understanding the 3 terms

$$\mathbb{E}_{\mathcal{D}} R[h_{\mathcal{D}}(\boldsymbol{x})] = \text{VARIANCE} + \text{BIAS}^2 + \text{NOISE}$$

How can we reduce noise?

- There is *nothing* we can do as it depends only on $p(\boldsymbol{x}, y)$

How can we reduce the variance?

- Use a lot of data (ie, increase the size of $\mathcal{D}$)
- Use a simple $h(\cdot)$, e.g., $h(\boldsymbol{x}) = \text{const}$

How can we reduce bias?

Understanding the 3 terms

$$\mathbb{E}_{\mathcal{D}} R[h_{\mathcal{D}}(\boldsymbol{x})] = \text{VARIANCE} + \text{BIAS}^2 + \text{NOISE}$$

How can we reduce noise?

- There is *nothing* we can do as it depends only on $p(\boldsymbol{x}, y)$

How can we reduce the variance?

- Use a lot of data (ie, increase the size of $\mathcal{D}$)
- Use a simple $h(\cdot)$, e.g., $h(\boldsymbol{x}) = \text{const}$

How can we reduce bias?

- Use more complex models
- But, increased flexibility can increase the VARIANCE term

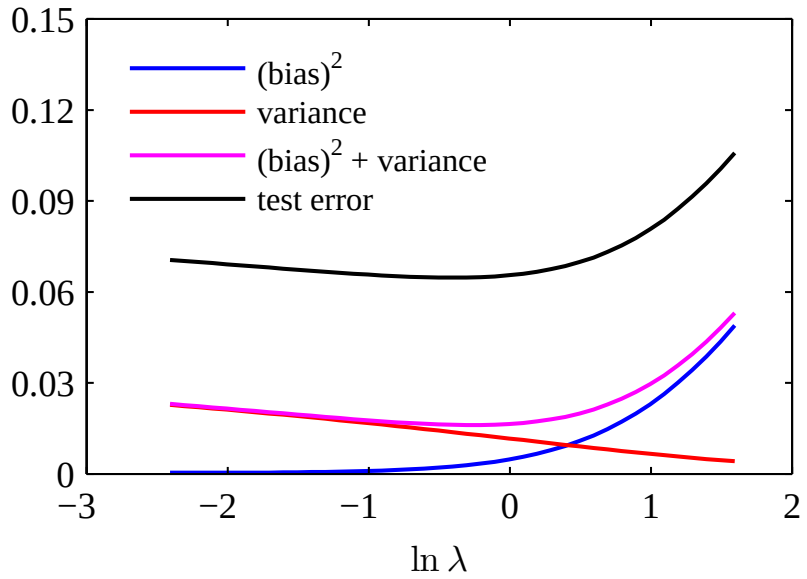
Bias/variance tradeoff

Error decomposes into 3 terms

$$\mathbb{E}_{\mathcal{D}} R[h_{\mathcal{D}}(\boldsymbol{x})] = \text{VARIANCE} + \text{BIAS}^2 + \text{NOISE}$$

The first and second terms are inherently in conflict

If we can compute all terms analytically, they will look like this



How can we perform nonlinear prediction without specifying nonlinear basis functions?

How can we perform nonlinear prediction without specifying nonlinear basis functions?

Definition of kernel function: a (positive semidefinite) 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 $\boldsymbol{\phi}(\cdot)$.

How can we perform nonlinear prediction without specifying nonlinear basis functions?

Definition of kernel function: a (positive semidefinite) 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 $\boldsymbol{\phi}(\cdot)$.

Examples we have seen

$$k(\boldsymbol{x}_m, \boldsymbol{x}_n) = (\boldsymbol{x}_m^{\mathrm{T}} \boldsymbol{x}_n)^2$$
$$k(\boldsymbol{x}_m, \boldsymbol{x}_n) = e^{-\|\boldsymbol{x}_m - \boldsymbol{x}_n\|_2^2 / 2\sigma^2}$$

Conditions for being a positive semidefinite kernel function

Mercer theorem (loosely), a bivariate function $k(\cdot, \cdot)$ is a positive semidefinite 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. We also refer $k(\cdot, \cdot)$ as a positive semidefinite kernel.

Why $\|\boldsymbol{x}_m - \boldsymbol{x}_n\|_2^2$ is not a positive semidefinite kernel?

Use the definition of positive semidefinite kernel function. We choose $N = 2$, and compute the matrix

$$\boldsymbol{K} = \begin{pmatrix} 0 & \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2^2 \\ \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2^2 & 0 \end{pmatrix}$$

Why $\|\boldsymbol{x}_m - \boldsymbol{x}_n\|_2^2$ is not a positive semidefinite kernel?

Use the definition of positive semidefinite kernel function. We choose $N = 2$, and compute the matrix

$$\boldsymbol{K} = \begin{pmatrix} 0 & \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2^2 \\ \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2^2 & 0 \end{pmatrix}$$

- PSD matrices have only non-negative eigenvalues
- This matrix has both *negative* and positive eigenvalues
- *Trace* of a matrix equals the sum of the diagonal elements, and also equals to the sum of the matrix's eigenvalues
  - In our case, the trace is zero
- Could also see this via determinant of $\boldsymbol{K}$

Flashback: Why would we want to use kernel functions?

Flashback: Why would we want to use kernel functions?

without specifying $\phi(\cdot)$, the kernel 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 exactly the same as

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

$$= \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_1) & \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_2) & \cdots & \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_N) \\ \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_1) & \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_2) & \cdots & \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_N) \\ \cdots & \cdots & \cdots & \cdots \\ \boldsymbol{\phi}(\boldsymbol{x}_N)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_1) & \boldsymbol{\phi}(\boldsymbol{x}_N)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_2) & \cdots & \boldsymbol{\phi}(\boldsymbol{x}_N)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_N) \end{pmatrix}$$

'Kernel trick'

- Many learning methods rely on training and test data only in the form of *inner products*, e.g., regularized least squares, nearest neighbors
- We can use a kernel function to introduce nonlinearity, i.e., *"kernelizing"* the methods
- Last lecture: Ridge Regression, Nearest Neighbor Classification
- Next lecture: Support Vector Machines

Kernelized nearest neighbors classifier (NNC)

Fundamental quantity is (squared) distance between two data points

$$d(\boldsymbol{x}_m, \boldsymbol{x}_n) = \|\boldsymbol{x}_m - \boldsymbol{x}_n\|_2^2 = \boldsymbol{x}_m^{\mathrm{T}}\boldsymbol{x}_m + \boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{x}_n - 2\boldsymbol{x}_m^{\mathrm{T}}\boldsymbol{x}_n$$

Kernelized nearest neighbors classifier (NNC)

Fundamental quantity is (squared) distance between two data points

$$d(\boldsymbol{x}_m, \boldsymbol{x}_n) = \|\boldsymbol{x}_m - \boldsymbol{x}_n\|_2^2 = \boldsymbol{x}_m^{\mathrm{T}}\boldsymbol{x}_m + \boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{x}_n - 2\boldsymbol{x}_m^{\mathrm{T}}\boldsymbol{x}_n$$

Can replace dot products by a kernel function $k(\cdot, \cdot)$:

$$d^{\mathrm{KERNEL}}(\boldsymbol{x}_m, \boldsymbol{x}_n) = k(\boldsymbol{x}_m, \boldsymbol{x}_m) + k(\boldsymbol{x}_n, \boldsymbol{x}_n) - 2k(\boldsymbol{x}_m, \boldsymbol{x}_n)$$

Kernelized nearest neighbors classifier (NNC)

Fundamental quantity is (squared) distance between two data points

$$d(\boldsymbol{x}_m, \boldsymbol{x}_n) = \|\boldsymbol{x}_m - \boldsymbol{x}_n\|_2^2 = \boldsymbol{x}_m^{\mathrm{T}}\boldsymbol{x}_m + \boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{x}_n - 2\boldsymbol{x}_m^{\mathrm{T}}\boldsymbol{x}_n$$

Can replace dot products by a kernel function $k(\cdot, \cdot)$:

$$d^{\mathrm{KERNEL}}(\boldsymbol{x}_m, \boldsymbol{x}_n) = k(\boldsymbol{x}_m, \boldsymbol{x}_m) + k(\boldsymbol{x}_n, \boldsymbol{x}_n) - 2k(\boldsymbol{x}_m, \boldsymbol{x}_n) = d(\boldsymbol{\phi}(\boldsymbol{x}_m), \boldsymbol{\phi}(\boldsymbol{x}_n))$$

- d^{KERNEL} equivalent to distance between $\boldsymbol{\phi}(\boldsymbol{x}_m)$ and $\boldsymbol{\phi}(\boldsymbol{x}_n)$ where $\boldsymbol{\phi}(\cdot)$ is the nonlinear mapping implied by the kernel function
- The nearest neighbor of a test point $\boldsymbol{x}$ is found via

$$\arg\min_n d^{\mathrm{KERNEL}}(\boldsymbol{x}, \boldsymbol{x}_n)$$

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.

Outline

Support vector machines

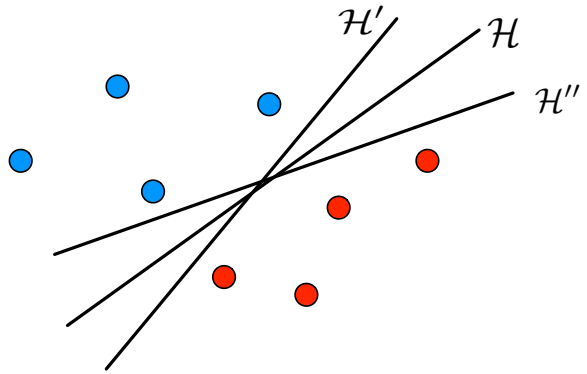
- One of the most commonly used classification algorithms
- This lecture: Geometric motivation
- Next lecture: Hinge-loss formulation, Kernel SVM

Intuition: where to put the decision boundary?

- Consider a *separable* training dataset (e.g., with two features)

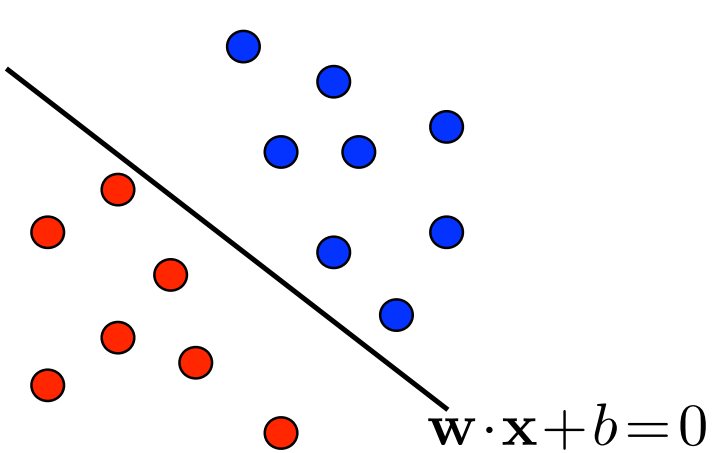
Intuition: where to put the decision boundary?

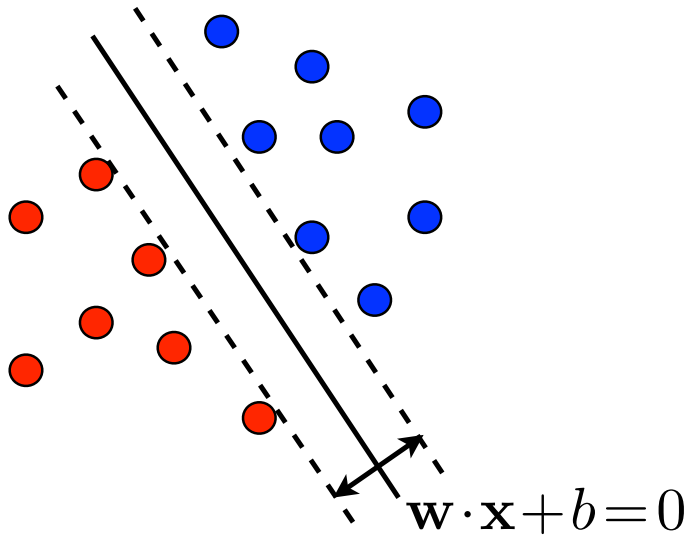
- Consider a *separable* training dataset (e.g., with two features)
- There are an *infinite* number of decision boundaries $\mathcal{H} : \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b = 0$!



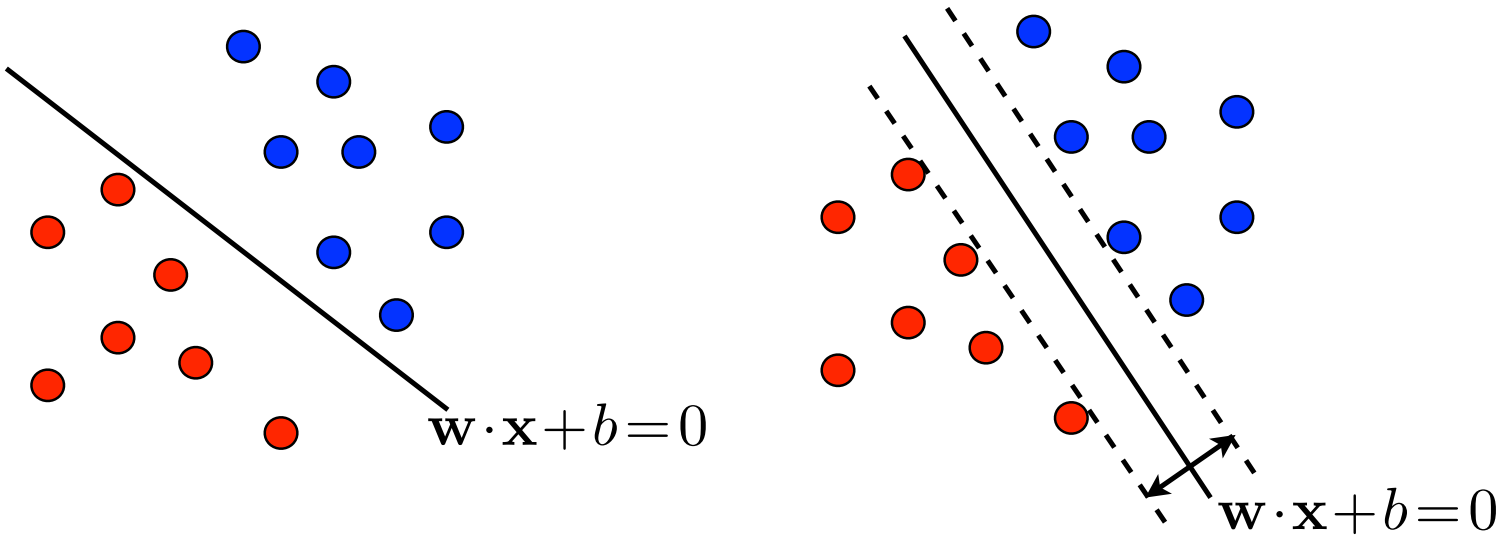
- Which one should we pick?

Intuition: where to put the decision boundary?





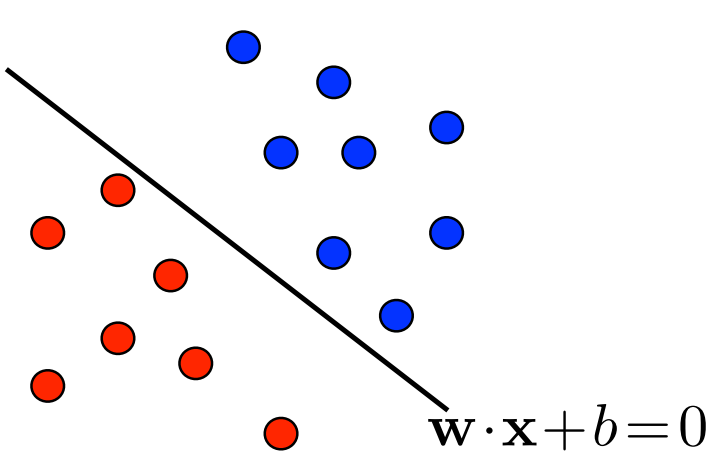
Intuition: where to put the decision boundary?

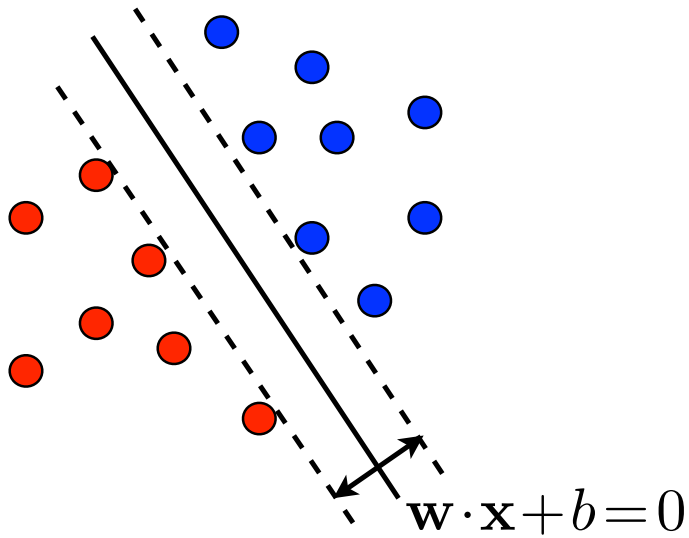


Idea: Find a decision boundary in the '*middle*' of the two classes. Ideally, we want a decision boundary that:

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

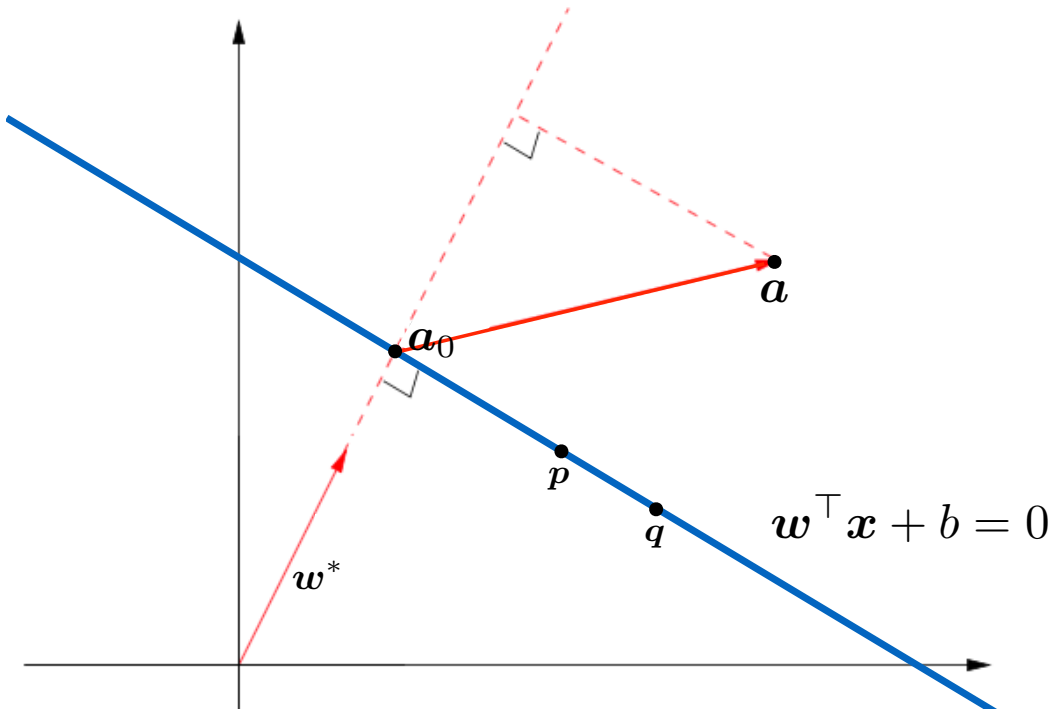
What is a hyperplane?





- 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

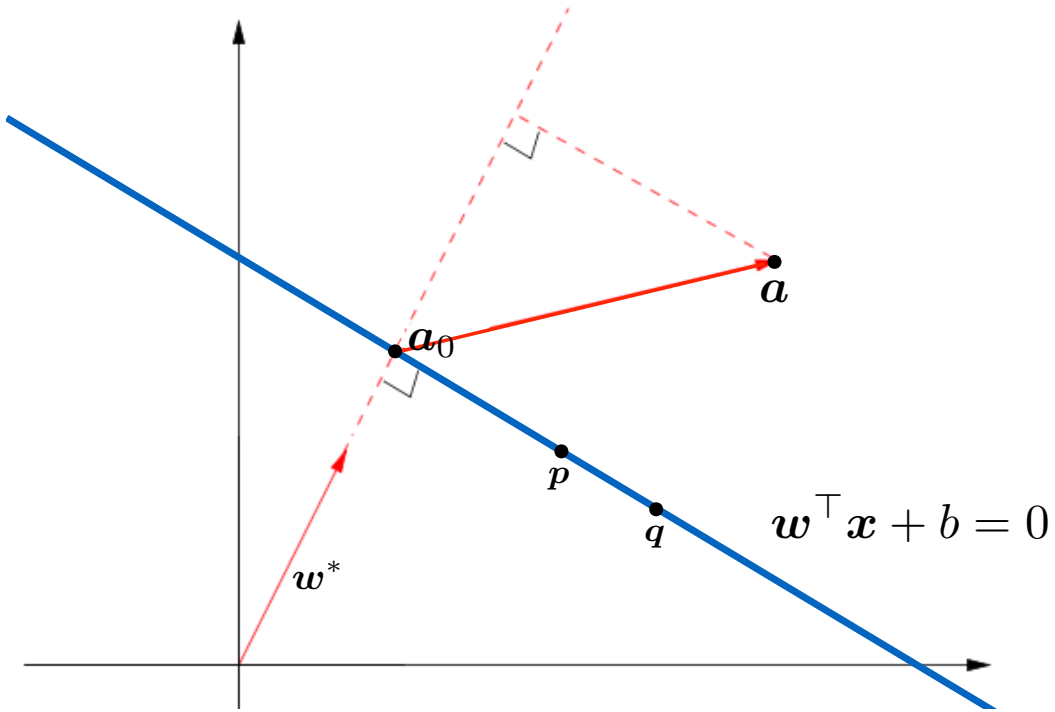
Properties of hyperplanes



Why is w normal to this hyperplane (or line in 2D)?

- If two points, p and q are both on the line, then

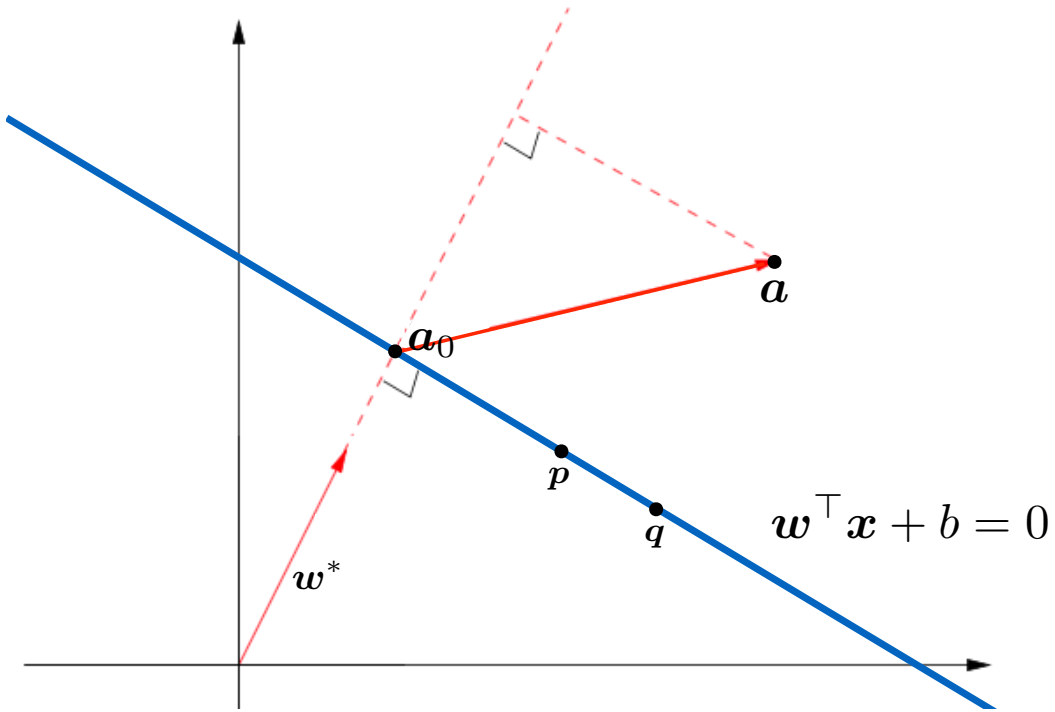
Properties of hyperplanes



Why is w normal to this hyperplane (or line in 2D)?

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

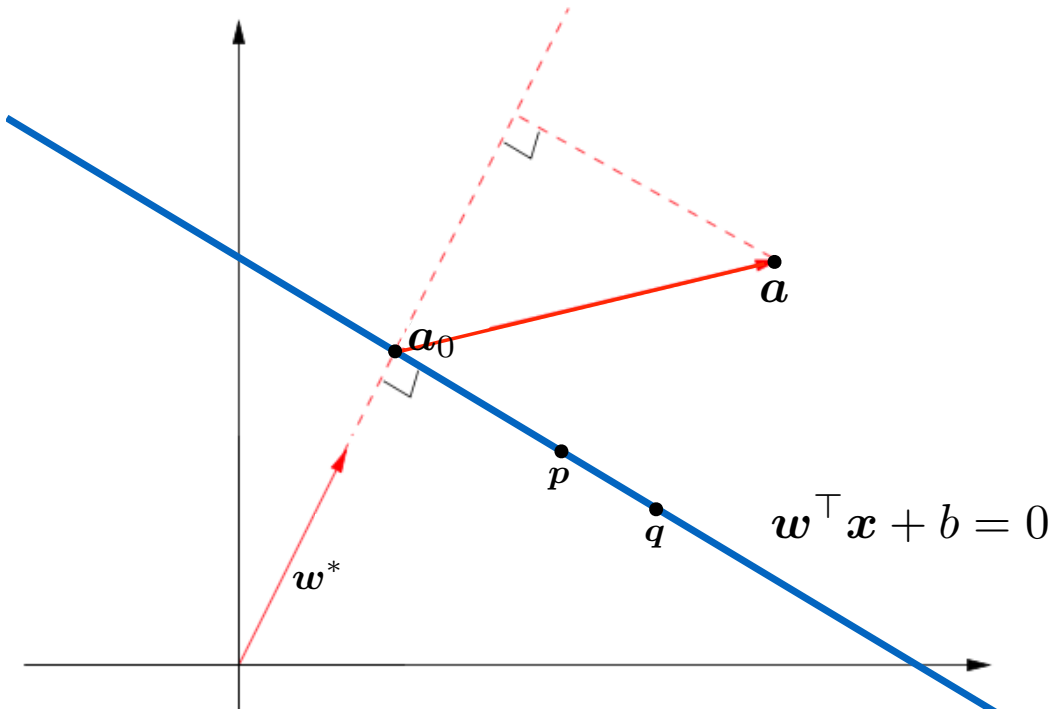
Properties of hyperplanes



Why is w normal to this hyperplane (or line in 2D)?

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

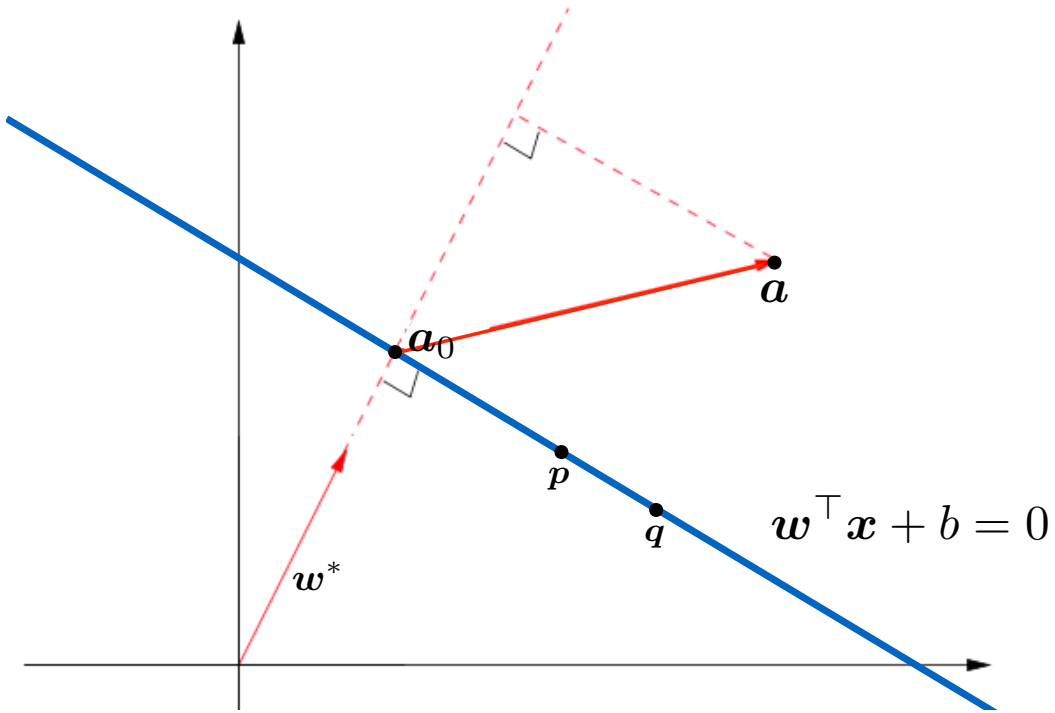
Properties of hyperplanes



Why is w normal to this hyperplane (or line in 2D)?

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

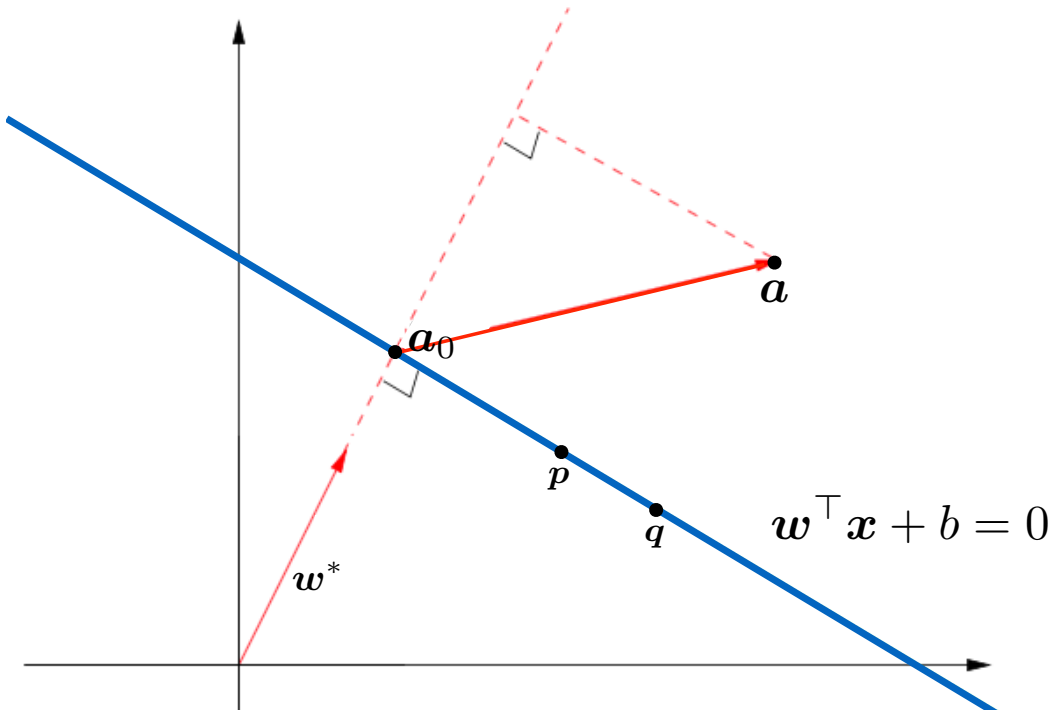
Properties of hyperplanes



How to compute signed distance from a to the hyperplane?

- We want to find distance between $\boldsymbol{a}$ and line in direction of $\boldsymbol{w}^*$

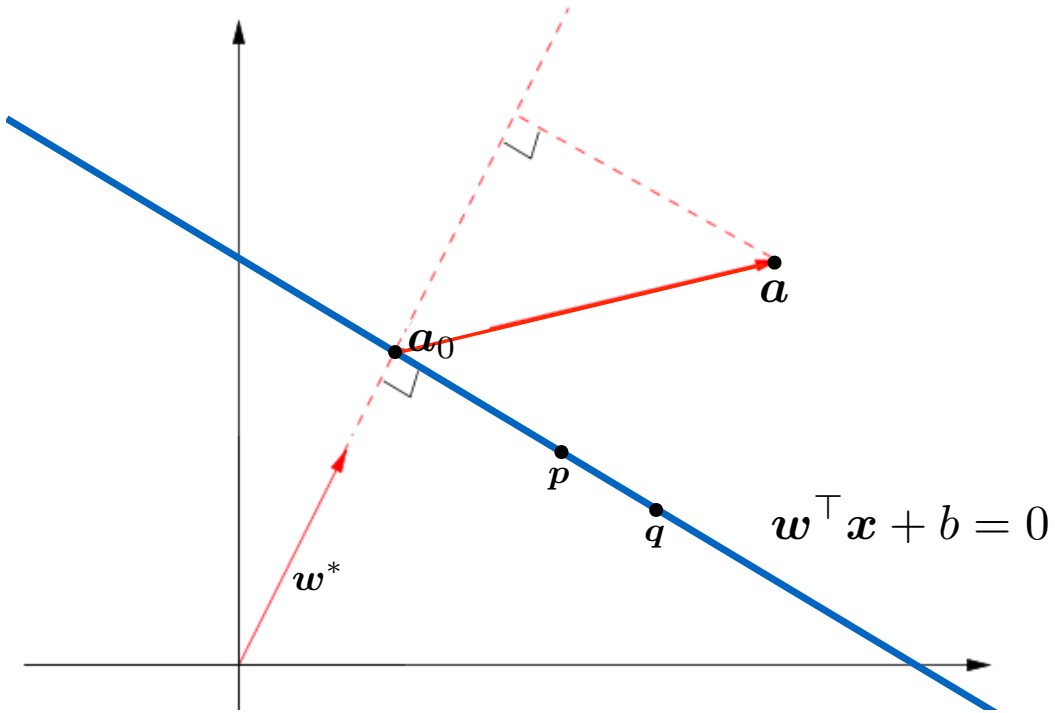
Properties of hyperplanes



How to compute signed distance from a to the hyperplane?

- We want to find distance between $\boldsymbol{a}$ and line in direction of $\boldsymbol{w}^*$
- If we define point $\boldsymbol{a}_0$ on the line, then this distance corresponds to length of $\boldsymbol{a} - \boldsymbol{a}_0$ in direction of $\boldsymbol{w}^*$, which equals $\boldsymbol{w}^{*\top}(\boldsymbol{a} - \boldsymbol{a}_0)$

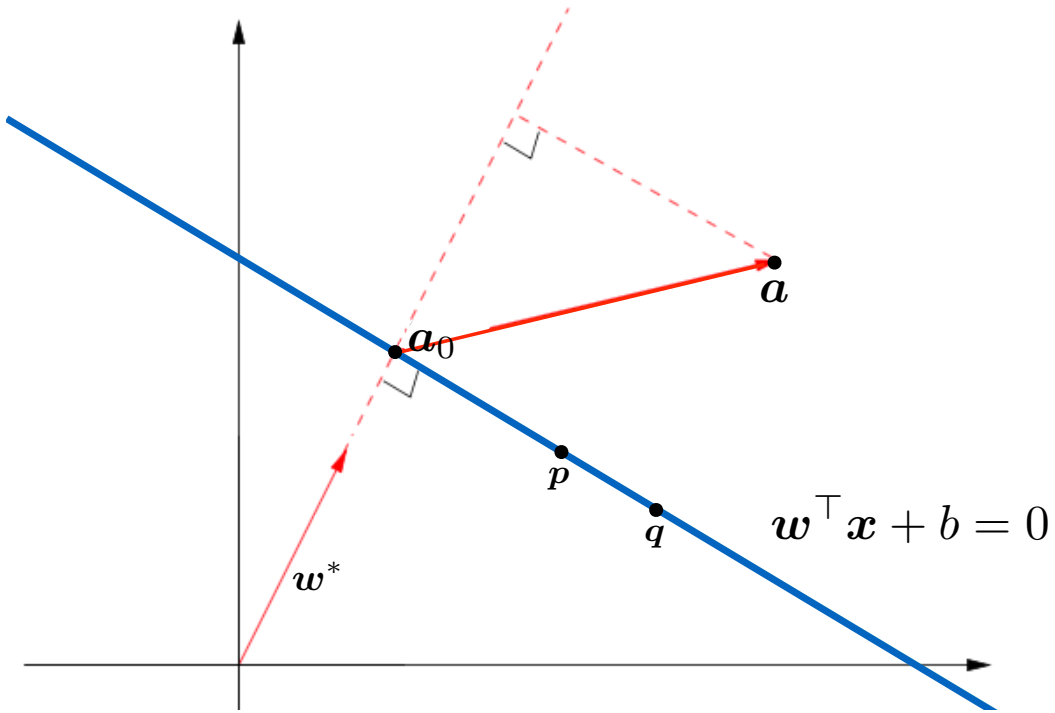
Properties of hyperplanes



How to compute signed distance from a to the hyperplane?

- We want to find distance between $\boldsymbol{a}$ and line in direction of $\boldsymbol{w}^*$
- If we define point $\boldsymbol{a}_0$ on the line, then this distance corresponds to length of $\boldsymbol{a} - \boldsymbol{a}_0$ in direction of $\boldsymbol{w}^*$, which equals $\boldsymbol{w}^{*\top}(\boldsymbol{a} - \boldsymbol{a}_0)$
- Since $\boldsymbol{w}^\top \boldsymbol{a}_0 = -b$,

Properties of hyperplanes



How to compute signed distance from a to the hyperplane?

- We want to find distance between $\boldsymbol{a}$ and line in direction of $\boldsymbol{w}^*$
- If we define point $\boldsymbol{a}_0$ on the line, then this distance corresponds to length of $\boldsymbol{a} - \boldsymbol{a}_0$ in direction of $\boldsymbol{w}^*$, which equals ${\boldsymbol{w}^*}^\top (\boldsymbol{a} - \boldsymbol{a}_0)$
- 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.

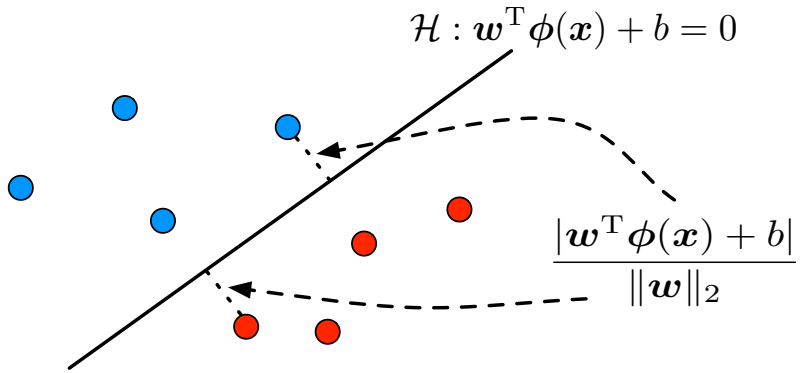
Namely, $(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b)$ and $\boldsymbol{x}$'s label y must have the same sign, so:

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

Optimizing the Margin

Margin Smallest distance between the hyperplane and all training points

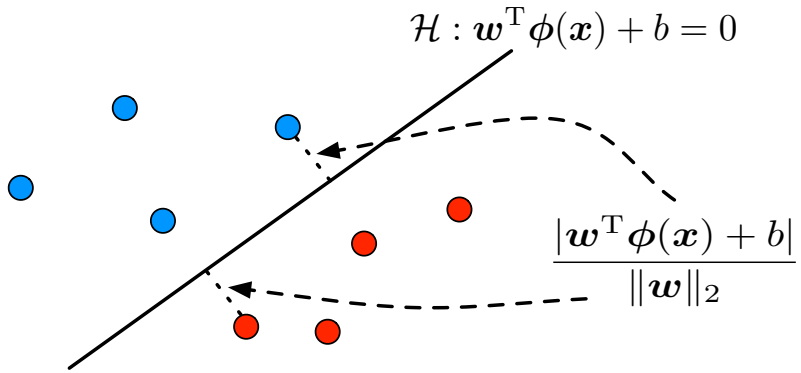
$$\text{MARGIN}(\boldsymbol{w}, b) = \min_n \frac{y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]}{\|\boldsymbol{w}\|_2}$$



Optimizing the Margin

Margin Smallest distance between the hyperplane and all training points

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

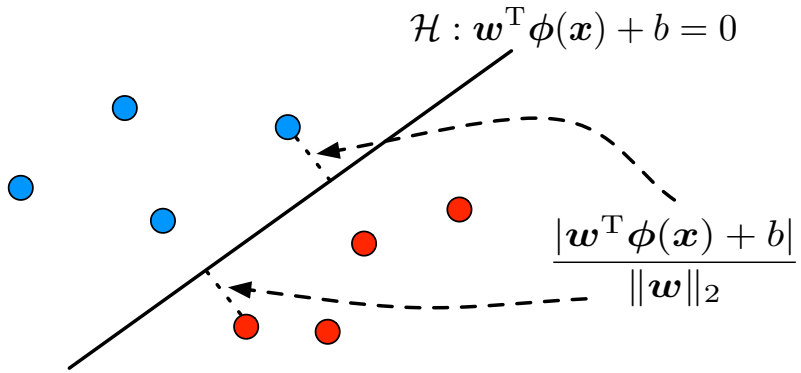


How should we pick (w, b) based on its margin?

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 to *maximize* the margin!

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

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 $\text{MARGIN}(\boldsymbol{w}, b)$ equation, and we have:

$$\text{MARGIN}(\boldsymbol{w}, b) = \frac{1}{\|\boldsymbol{w}\|_2}$$

Rescaled Margin

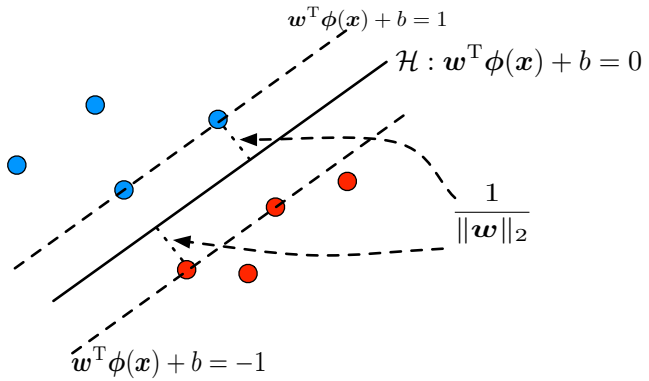
We can further constrain the problem by scaling $(\boldsymbol{w}, b)$ such that

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

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

$$\text{MARGIN}(\boldsymbol{w}, b) = \frac{1}{\|\boldsymbol{w}\|_2}$$

Hence the points closest to the decision boundary are at distance 1!



SVM: max margin formulation for separable data

Assuming separable training data, we thus want to solve:

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

This is equivalent to

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

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

SVM for non-separable data

SVM formulation for separable data

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

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

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

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

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

Constraints in non-separable setting

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

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

SVM for non-separable data

Constraints in separable setting

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

Constraints in non-separable setting

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

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

- 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:

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

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:

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

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:

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

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?

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

- This is a *convex quadratic program*: 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?

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_n \xi_n$$

$$\text{s.t.} \quad y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1 - \xi_n, \quad \forall \ \ n$$

$$\xi_n \geq 0, \quad \forall\ n$$

- This is a *convex quadratic program*: 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, e.g., Matlab's `quadprog()` function.

How to solve this problem?

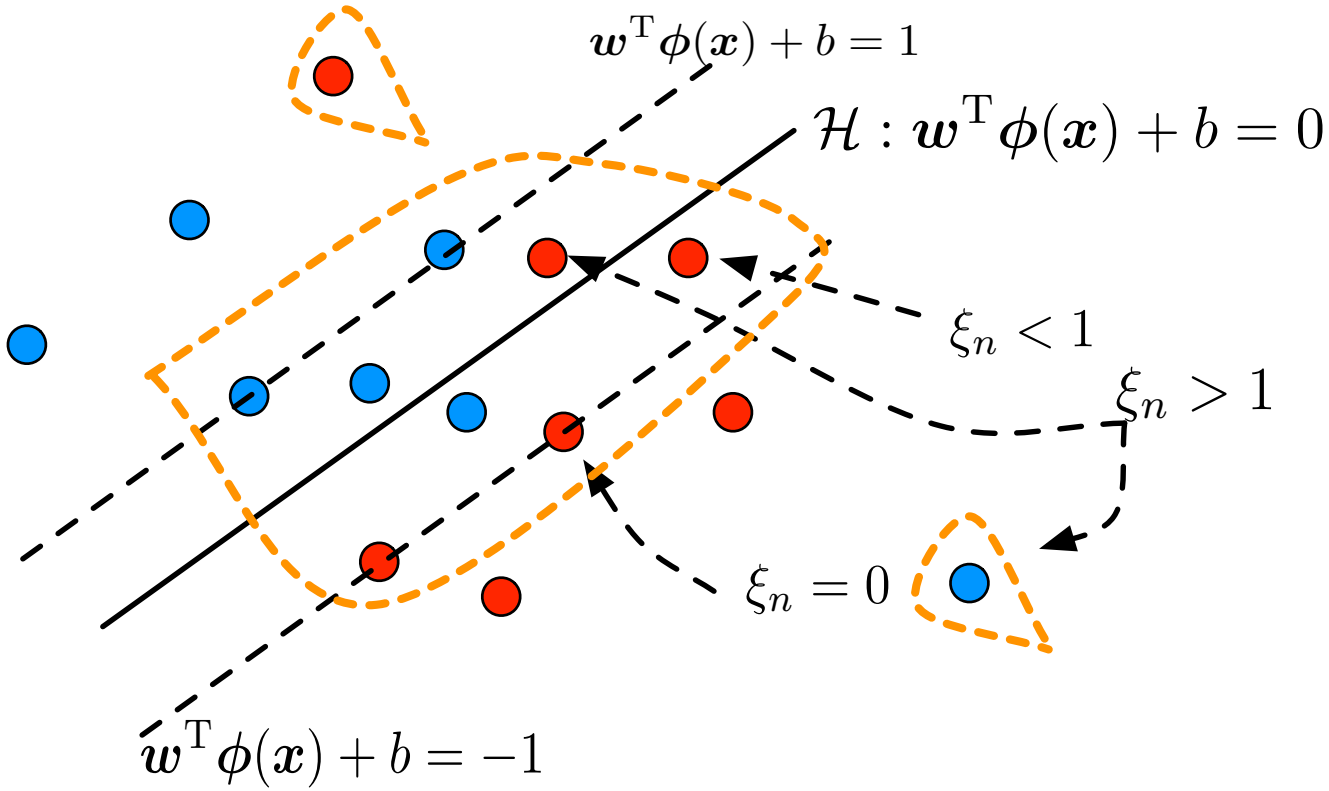
$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_n \xi_n \\ \text{s.t.} \quad & y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1 - \xi_n, \quad \forall \quad n \\ & \xi_n \geq 0, \quad \forall\ n \end{aligned}$$

- This is a *convex quadratic program*: 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, e.g., Matlab's `quadprog()` function.
- 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 solution is only determined by a subset of the training samples (as we will see in more detail in the next lecture)
- These samples 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