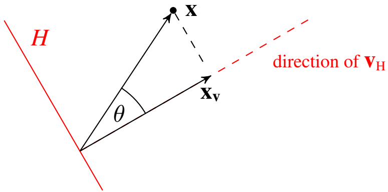
WHY *minimize* $\|\mathbf{v}_{\mathrm{H}}\|$?

We can project a vector $\mathbf{x}$ (think: data point) onto the direction of $\mathbf{v}_{\mathrm{H}}$ and obtain a vector $\mathbf{x}_{\mathbf{v}}$.



- If H has no offset ($c = 0$), the Euclidean distance of $\mathbf{x}$ from H is

$$d(\mathbf{x}, H) = \|\mathbf{x}_{\mathbf{v}}\| = \cos\theta \cdot \|\mathbf{x}\| \ .$$

  It does not depend on the length of $\mathbf{v}_{\mathrm{H}}$.
- The scalar product $\langle \mathbf{x}, \mathbf{v}_{\mathrm{H}} \rangle$ *does* increase if the length of $\mathbf{v}_{\mathrm{H}}$ increases.
- To compute the distance $\|\mathbf{x}_{\mathbf{v}}\|$ from $\langle \mathbf{x}, \mathbf{v}_{\mathrm{H}} \rangle$, we have to scale out $\|\mathbf{v}_{\mathrm{H}}\|$:

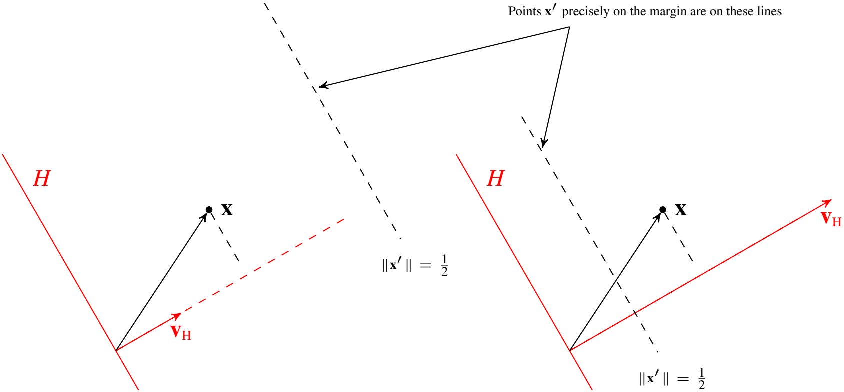
$$\|\mathbf{x}_{\mathbf{v}}\| = \cos\theta \cdot \|\mathbf{x}\| = \frac{\langle \mathbf{x}, \mathbf{v}_{\mathrm{H}} \rangle}{\|\mathbf{v}_{\mathrm{H}}\|}$$

WHY *minimize* $\|\mathbf{v}_{\mathrm{H}}\|$?

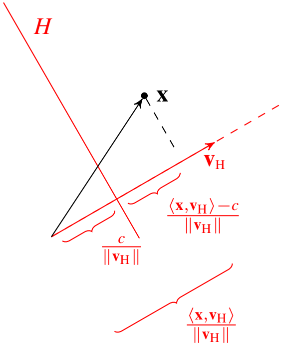
If we scale $\mathbf{v}_{\mathrm{H}}$ by α, we have to scale $\mathbf{x}$ by $1/\alpha$ to keep $\langle \mathbf{v}_{\mathrm{H}}, \mathbf{x} \rangle$ constant, e.g.:

$$1 = \langle \mathbf{v}_{\mathrm{H}}, \mathbf{x} \rangle = \left\langle \alpha \mathbf{v}_{\mathrm{H}}, \frac{1}{\alpha} \mathbf{x} \right\rangle .$$

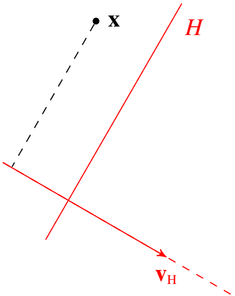
A point $\mathbf{x}'$ is precisely on the margin if $\langle \mathbf{x}', \mathbf{v}_{\mathrm{H}} \rangle = 1$.
Look at what happens if we scale $\mathbf{v}_{\mathrm{H}}$:



Distance With Offset



For an affine plane, we have to substract the offset.



The optimization algorithm can also rotate the vector $\mathbf{v}_{\mathrm{H}}$, which rotates the plane.

Soft-Margin Classifiers

Soft-margin classifiers are maximum-margin classifiers which permit some points to lie on the wrong side of the margin, or even of the hyperplane.

Motivation 1: Nonseparable data

SVMs are linear classifiers; without further modifications, they cannot be trained on a non-separable training data set.

Motivation 2: Robustness

- Recall: Location of SVM classifier depends on position of (possibly few) support vectors.
- Suppose we have two training samples (from the same joint distribution on (X, Y)) and train an SVM on each.
- If locations of support vectors vary significantly between samples, SVM estimate of $\mathbf{v}_{\mathrm{H}}$ is "brittle" (depends too much on small variations in training data). $\longrightarrow$ Bad generalization properties.
- Methods which are not susceptible to small variations in the data are often referred to as **robust**.

Slack Variables

Idea

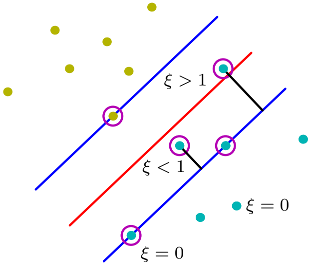
Permit training data to cross the margin, but impose cost which increases the further beyond the margin we are.

Formalization

We replace the training rule $\tilde{y}_i(\langle \mathbf{v}_{\mathrm{H}}, \tilde{\mathbf{x}}_i \rangle - c) \geq 1$ by

$$\tilde{y}_i(\langle \mathbf{v}_{\mathrm{H}}, \tilde{\mathbf{x}}_i \rangle - c) \geq 1 - \xi_i$$

with $\xi_i \geq 0$. The variables ξ_i are called **slack variables**.



Soft-margin optimization problem

$$\begin{aligned} \min_{\mathbf{v}_{\mathrm{H}},c} \quad & \|\mathbf{v}_{\mathrm{H}}\|^2 + \gamma \sum_{i=1}^{n} \xi^2 \\ \text{s.t.} \quad & \tilde{y}_i(\langle \mathbf{v}_{\mathrm{H}}, \tilde{\mathbf{x}}_i \rangle - c) \geq 1 - \xi_i \quad \text{for } i = 1, \ldots, n \\ & \xi_i \geq 0, \quad \text{for } i = 1, \ldots, n \end{aligned}$$

The training algorithm now has a **parameter** $\gamma > 0$ for which we have to choose a "good" value. γ is usually set by a method called *cross validation* (discussed later). Its value is fixed before we start the optimization.

Role of γ

- Specifies the "cost" of allowing a point on the wrong side.
- If γ is very small, many points may end up beyond the margin boundary.
- For $\gamma \to \infty$, we recover the original SVM.

SOFT-MARGIN SVM

Soft-margin dual problem

The slack variables vanish in the dual problem.

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^n} \quad W(\boldsymbol{\alpha}) := \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i\alpha_j\tilde{y}_i\tilde{y}_j\left(\langle \tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j \rangle + \frac{1}{\gamma}\mathbb{I}\{i=j\}\right)$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \tilde{y}_i\alpha_i = 0$$

$$\alpha_i \geq 0 \quad \text{for } i = 1,\ldots,n$$

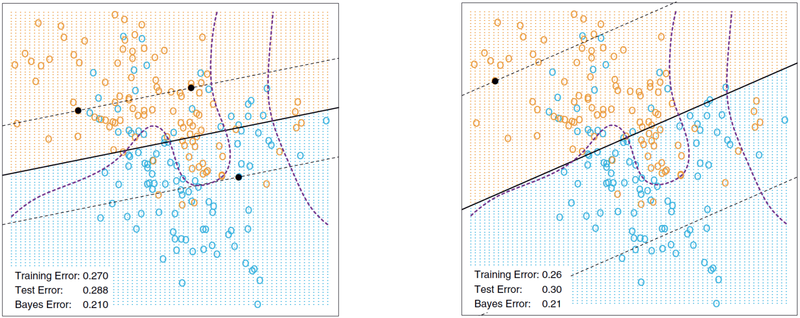
Soft-margin classifier

The classifier looks exactly as for the original SVM:

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{n} \tilde{y}_i\alpha_i^* \langle \tilde{\mathbf{x}}_i, \mathbf{x}\rangle - c\right)$$

Note: Each point on wrong side of the margin is an additional support vector ($\alpha_i^* \neq 0$), so the ratio of support vectors can be substantial when classes overlap.

INFLUENCE OF MARGIN PARAMETER



$\gamma = 100000$ $\qquad\qquad$ $\gamma = 0.01$

Changing γ significantly changes the classifier (note how the slope changes in the figures). We need a method to select an appropriate value of γ, in other words: to learn γ from data.

TOOLS: OPTIMIZATION METHODS

Terminology

An **optimization problem** for a given function $f : \mathbb{R}^d \to \mathbb{R}$ is a problem of the form

$$\min_{\mathbf{x}} f(\mathbf{x})$$

which we read as "find $\mathbf{x}_0 = \arg\min_{\mathbf{x}} f(\mathbf{x})$".
A **constrained optimization problem** adds additional requirements on **x**,

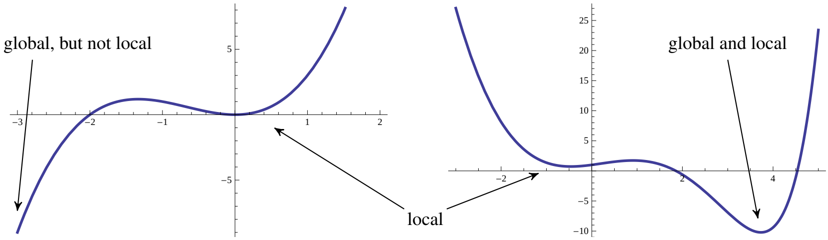
$$\begin{aligned} &\min_{\mathbf{x}} \; f(\mathbf{x}) \\ \text{subject to} \quad &\mathbf{x} \in G , \end{aligned}$$

where $G \subset \mathbb{R}^d$ is called the **feasible set**. The set G is often defined by equations, e.g.

$$\begin{aligned} &\min_{\mathbf{x}} \; f(\mathbf{x}) \\ \text{subject to} \quad &g(\mathbf{x}) \geq 0 \end{aligned}$$

The equation g is called a **constraint**.

Types of Minima



Local and global minima

A minimum of f at x is called:

- **Global** if f assumes no smaller value on its domain.
- **Local** if there is some open neighborhood U of x such that $f(x)$ is a global minimum of f restricted to U.

Analytic criteria for local minima

Recall that $\mathbf{x}$ is a local minimum of f if

$$f'(\mathbf{x}) = 0 \qquad \text{and} \qquad f''(\mathbf{x}) > 0 \; .$$

In $\mathbb{R}^d$,

$$\nabla f(\mathbf{x}) = 0 \qquad \text{and} \qquad H_f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_i \partial x_j}(\mathbf{x}) \right)_{i,j=1,\dots,n} \text{ positive definite.}$$

The $d \times d$-matrix $H_f(\mathbf{x})$ is called the **Hessian matrix** of f at $\mathbf{x}$.

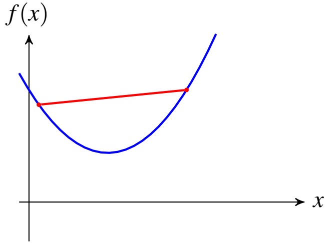
Numerical methods

All numerical minimization methods perform roughly the same steps:

- Start with some point x_0.
- Our goal is to find a sequence $x_0, \dots, x_m$ such that $f(x_m)$ is a minimum.
- At a given point x_n, compute properties of f (such as $f'(x_n)$ and $f''(x_n)$).
- Based on these values, choose the next point x_{n+1}.

The information $f'(x_n), f''(x_n)$ etc is always *local at* x_n, and we can only decide whether a point is a local minimum, not whether it is global.

Convex Functions



Definition

A function f is **convex** if every line segment between function values lies above the graph of f.

Analytic criterion

A twice differentiable function is convex if $f''(x) \geq 0$ (or $H_f(\mathbf{x})$ positive semidefinite) for all $\mathbf{x}$.

Implications for optimization

If f is convex, then:

- $f'(x) = 0$ is a sufficient criterion for a minimum.
- Local minima are global.
- If f is **strictly convex** ($f'' > 0$ or H_f positive definite), there is only one minimum (which is both gobal and local).

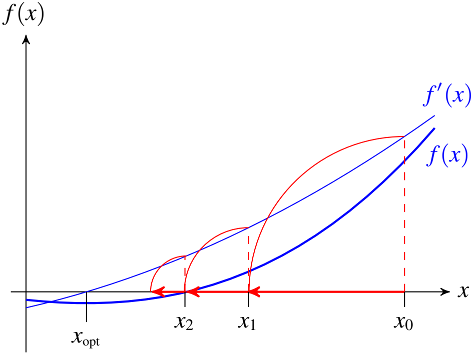
Algorithm

Gradient descent searches for a minimum of f.

1. Start with some point $x \in \mathbb{R}$ and fix a precision $\varepsilon > 0$.
2. Repeat for $n = 1, 2, \ldots$

$$x_{n+1} := x_n - f'(x_n)$$

3. Terminate when $|f'(x_n)| < \varepsilon$.



Newton's Method: Roots

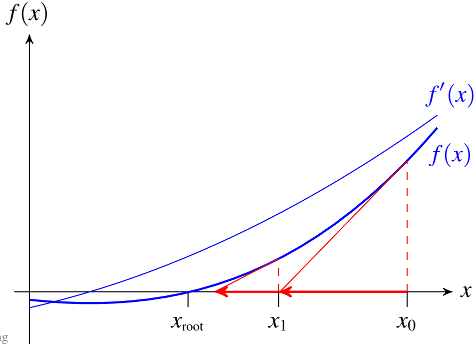
Algorithm

Newton's method searches for a **root** of f, i.e. it solves the equation $f(\mathbf{x}) = 0$.

1. Start with some point $x \in \mathbb{R}$ and fix a precision $\varepsilon > 0$.
2. Repeat for $n = 1, 2, \ldots$

$$x_{n+1} := x_n - f(x_n)/f'(x_n)$$

3. Terminate when $|f(x_n)| < \varepsilon$.



Function evaluation

Most numerical evaluations of functions ($\sqrt{a}$, $\sin(a)$, $\exp(a)$, etc) are implemented using Newton's method. To evaluate g at a, we have to transform $x = g(a)$ into an equivalent equation of the form

$$f(x, a) = 0 .$$

We then fix a and solve for x using Newton's method for roots.

Example: Square root

To eveluate $g(a) = \sqrt{a}$, we can solve

$$f(x, a) = x^2 - a = 0 .$$

This is essentially how `sqrt()` is implemented in the standard C library.

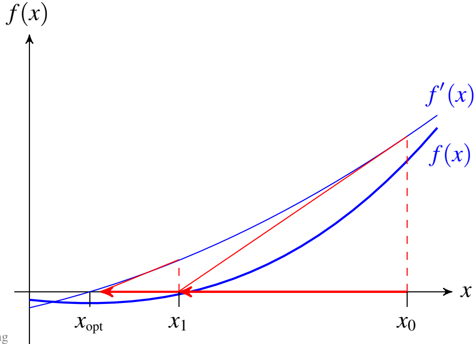
Algorithm

We can use Newton's method for minimization by applying it to solve $f'(\mathbf{x}) = 0$.

1. Start with some point $x \in \mathbb{R}$ and fix a precision $\varepsilon > 0$.
2. Repeat for $n = 1, 2, \ldots$

$$x_{n+1} := x_n - f'(x_n)/f''(x_n)$$

3. Terminate when $|f'(x_n)| < \varepsilon$.



Multiple Dimensions

In $\mathbb{R}^d$ we have to replace the derivatives by their vector space analogues.

Gradient descent

$$\mathbf{x}_{n+1} := \mathbf{x}_n - \nabla f(\mathbf{x}_n)$$

Newton's method for minima

$$\mathbf{x}_{n+1} := \mathbf{x}_n - H_f^{-1}(\mathbf{x}_n) \cdot \nabla f(\mathbf{x}_n)$$

The inverse of $H_f(\mathbf{x})$ exists only if the matrix is positive definite (not if it is only semidefinite), i.e. f has to be strictly convex.

The Hessian measures the curvature of f.

Effect of the Hessian

Multiplication by H_f^{-1} in general changes the direction of $\nabla f(\mathbf{x}_n)$. The correction takes into account how $\nabla f(\mathbf{x})$ changes away from $\mathbf{x}_n$, as estimated using the Hessian at $\mathbf{x}_n$.

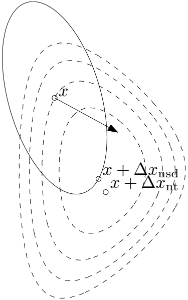


Figure: Arrow is ∇f, $x + \Delta x_{\text{nt}}$ is Newton step.