CLASSIFICATION

ASSUMPTIONS AND TERMINOLOGY

In a **classification problem**, we record measurements $\mathbf{x}_1, \mathbf{x}_2, \ldots$.

We assume:

1. All measurements can be represented as elements of a Euclidean $\mathbb{R}^d$.
2. Each $\mathbf{x}_i$ belongs to exactly one out of K categories, called **classes**. We express this using variables $y_i \in [K]$, called **class labels**:

$$y_i = k \quad \Leftrightarrow \quad \text{"}\mathbf{x}_i \text{ in class } k\text{"}$$

3. The classes are characterized by the (unknown!) joint distribution of (X, Y), whose density we denote $p(x, y)$. The conditional distribution with density $p(x|y = k)$ is called the **class-conditional distribution** of class k.
4. The only information available on the distribution p is a set of example measurements *with* labels,

$$(\tilde{\mathbf{x}}_1, \tilde{y}_1), \ldots, (\tilde{\mathbf{x}}_n, \tilde{y}_n) \ ,$$

called the **training data**.

Definition

A **classifier** is a function

$$f : \mathbb{R}^d \longrightarrow [K] \ ,$$

i.e. a function whose argument is a measurement and whose output is a class label.

Learning task

Using the training data, we have to estimate a good classifier. This estimation procedure is also called **training**.

A good classifier should generalize well to new data. Ideally, we would like it to perform with high accuracy on data sampled from p, but all we know about p is the training data.

Simplifying assumption

We first develop methods for the two-class case (K=2), which is also called **binary classification**. In this case, we use the notation

$$y \in \{-1, +1\} \qquad \text{instead of} \qquad y \in \{1, 2\}$$

Supervised vs. unsupervised

Fitting a model using labeled data is called **supervised learning**. Fitting a model when only $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n$ are available, but no labels, is called **unsupervised learning**.

Types of supervised learning methods

- Classification: Labels are discrete, and we estimate a classifier $f : \mathbb{R}^d \longrightarrow [K]$,
- Regression: Labels are real-valued ($y \in \mathbb{R}$), and we estimate a continuous function $f : \mathbb{R}^d \longrightarrow \mathbb{R}$. This functions is called a **regressor**.

A Very Simple Classifier

Algorithm

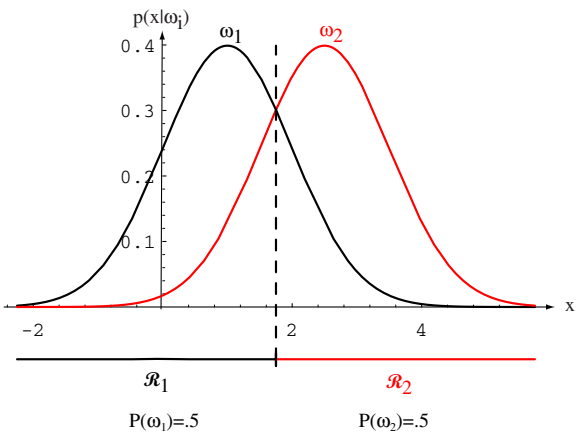
1. On training data, fit a Gaussian into each class (by MLE).
   Result: Densities $g(\mathbf{x}|\mu_\oplus, \Sigma_\oplus)$ and $g(\mathbf{x}|\mu_\ominus, \Sigma_\ominus)$
2. Classify test point according to which density assigns larger value:

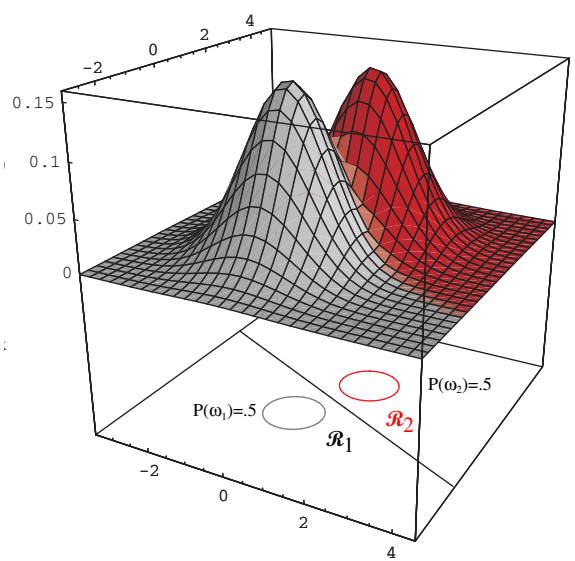
$$y_i := \begin{cases} +1 & \text{if } g(\mathbf{x}_i|\mu_\oplus, \Sigma_\oplus) > g(\mathbf{x}_i|\mu_\ominus, \Sigma_\ominus) \\ -1 & \text{otherwise} \end{cases}$$

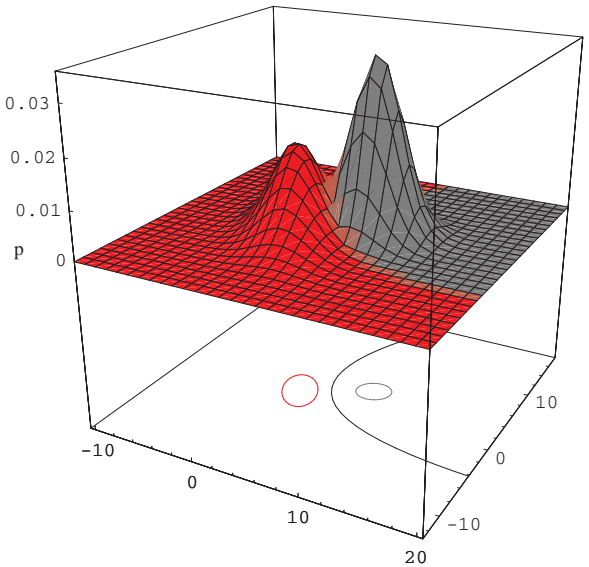
Resulting classifier

- Hyperplane if $\Sigma_\oplus$=$\Sigma_\ominus$ = constant $\cdot$ diag$(1, \ldots, 1)$ (=isotropic Gaussians)
- Quadratic hypersurface otherwise.

A Very Simple Classifier







Discussion

Possible weakness

1. Distributional assumption.
2. Density estimates emphasize main bulk of data. Critical region for classification is at decision boundary, i.e. region between classes.

Consequence

- Classification algorithms focus on class boundary.
- Technically, this means: We focus on estimating a good decision surface (e.g. a hyperplane) between the classes; we do *not* try to estimate a distribution.

Our program in the following

- First develop methods for the linear case, i.e. separate classes by a hyperplane.
- Then: Consider methods that transform linear classifier into non-linear ones.
- Finally: Discuss a family of classification methods that are non-linear by design.

Measuring Performance: Loss Functions

Definition

A **loss function** is a function

$$L : [K] \times [K] \longrightarrow [0, \infty) \ ,$$

which we read as

$$L : (\text{true class label } y, \ \text{classifier output } f(x)) \longmapsto \ \text{loss value} \ \ .$$

Example: The two most common loss functions

1. The **0-1 loss** is used in classification. It counts mistakes:

$$L^{0\text{-}1}(y, f(\mathbf{x})) = \begin{cases} 0 & f(\mathbf{x}) = y \\ 1 & f(\mathbf{x}) \neq y \end{cases}$$

2. **Squared-error loss** is used in regression:

$$L^{\text{se}}(y, f(\mathbf{x})) := \|y - f(\mathbf{x})\|_2^2$$

   Its value depends on how far off we are: Small errors hardly count, large ones are very expensive.

RISK

Motivation

It may be a good strategy to allow (even expensive) errors for values of $\mathbf{x}$ which are very unlikely to occur

Definition

The **risk** $R(f)$ of a classifier f is its expected loss under p, that is,

$$R(f) := \mathbb{E}_p[L(y, f(\mathbf{x})] = \int L(y, f(\mathbf{x}))p(\mathbf{x}, y)d\mathbf{x}dy = \sum_{y=1}^{K} \int L(y, f(\mathbf{x}))p(\mathbf{x}, y)d\mathbf{x} \ .$$

When we train f, we do not know p, and have to approximate R using the data:

The **empirical risk** $\hat{R}_n(f)$ is the plug-in estimate of $R(f)$, evaluated on the training sample $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \ldots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$:

$$\hat{R}_n(f) := \frac{1}{n} \sum_{i=1}^{n} L(\tilde{y}_i, f(\tilde{\mathbf{x}}_i))$$

NAIVE BAYES CLASSIFIERS

Bayes Equation

Simplest form

- Random variables $X \in \mathbf{X}$ and $Y \in \mathbf{Y}$, where $\mathbf{X}, \mathbf{Y}$ are finite sets.
- Each possible value of X and Y has positive probability.

Then

$$P(X = x, Y = y) = P(y|x)P(x) = P(x|y)P(y)$$

and we obtain

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{P(x|y)P(y)}{\sum_{y \in \mathcal{Y}} P(x|y)P(y)}$$

It is customary to name the components,

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}$$

In terms of densities

For continuous sets $\mathbf{X}$ and $\mathbf{Y}$,

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} = \frac{p(x|y)p(y)}{\int_{\mathbf{Y}} p(x|y)dy}$$

Classification

We define a classifier as

$$f(\mathbf{x}) := \arg\max_{y \in [K]} p(y|\mathbf{x})$$

where $\mathbf{Y} = [K]$ and $\mathbf{X}$ = sample space of data variable.

With the Bayes equation, we obtain

$$f(\mathbf{x}) = \arg\max_{y} \frac{P(x|y)P(y)}{P(x)} = \arg\max_{y} P(x|y)P(y)$$

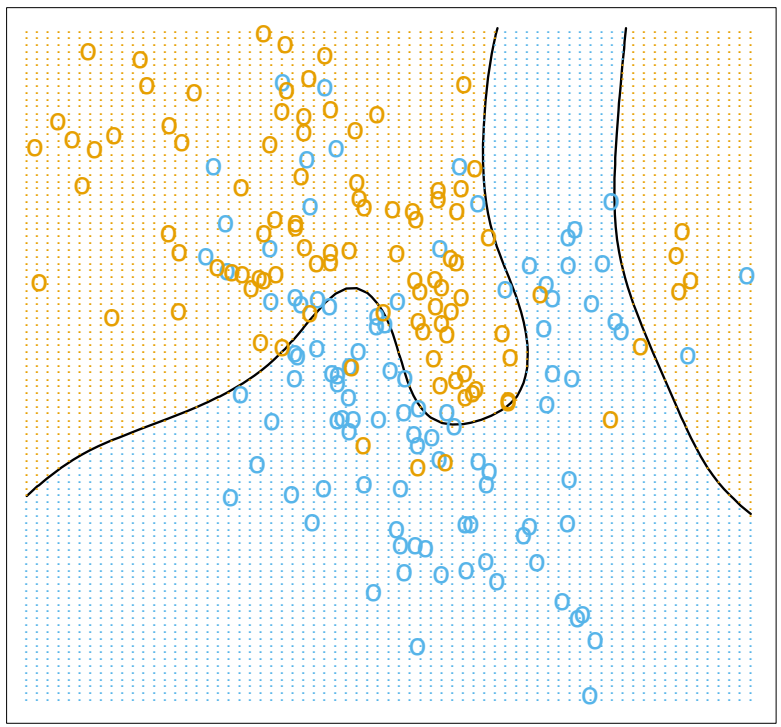
If the class-conditional distribution is continuous, we use

$$f(\mathbf{x}) = \arg\max_{y} p(x|y)P(y)$$

BAYES-OPTIMAL CLASSIFIER

Optimal classifier

- In the risk framework, the best possible classifier is the one which minimizes the expected risk.
- Which classifier is optimal depends on the chosen cost function.



Zero-one loss

Under zero-one loss, the classifier which minimizes the risk is the classifier

$$f(\mathbf{x}) = \arg\max_y P(x|y)P(y)$$

from the previous slide. When computed from the *true* distribution of (X, Y), this classifier is called the **Bayes-optimal classifier** (or **Bayes classifier** for short).

EXAMPLE: SPAM FILTERING

Representing emails

- $\mathbf{Y} = \{$ spam, email $\}$
- $\mathbf{X} = \mathbb{R}^d$
- Each axis is labelled by one possible word.
- d = number of words in vocabulary
- x_j = number of occurences of word j in email represented by $\mathbf{x}$

For example, if axis j represents the term "the", $x_j = 3$ means that "the" occurs three times in the email $\mathbf{x}$. This representation is called a **vector space model of text**.

Example dimensions

	george	you	your	hp	free	hpl	!	our	re	edu	remove
spam	0.00	2.26	1.38	0.02	0.52	0.01	0.51	0.51	0.13	0.01	0.28
email	1.27	1.27	0.44	0.90	0.07	0.43	0.11	0.18	0.42	0.29	0.01

With Bayes equation

$$f(\mathbf{x}) = \operatorname*{argmax}_{y \in \{\text{spam,email}\}} P(y|\mathbf{x}) = \operatorname*{argmax}_{y \in \{\text{spam,email}\}} p(\mathbf{x}|y)P(y)$$

Simplifying assumption

The classifier is called a **naive Bayes** classifier if it assumes

$$p(\mathbf{x}|y) = \prod_{j=1}^{d} p_j(x_i|y) \ ,$$

i.e. if it treats the individual dimensions of $\mathbf{x}$ as conditionally independent given y.

In spam example

- Corresponds to the assumption that the number of occurrences of a word carries information about y.
- Co-occurrences (how often do given combinations of words occur?) is neglected.

Class prior

The distribution $P(y)$ is easy to estimate from training data:

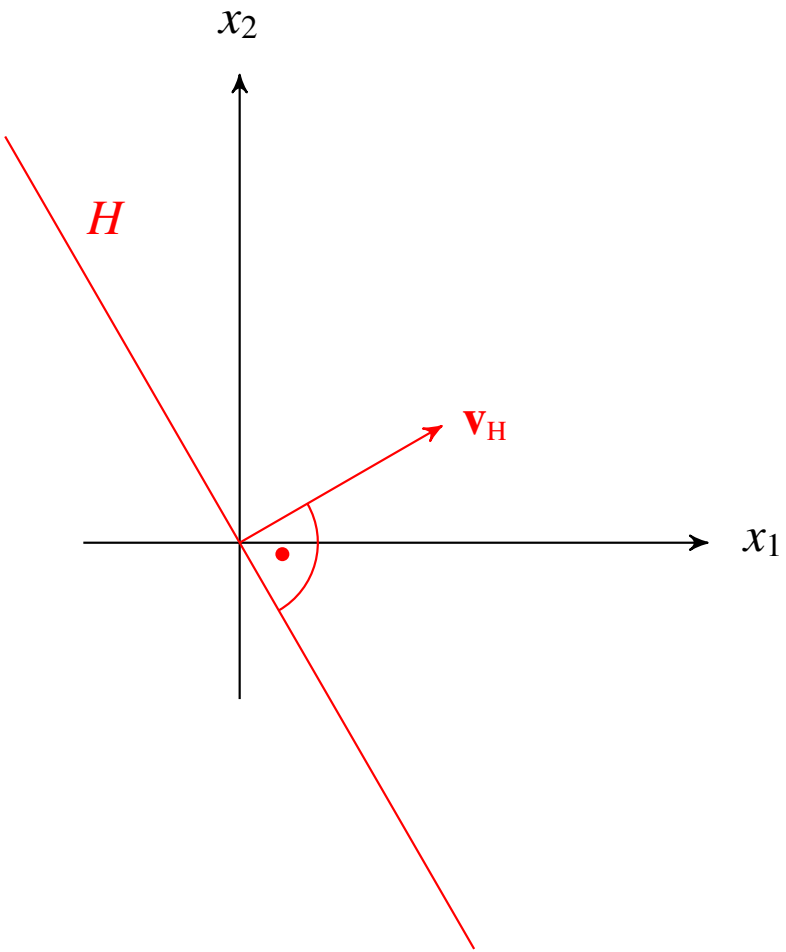
$$P(y) = \frac{\#\text{observations in class } y}{\#\text{observations}}$$

Class-conditional distributions

The class conditionals $p(x|y)$ usually require a modeling assumption. Under a given model:

- Separate the training data into classes.
- Estimate $p(x|y)$ on class y by maximum likelihood.

Linear Classification



Hyperplanes

A **hyperplane** in $\mathbb{R}^d$ is a linear subspace of dimension $(d-1)$.

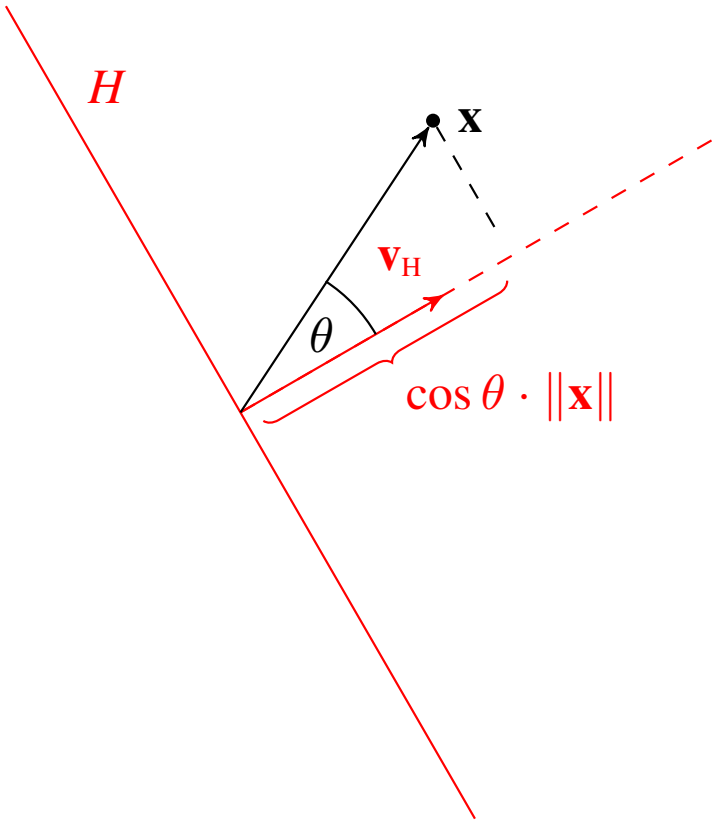
- A $\mathbb{R}^2$-hyperplane is a line, a $\mathbb{R}^3$-hyperplane is a plane.
- As a linear subspace, a hyperplane always contains the origin.

Normal vectors

A hyperplane H can be represented by a **normal vector**. The hyperplane with normal vector $\mathbf{v}_H$ is the set

$$H = \{\mathbf{x} \in \mathbb{R}^d \mid \langle \mathbf{x}, \mathbf{v}_H \rangle = 0\} \ .$$

WHICH SIDE OF THE PLANE ARE WE ON?



Distance from the plane

- The projection of $\mathbf{x}$ onto the direction of $\mathbf{v}_H$ has length $\langle \mathbf{x}, \mathbf{v}_H \rangle$ *measured in units of* $\mathbf{v}_H$, i.e. length $\langle \mathbf{x}, \mathbf{v}_H \rangle / \|\mathbf{v}_H\|$ in the units of the coordinates.
- Recall the cosine rule for the scalar product,

$$\cos\theta = \frac{\langle \mathbf{x}, \mathbf{v}_H \rangle}{\|\mathbf{x}\| \cdot \|\mathbf{v}_H\|} \ .$$

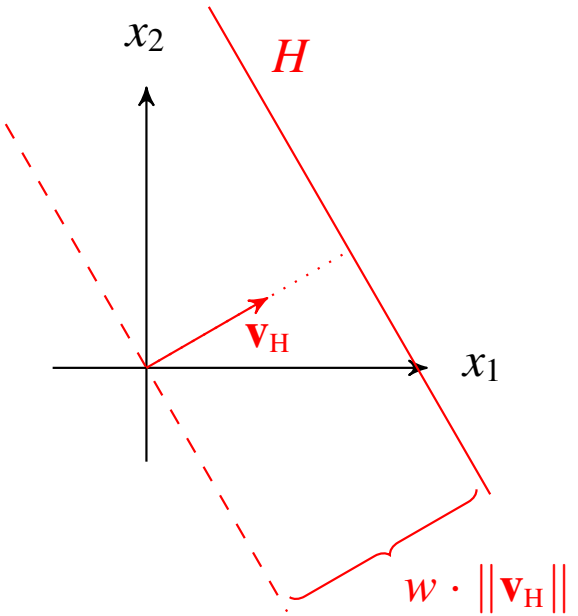
- Consequence: The distance of $\mathbf{x}$ from the plane is given by

$$d(\mathbf{x}, H) = \frac{\langle \mathbf{x}, \mathbf{v}_H \rangle}{\|\mathbf{v}_H\|} = \cos\theta \cdot \|\mathbf{x}\| \ .$$

Which side of the plane?

- The cosine satisfies $\cos\theta > 0$ iff $\theta \in (-\pi, \pi)$.
- We can decide which side of the plane $\mathbf{x}$ is on using

$$\operatorname{sgn}(\cos\theta) = \operatorname{sgn} \langle \mathbf{x}, \mathbf{v}_H \rangle \ .$$



Affine Hyperplanes

- An **affine hyperplane** $H_{\mathbf{w}}$ is a hyperplane translated (shifted) by a vector $\mathbf{w}$, i.e. $H_{\mathbf{w}} = H + w$.
- We choose $\mathbf{w}$ in the direction of $\mathbf{v}_H$, i.e. $\mathbf{w} = c \cdot \mathbf{v}_H$ for $c > 0$.

Which side of the plane?

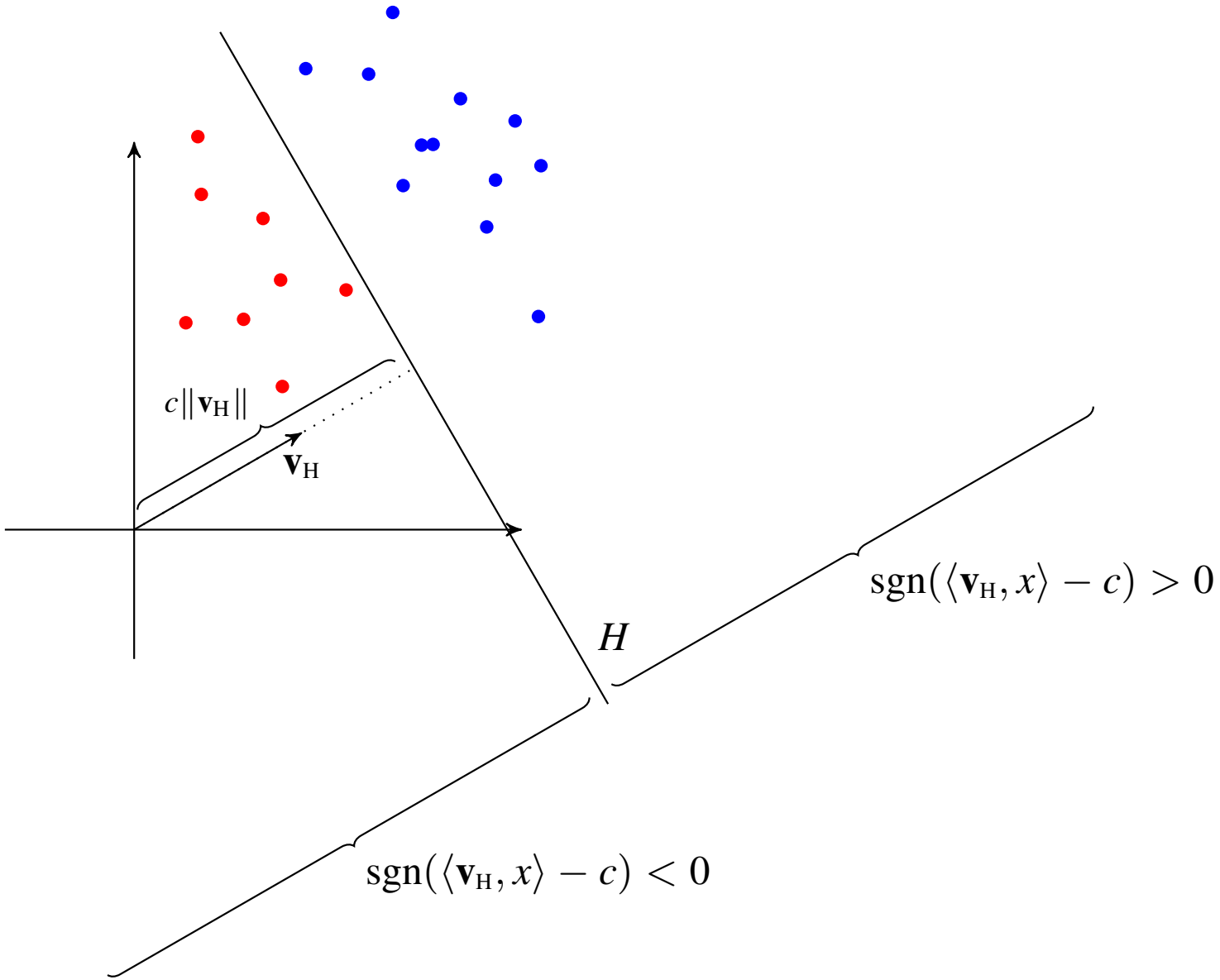
- Which side of $H_{\mathbf{w}}$ a point $\mathbf{x}$ is on is determined by

$$\operatorname{sgn}(\langle \mathbf{x} - \mathbf{w}, \mathbf{v}_H \rangle) = \operatorname{sgn}(\langle \mathbf{x}, \mathbf{v}_H \rangle - c \langle \mathbf{v}_H, \mathbf{v}_H \rangle) = \operatorname{sgn}(\langle \mathbf{x}, \mathbf{v}_H \rangle - c\|\mathbf{v}_H\|^2) \ .$$

- If $\mathbf{v}_H$ is a unit vector, we can use

$$\operatorname{sgn}(\langle \mathbf{x} - \mathbf{w}, \mathbf{v}_H \rangle) = \operatorname{sgn}(\langle \mathbf{x}, \mathbf{v}_H \rangle - c) \ .$$

Classification with Affine Hyperplanes



Definition

A **linear classifier** is a function of the form

$$f_{\mathrm{H}}(\mathbf{x}) := \operatorname{sgn}(\langle \mathbf{x}, \mathbf{v}_{\mathrm{H}} \rangle - c) \ ,$$

where $\mathbf{v}_{\mathrm{H}} \in \mathbb{R}^d$ is a vector and $c \in \mathbb{R}_+$.

Note: We usually assume $\mathbf{v}_{\mathrm{H}}$ to be a unit vector. If it is not, f_{H} still defines a linear classifier, but c describes a shift of a different length.

Definition

Two sets $A, B \in \mathbb{R}^d$ are called **linearly separable** if there is an affine hyperplane H which separates them, i.e. which satisfies

$$\langle \mathbf{x}, \mathbf{v}_{\mathrm{H}} \rangle - c = \begin{cases} < 0 & \text{if } \mathbf{x} \in A \\ > 0 & \text{if } \mathbf{x} \in B \end{cases}$$

THE PERCEPTRON ALGORITHM

RISK MINIMIZATION

Definition

Let $\mathcal{H}$ be the set of all classifiers considered in a given classification problem. The set $\mathcal{H}$ is called a **hypothesis space**.

For linear classifiers, $\mathcal{H} = \{$ all hyperplanes in $\mathbb{R}^d\}$.

Selecting a classifier

Select $f \in \mathcal{H}$ which minimizes risk. With zero-one loss:

$$f \in \operatorname*{argmin}_{f \in \mathcal{H}} R(f) = \operatorname*{argmin}_{f \in \mathcal{H}} \mathbb{E}_p[L(y, f(\mathbf{x}))]$$

We cannot evaluatate this expression, since we do not know p.

Approximation with data: Empirical risk minimization

We approximate the risk criterion by the empirical risk

$$f \in \arg\min f \in \mathcal{H} \hat{R}_n(f) = \operatorname*{argmin}_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i))$$

If we choose $L = L^{0\text{-}1}$, this minimizes the number of errors on the training data.

Parameterizing the hypothesis space

- Linear classification: Every $f \in \mathcal{H}$ is of the form $f(\mathbf{x}) = \operatorname{sgn}(\langle \mathbf{x}, \mathbf{v}_{\mathrm{H}} \rangle - c)$.
- f can be specified by specifying $\mathbf{v}_{\mathrm{H}} \in \mathbb{R}^d$ and $c \in \mathbb{R}$.
- We collect $\mathbf{v}_{\mathrm{H}}$ and c in a single vector $\mathbf{z} := (-c, \mathbf{v}_{\mathrm{H}}) \in \mathbb{R}^{d+1}$.

We now have

$$\langle \mathbf{x}, \mathbf{v}_{\mathrm{H}} \rangle - c = \left\langle \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix}, \mathbf{z} \right\rangle \qquad \text{and} \qquad f(\mathbf{x}) = \operatorname{sgn}\left\langle \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix}, \mathbf{z} \right\rangle$$

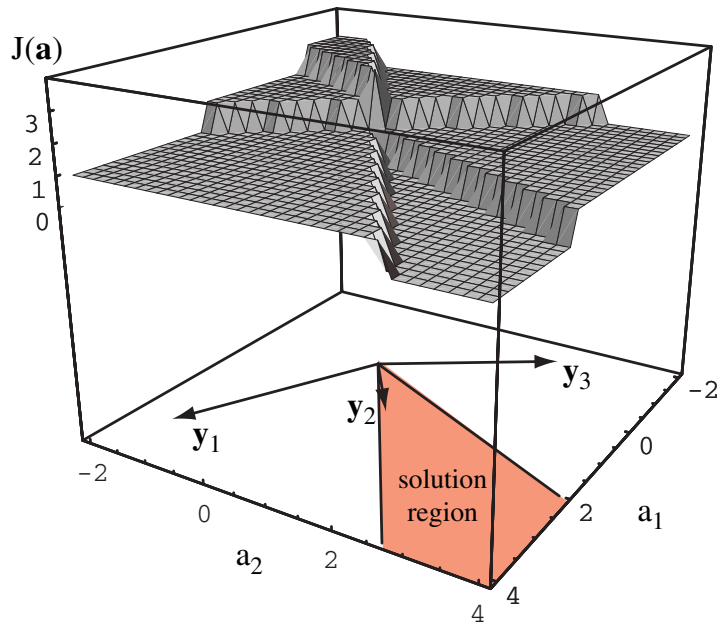
The *affine* plane in $\mathbb{R}^d$ can now be interpreted as a *linear* plane in $\mathbb{R}^{d+1}$. The $d+1$-dimensional coordinates in the representation are called **homogeneous coordinates**.

Numerical minimization of the empirical risk

Naive strategy:

1. Substitute the parametrization of f into $\hat{R}_n(f)$ (evaluated on the training data).
2. Minimize with respect to $\mathbf{z}$ by numerical optimization.

Problem: $\hat{R}_n(f)$ is piece-wise constant.



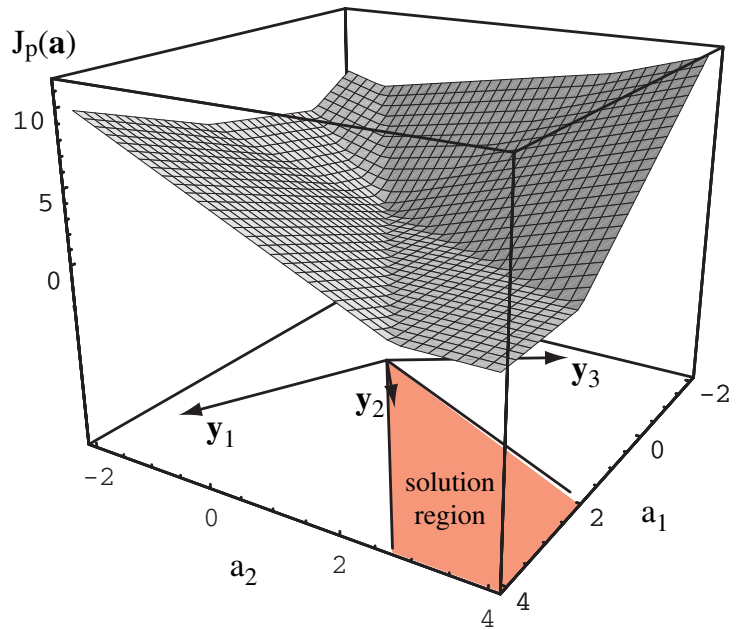
Solution region

The solution region is set of vectors $\mathbf{z}$ which achieve zero training error.

- If the training data is linearly separable, the solution region is a cone in $\mathbb{R}^{d+1}$.
- Otherwise, the solution region is empty.

Perceptron cost function

- Error rate not suited for numerical optimization.
- Strategy: Approximate $\hat{R}_n(f)$ by a piece-wise linear function.



The approximation

$$C_{\mathrm{P}}(f) := \sum_{i=1}^{n} \mathbb{I}\{f(\tilde{\mathbf{x}}_i) \neq \tilde{y}_i\} \left| \left\langle \mathbf{z}, \begin{pmatrix} 1 \\ \tilde{\mathbf{x}}_i \end{pmatrix} \right\rangle \right|$$

is called the **Perceptron cost function**.

Cost functions

The more general theme is that we substitute $\hat{R}_n$ by a **cost function** $C : \mathcal{H} \longrightarrow \mathbb{R}_+$.
A cost function defines a training strategy as

$$\text{training method} \;=\; \text{cost function} \;+\; \text{minimization algorithm}$$

The Perceptron

A linear classifier obtained by minimizing the Perceptron cost function is called a **Perceptron**.

Algorithm

Repeat until $C_{\mathrm{P}}(\mathbf{z}^k) = 0$:

$$\mathbf{z}^{k+1} := \mathbf{z}^k - \alpha(k)\nabla C_{\mathrm{P}}(\mathbf{z}^k)$$

where k enumerates iterations.

Step size

The step size parameter α is called the **learning rate**. Common choices are

$$\alpha(k) = 1 \qquad \text{or} \qquad \alpha(k) = \frac{1}{k} \ .$$

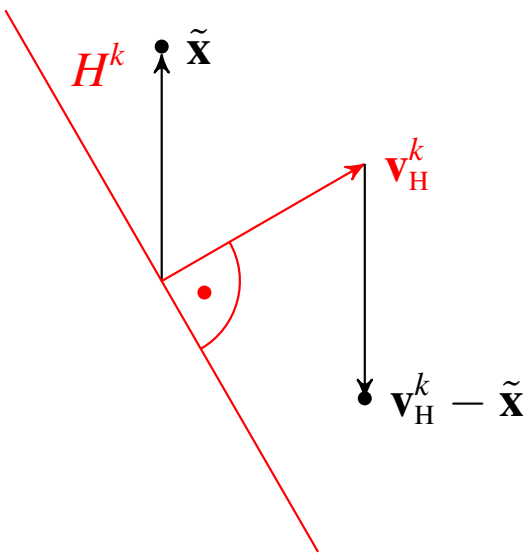
Gradient of the cost function

$$\nabla_{\mathbf{z}} C_{\mathrm{P}}(\mathbf{z}) = \sum_{i=1}^{n} \mathbb{I}\{f_{\mathrm{H}}(\tilde{\mathbf{x}}_i) \neq \tilde{y}_i\} \, \nabla_{\mathbf{z}} \left| \left\langle \mathbf{z}, \begin{pmatrix} 1 \\ \tilde{\mathbf{x}}_i \end{pmatrix} \right\rangle \right| = \sum_{i=1}^{n} \mathbb{I}\{f(\tilde{\mathbf{x}}_i) \neq \tilde{y}_i\} \cdot \operatorname{sgn}\left(\left\langle \mathbf{z}, \begin{pmatrix} 1 \\ \tilde{\mathbf{x}}_i \end{pmatrix} \right\rangle \right) \cdot \begin{pmatrix} 1 \\ \tilde{\mathbf{x}}_i \end{pmatrix}$$

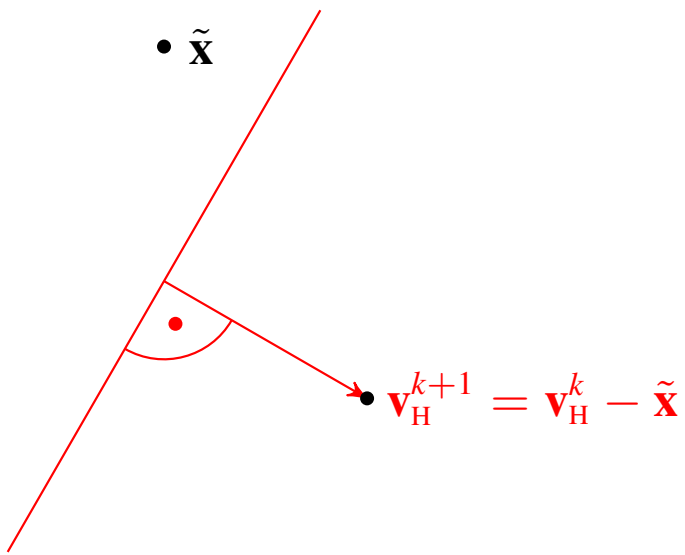
$$= \sum_{i=1}^{n} \mathbb{I}\{f(\tilde{\mathbf{x}}_i) \neq \tilde{y}_i\} \cdot (-\tilde{y}_i) \begin{pmatrix} 1 \\ \tilde{\mathbf{x}}_i \end{pmatrix} \; .$$

Effect for a single training point

Step k: $\tilde{\mathbf{x}}$ (in class -1) classified incorrectly



Step $k+1$



Simplifying assumption: H contains origin

Does the Perceptron Work?

The algorithm we discussed before is called the **batch Perceptron**. For learning rate $\alpha = 1$, we can equivalently add data points one at a time.

Alternative Algorithm

Repeat until $C_{\mathrm{P}}(\mathbf{z}) = 0$:

1. For all $i = 1, \ldots, n$: $\quad \mathbf{z}^k := \mathbf{z}^k + \tilde{y}_i \begin{pmatrix} 1 \\ \tilde{\mathbf{x}}_i \end{pmatrix}$
2. $k := k + 1$

This is called the **fixed-increment single-sample Perceptron**, and is somewhat easier to analyze than the batch Perceptron.

Theorem: Perceptron convergence

If (and only if) the training data is linearly separable, the fixed-increment single-sample Perceptron terminates after a finite number of steps with a valid solution vector $\mathbf{z}$ (i.e. a vector which classifies all training data points correctly).