Applied Data Mining

May 29, 2025

INTRODUCTION

COURSE WEBPAGE

http://stat.columbia.edu/~porbanz/UN3106S18.html

This class

What to expect

- This class is an introduction to machine learning.
- Topics: Classification; “learning”; basic neural networks; etc

Homework

- Programming + “theoretical” questions.
- All programming will be done in R.

What this class is not

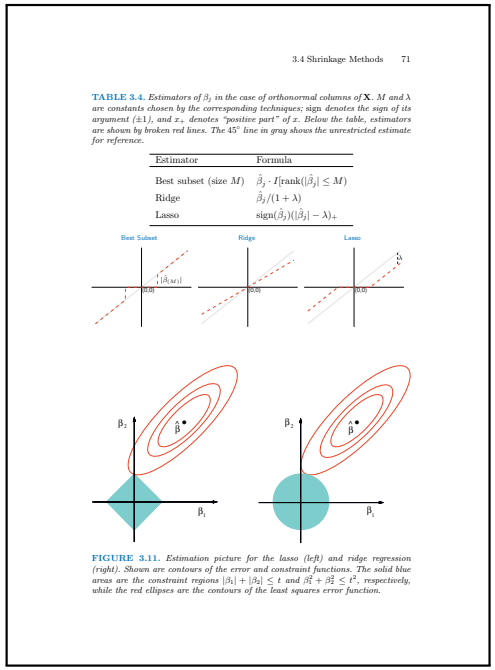
- Applied.

The purpose of this class is to understand how some of the most important machine learning methods work.

Books

T. Hastie, J. Friedman and R. Tibshirani:
"The Elements of Statistical Learning".
2nd Edition, Springer, 2009.

Available online.

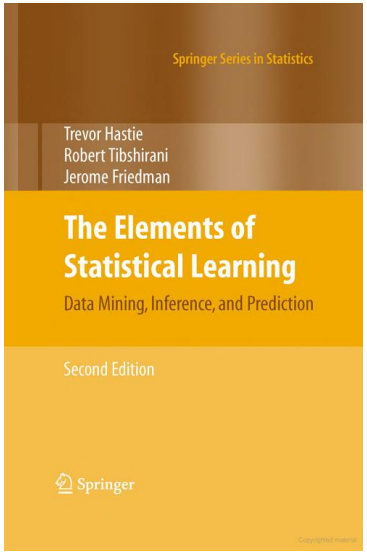


3.4 Shrinkage Methods 71

TABLE 3.4. *Estimators of $\hat{\beta}_j$ in the case of orthonormal columns of* $\mathbf{X}$. *M and λ are constants chosen by the corresponding techniques; sign denotes the sign of its argument (± 1), and x_+ denotes "positive part" of x. Below the table, estimators are shown by broken red lines. The 45° line in gray shows the unrestricted estimate for reference.*

Estimator	Formula
Best subset (size M)	$\hat{\beta}_j \cdot I[\text{rank}(\lvert\hat{\beta}_j\rvert \le M)$
Ridge	$\hat{\beta}_j/(1+\lambda)$
Lasso	$\text{sign}(\hat{\beta}_j)(\lvert\hat{\beta}_j\rvert - \lambda)_+$

FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $\lvert\beta_1\rvert + \lvert\beta_2\rvert \le t$ and $\beta_1^2 + \beta_2^2 \le t^2$, respectively, while the red ellipses are the contours of the least squares error function.*



⟵ It's much prettier inside.

Links to this book and other potentially useful references will be added to the class homepage as they become relevant. All of these are optional; the relevant material are the course slides.

Background Knowledge

- Euclidean space; vectors
- Scalar products
- Derivatives and gradients of functions
- Probability distributions and densities. Example:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} \qquad \text{or} \qquad p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

- Gaussian distribution on $\mathbb{R}$ and $\mathbb{R}^d$
- (Eigenvalues and eigenvectors)
- Regression

Problem setting

Classification methods subdivide data into several, distinct classes. More formally:

- Data $x_1, x_2, \ldots$.
- Each observations falls into one of K categories (the *classes*).
- Learning task: Find a classification function

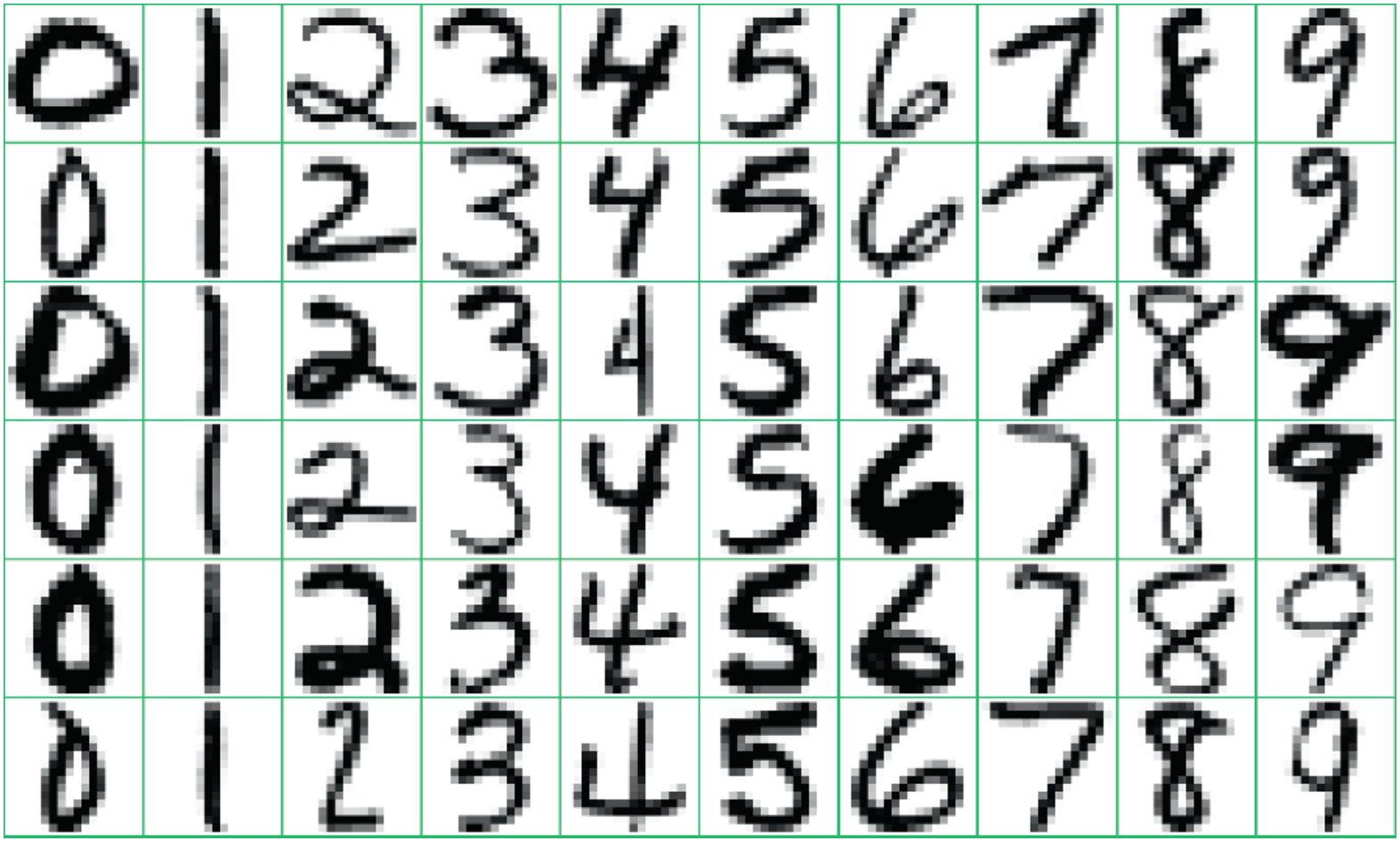
$$f : \mathbf{X} \to \{1, \ldots, K\} \ .$$

- Input of the learning problem: Correctly categorized examples $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n$.

Approach

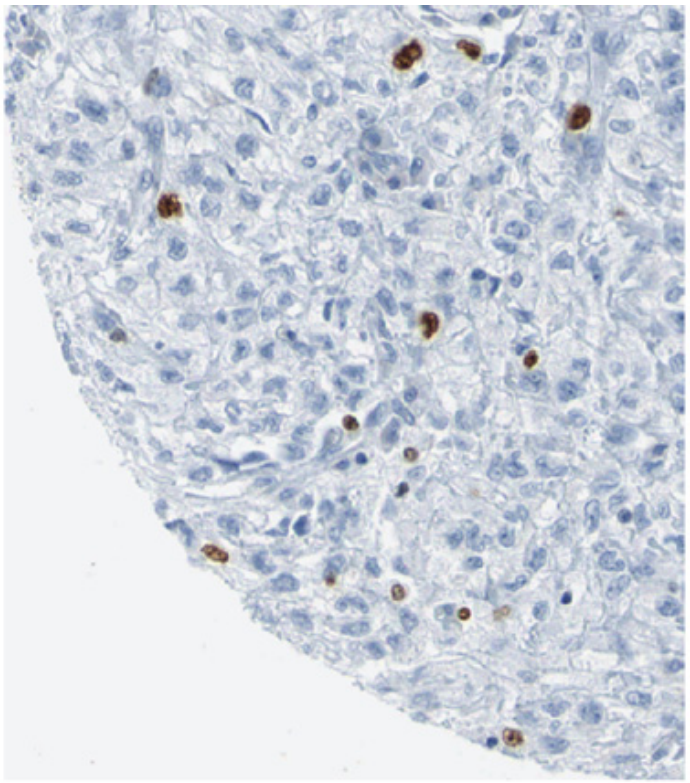
- Define:
  1. A set of possible classification functions f (the *hypothesis set*).
  2. A *cost function* which assumes a large value when mistakes are made.
- To find a good classifier, search the hypothesis class for the f which keeps costs as small as possible.
- Different types of errors can be more or less expensive.

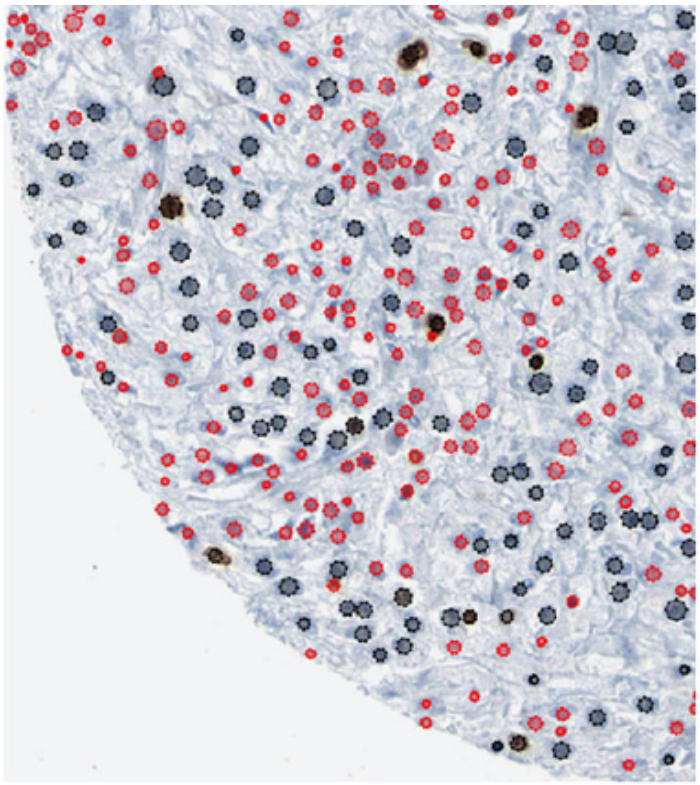
USPS Data



Each digit: 16×16 pixels, i.e. $x \in \mathbb{R}^{256}$

Cancer diagnosis





Previous examples as classification problems

- USPS data: 10 classes (+ one "outlier class")
- Cancer diagnosis: 4 classes

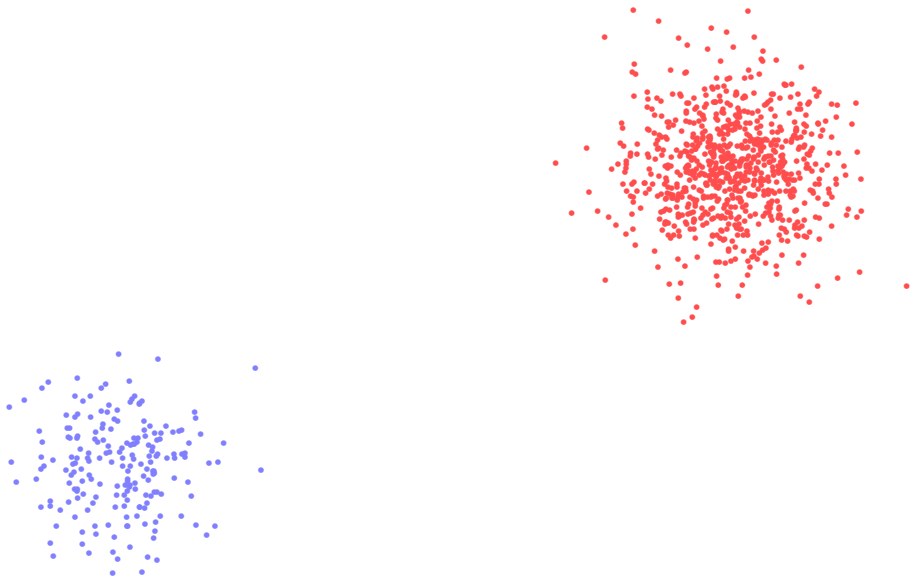
Face recognition

- Hard problem, but much recent progress.
- Deployable systems can now have around 90+% accuracy on people in their database.
- 1 class per person in data base + 1 class for "none of those".

Fingerprint recognition

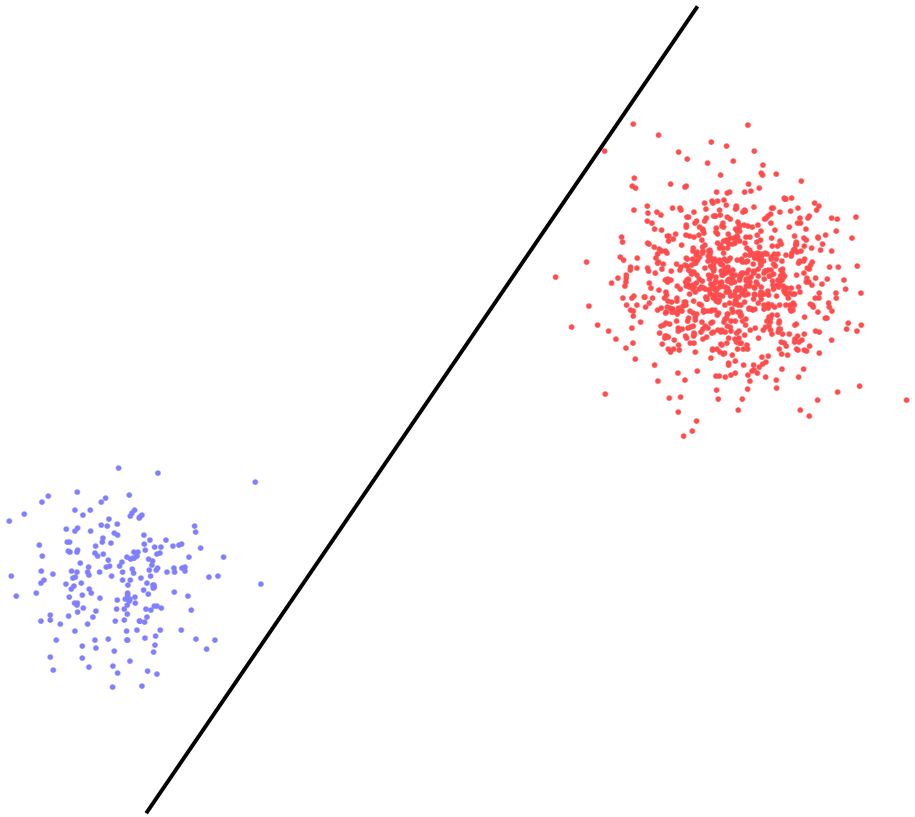
- Again: 1 class per person in data base + 1 class for "none of those".
- Deployable systems have been available for ca. 15 years.
- Development of computer systems lead to reassessment of human error rates.

TWO-CLASS CLASSIFICATION: BASIC IDEAS



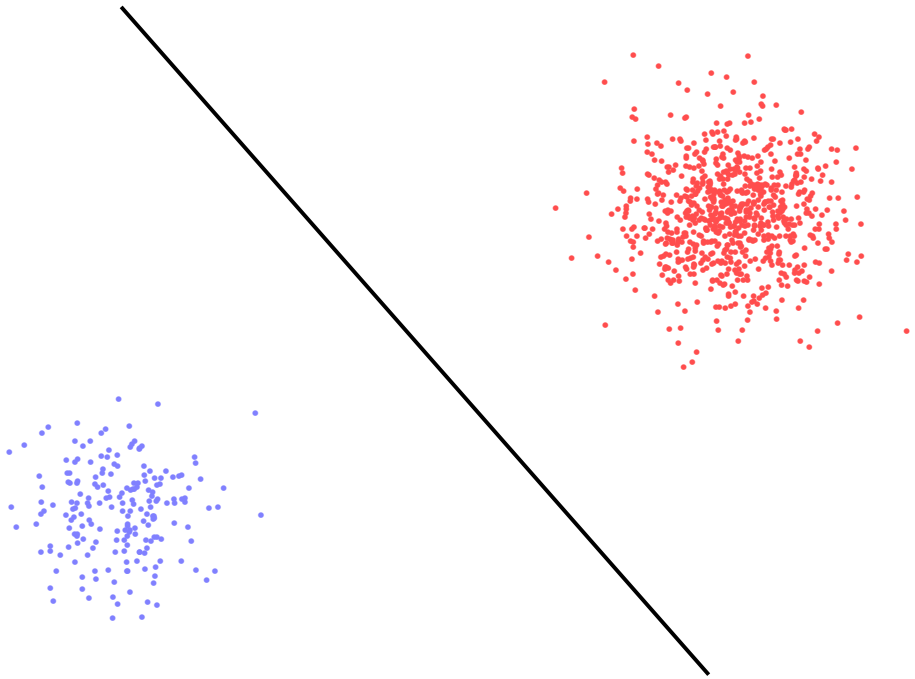
1. Represent each object as a point; axes = measurements ($\rightarrow$ *vector spaces*)
2. Separate classes by hyperplane ($\rightarrow$ *scalar products*)
3. Define a function that measures how well the plane separates classes; small values indicate a good fit.
4. Find "good" hyperplane by minimizing function ($\rightarrow$ *derivatives, gradients, Hessians, etc*)

TWO-CLASS CLASSIFICATION: BASIC IDEAS



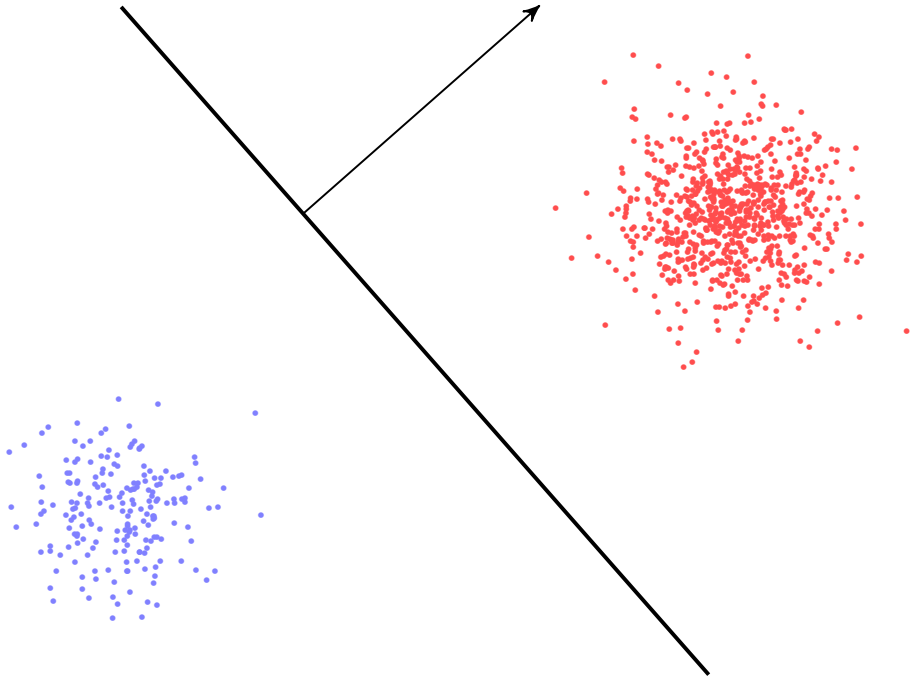
1. Represent each object as a point; axes = measurements ($\rightarrow$ *vector spaces*)
2. Separate classes by hyperplane ($\rightarrow$ *scalar products*)
3. Define a function that measures how well the plane separates classes; small values indicate a good fit.
4. Find "good" hyperplane by minimizing function ($\rightarrow$ *derivatives, gradients, Hessians, etc*)

TWO-CLASS CLASSIFICATION: BASIC IDEAS



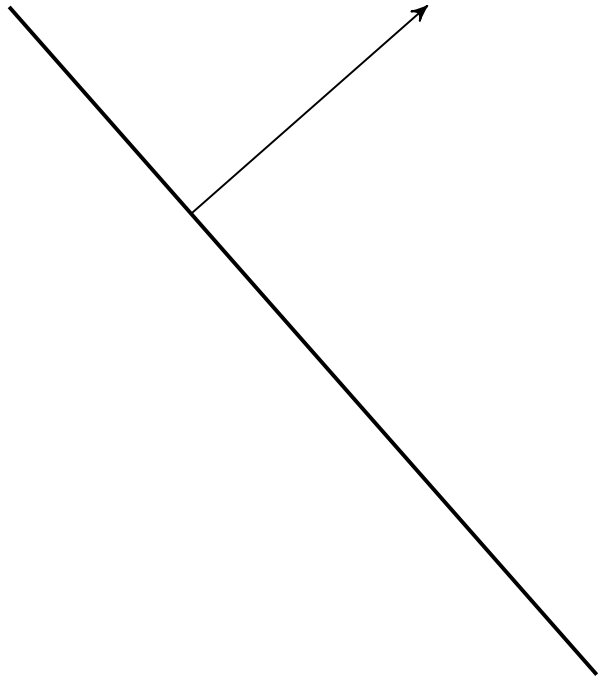
1. Represent each object as a point; axes = measurements ($\rightarrow$ *vector spaces*)
2. Separate classes by hyperplane ($\rightarrow$ *scalar products*)
3. Define a function that measures how well the plane separates classes; small values indicate a good fit.
4. Find "good" hyperplane by minimizing function ($\rightarrow$ *derivatives, gradients, Hessians, etc*)

TWO-CLASS CLASSIFICATION: BASIC IDEAS



1. Represent each object as a point; axes = measurements ($\rightarrow$ *vector spaces*)
2. Separate classes by hyperplane ($\rightarrow$ *scalar products*)
3. Define a function that measures how well the plane separates classes; small values indicate a good fit.
4. Find "good" hyperplane by minimizing function ($\rightarrow$ *derivatives, gradients, Hessians, etc*)

TWO-CLASS CLASSIFICATION: BASIC IDEAS



1. Represent each object as a point; axes = measurements ($\rightarrow$ *vector spaces*)
2. Separate classes by hyperplane ($\rightarrow$ *scalar products*)
3. Define a function that measures how well the plane separates classes; small values indicate a good fit.
4. Find "good" hyperplane by minimizing function ($\rightarrow$ *derivatives, gradients, Hessians, etc*)

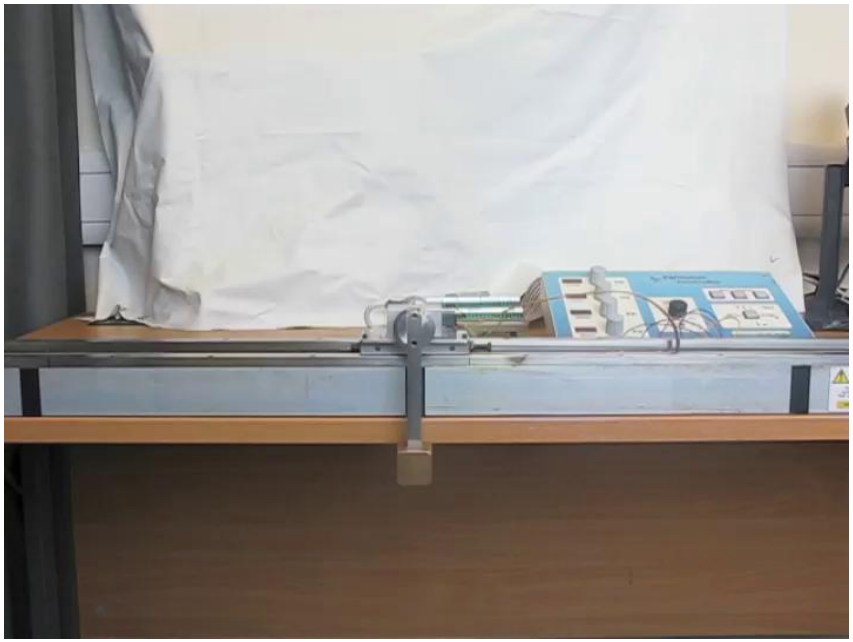
PENDULUM

(WORK OF MARC DEISENROTH AND CARL EDWARD RASMUSSEN)

Task

Balance the pendulumn upright by moving the sled left and right.

- The computer can control *only* the motion of the sled.
- Available data: Current state of system (measured 25 times/second).



Formalization

State = 4 variables (sled location, sled velocity, angle, angular velocity)
Actions = sled movements

The system can be described by a function

$$\begin{aligned} f: \quad & \mathcal{S} \times \mathcal{A} \quad \to \quad \mathcal{S} \\ & (\text{state}, \text{action}) \mapsto \text{state} \end{aligned}$$

Historical origins: Artificial intelligence and engineering

Machines need to...

- recognize patterns (e.g. vision, language)
- make decisions based on experience (= data)
- predict
- cope with uncertainty

Today

- There is no clear dividing line between machine learning and statistics anymore.
- Engineering aspects (such as software development and specialized hardware) have become much more important as machine learning systems get deployed.

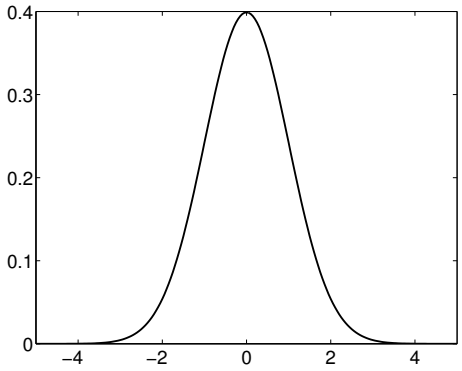
Modern applications: (A few) Examples

- medical diagnosis
- face detection/recognition
- speech and handwriting recognition
- web search
- recommender systems
- bioinformatics
- natural language processing
- computer vision

REVIEW: GAUSSIAN DISTRIBUTIONS

Gaussian density in one dimension

$$p(x;\mu,\sigma) := \frac{1}{\sqrt{2\pi}\sigma}\exp\Big(-\frac{(x-\mu)^2}{2\sigma^2}\Big)$$



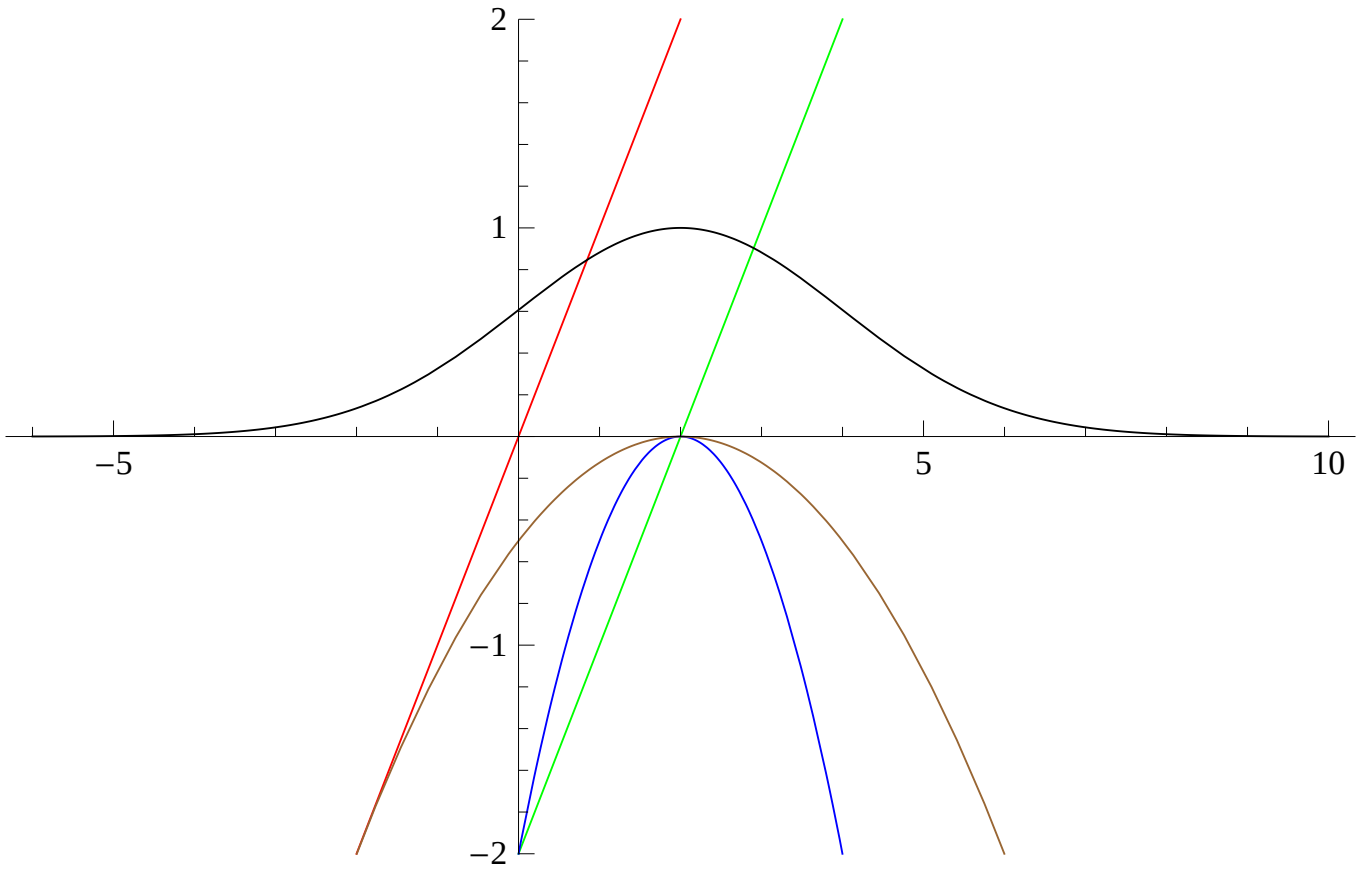
- μ = expected value of x, σ^2 = variance, σ = standard deviation
- The quotient $\frac{x-\mu}{\sigma}$ measures deviation of x from its expected value units of σ (i.e. σ defines the length scale)

Recall: Standard deviation around the mean

- Recall that the interval $[\mu-\sigma, \mu+\sigma]$ ("one standard deviation") always contains the same amount of probability mass (ca. 68.27%), regardless of the choice of μ and σ.
- Similarly, the intervall $[\mu-2\sigma, \mu+2\sigma]$ contains $\sim 95.45\%$ of the mass, and $[\mu-3\sigma, \mu+3\sigma]$ contains $\sim 99.73\%$.

Components of a 1D Gaussian

$\mu = 2, \sigma = 2$



- Red: $x \mapsto x$
- Green: $x \mapsto x - \mu$
- Blue: $x \mapsto -\frac{1}{2}(x - \mu)^2$
- Brown: $x \mapsto -\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2$
- Black: $x \mapsto \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$

Recall: Covariance

The covariance of two random variables X_1, X_2 is

$$\mathrm{Cov}[X_1, X_2] = \mathbb{E}[(X_1 - \mathbb{E}[X_1])(X_2 - \mathbb{E}[X_2])] \ .$$

If $X_1 = X_2$, the covariance is the variance: $\mathrm{Cov}[X, X] = \mathrm{Var}[X]$.

Covariance matrix

If $X = (X_1, \ldots, X_m)$ is a random vector with values in $\mathbb{R}^m$, the matrix of all covariances

$$\mathrm{Cov}[X] := (\mathrm{Cov}[X_i, X_j])_{i,j} = \begin{pmatrix} \mathrm{Cov}[X_1, X_1] & \cdots & \mathrm{Cov}[X_1, X_m] \\ \vdots & & \vdots \\ \mathrm{Cov}[X_m, X_1] & \cdots & \mathrm{Cov}[X_m, X_m] \end{pmatrix}$$

is called the **covariance matrix** of X.

Notation

It is customary to denote the covariance matrix $\mathrm{Cov}[X]$ by Σ.

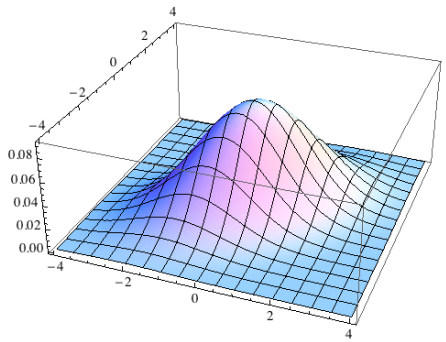
Gaussian density in m dimensions

The quadratric function

$$-\frac{(x-\mu)^2}{2\sigma^2} \quad = \quad -\frac{1}{2}(x-\mu)(\sigma^2)^{-1}(x-\mu)$$

is replaced by a quadratic form:

$$p(\mathbf{x};\boldsymbol{\mu},\Sigma) := \frac{1}{\sqrt{2\pi\det(\Sigma)}}\exp\Big(-\frac{1}{2}\left\langle(\mathbf{x}-\boldsymbol{\mu}),\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\rangle\Big)$$

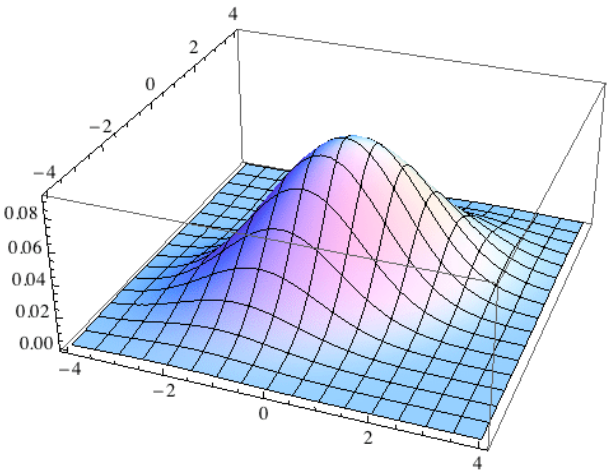


Covariance matrix of a Gaussian

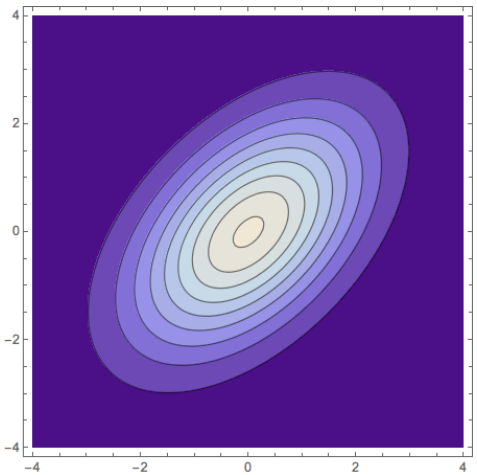
If a random vector $X \in \mathbb{R}^m$ has Gaussian distribution with density $p(\mathbf{x};\mu,\Sigma)$, its covariance matrix is $\mathrm{Cov}[X] = \Sigma$. In other words, a Gaussian is parameterized by its covariance.

GAUSSIAN DENSITY: EXAMPLE

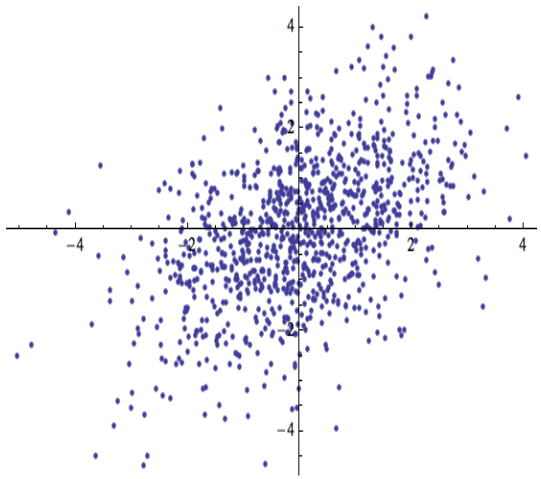
$$p(\mathbf{x}; \boldsymbol{\mu}, \Sigma) \quad \text{with} \quad \boldsymbol{\mu} = (0, 0) \quad \text{with} \quad \Sigma = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$



Density

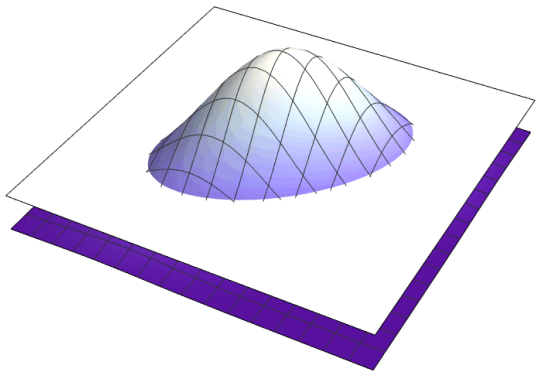


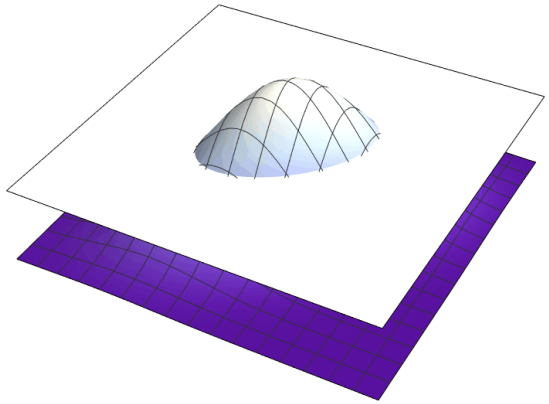
Contour lines



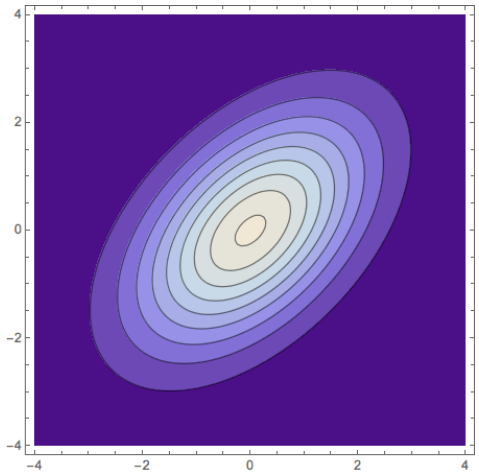
1000 sample points

CONTOUR LINES





Intersect density with a horizontal plane, draw intersection as a curve, and project it down onto the plane.



Each elliptical line is such a contour, for planes at different heights.

Contours and standard deviation

- Each ellipse consists of all points $\mathbf{x} \in \mathbb{R}^2$ that satisfy the equation

$$\left\langle \mathbf{x}, \Sigma^{-1}\mathbf{x} \right\rangle = c \qquad \text{for some fixed } c > 0 \ .$$

  Changing c changes the size of the ellipse.

- The ellipses play the same role as intervals around the mean for 1D Gaussians: The ellipse with $\left\langle \mathbf{x}, \Sigma^{-1}\mathbf{x} \right\rangle = 1$ contains $\sim 68.27\%$ of the probability mass, etc.

- That is: The area within the ellipse given by $\left\langle \mathbf{x}, \Sigma^{-1}\mathbf{x} \right\rangle = k$ corresponds to k standard deviations.

TOOLS: MAXIMUM LIKELIHOOD

Models

A **model** $\mathcal{P}$ is a set of probability distributions. We index each distribution by a parameter value $\theta \in \mathcal{T}$; we can then write the model as

$$\mathcal{P} = \{P_\theta | \theta \in \mathcal{T}\} \ .$$

The set $\mathcal{T}$ is called the **parameter space** of the model.

Parametric model

The model is called **parametric** if the number of parameters (i.e. the dimension of the vector θ) is (1) finite and (2) independent of the number of data points. Intuitively, the complexity of a parametric model does not increase with sample size.

Density representation

For parametric models, we can assume that $\mathcal{T} \subset \mathbb{R}^d$ for some fixed dimension d. We usually represent each P_θ be a density function $p(x|\theta)$.

Setting

- Given: Data $x_1, \ldots, x_n$, parametric model $\mathcal{P} = \{p(x|\theta) \mid \theta \in \mathcal{T}\}$.
- Objective: Find the distribution in $\mathcal{P}$ which best explains the data. That means we have to choose a "best" parameter value $\hat{\theta}$.

Maximum Likelihood approach

Maximum Likelihood assumes that the data is best explained by the distribution in $\mathcal{P}$ under which it has the highest probability (or highest density value).

Hence, the **maximum likelihood estimator** is defined as

$$\hat{\theta}_{\text{ML}} := \arg\max_{\theta \in \mathcal{T}} p(x_1, \ldots, x_n|\theta)$$

the parameter which maximizes the joint density of the data.

The i.i.d. assumption

The standard assumption of ML methods is that the data is **independent and identically distributed (i.i.d.)**, that is, generated by independently sampling repeatedly from the same distrubtion P.

If the density of P is $p(x|\theta)$, that means the joint density decomposes as

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i|\theta)$$

Maximum Likelihood equation

The analytic criterion for a maximum likelihood estimator (under the i.i.d. assumption) is:

$$\nabla_\theta \left(\prod_{i=1}^{n} p(x_i|\theta) \right) = 0$$

We use the "logarithm trick" to avoid a huge product rule computation.

Not examinable.

Recall: Logarithms turn products into sums

$$\log\left(\prod_i f_i\right) = \sum_i \log(f_i)$$

Logarithms and maxima

The logarithm is monotonically increasing on $\mathbb{R}_+$.

Consequence: Application of log does not change the *location* of a maximum or minimum:

$$\max_y \log(g(y)) \neq \max_y g(y) \qquad \text{The } \textit{value} \text{ changes.}$$

$$\arg\max_y \log(g(y)) = \arg\max_y g(y) \qquad \text{The } \textit{location} \text{ does not change.}$$

Not examinable.

Likelihood and logarithm trick

$$\hat{\theta}_{\text{ML}} = \arg\max_{\theta} \prod_{i=1}^{n} p(x_i|\theta) = \arg\max_{\theta} \log\Big(\prod_{i=1}^{n} p(x_i|\theta)\Big) = \arg\max_{\theta} \sum_{i=1}^{n} \log p(x_i|\theta)$$

Analytic maximality criterion

$$0 = \sum_{i=1}^{n} \nabla_\theta \log p(x_i|\theta) = \sum_{i=1}^{n} \frac{\nabla_\theta p(x_i|\theta)}{p(x_i|\theta)}$$

Whether or not we can solve this analytically depends on the choice of the model!

Not examinable.

EXAMPLE: GAUSSIAN MEAN MLE

Model: Multivariate Gaussians

The model $\mathcal{P}$ is the set of all Gaussian densities on $\mathbb{R}^d$ with *fixed* covariance matrix Σ,

$$\mathcal{P} = \{g(\,.\,|\mu, \Sigma) \,|\, \mu \in \mathbb{R}^d\}\ ,$$

where g is the Gaussian density function. The parameter space is $\mathcal{T} = \mathbb{R}^d$.

MLE equation

We have to solve the maximum equation

$$\sum_{i=1}^{n} \nabla_\mu \log g(x_i|\mu, \Sigma) = 0$$

for μ.

Not examinable.

EXAMPLE: GAUSSIAN MEAN MLE

$$
\begin{aligned}
0 &= \sum_{i=1}^{n} \nabla_\mu \log \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\Big(-\frac{1}{2}\Big\langle (x_i-\mu), \Sigma^{-1}(x_i-\mu)\Big\rangle\Big) \\
&= \sum_{i=1}^{n} \nabla_\mu \Big(\log\Big(\frac{1}{\sqrt{(2\pi)^d |\Sigma|}}\Big) + \log\Big(\exp\Big(-\frac{1}{2}\Big\langle (x_i-\mu), \Sigma^{-1}(x_i-\mu)\Big\rangle\Big)\Big) \\
&= \sum_{i=1}^{n} \nabla_\mu \Big(-\frac{1}{2}\Big\langle (x_i-\mu), \Sigma^{-1}(x_i-\mu)\Big\rangle\Big) = -\sum_{i=1}^{n} \Sigma^{-1}(x_i-\mu)
\end{aligned}
$$

Multiplication by $(-\Sigma)$ gives

$$
0 = \sum_{i=1}^{n} (x_i - \mu) \qquad \Rightarrow \qquad \mu = \frac{1}{n}\sum_{i=1}^{n} x_i
$$

Conclusion

The maximum likelihood estimator of the Gaussian expectation parameter for fixed covariance is

$$
\hat{\mu}_{\mathrm{ML}} := \frac{1}{n}\sum_{i=1}^{n} x_i
$$

Not examinable.

Model: Multivariate Gaussians

The model $\mathcal{P}$ is now

$$\mathcal{P} = \{g(\,.\,|\mu, \Sigma) \,|\, \mu \in \mathbb{R}^d, \Sigma \in \Delta_d\}\ ,$$

where Δ_d is the set of all possible $d \times d$ covariance matrices. The parameter space is $\mathcal{T} = \mathbb{R}^d \times \Delta_d$.

ML approach

Since we have just seen that the ML estimator of μ does not depend on Σ, we can compute $\hat{\mu}_{\text{ML}}$ first. We then estimate Σ using the criterion

$$\sum_{i=1}^{n} \nabla_\Sigma \log g(x_i|\hat{\mu}_{\text{ML}}, \Sigma) = 0$$

Solution

The ML estimator of Σ is

$$\hat{\Sigma}_{\text{ML}} := \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu}_{\text{ML}})(x_i - \hat{\mu}_{\text{ML}})^t\ .$$

Not examinable.

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

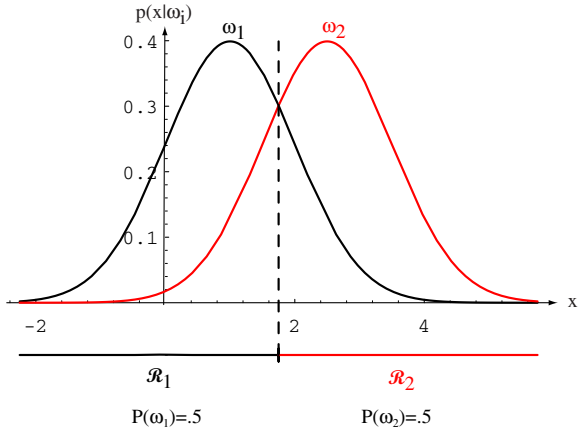
1. On training data, fit a Gaussian into each class (by MLE).
   Result: Densities $g(\mathbf{x}|\mu_\oplus, \Sigma_\oplus)$ and $g(\mathbf{x}|\mu_\ominus, \Sigma_\ominus)$
2. Classify a new point $\mathbf{x}$ according to which density assigns larger value:

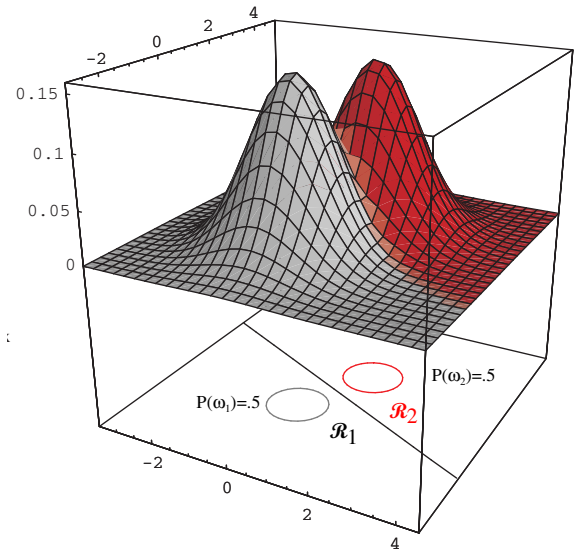
$$y_i := \begin{cases} +1 & \text{if } g(\mathbf{x}|\mu_\oplus, \Sigma_\oplus) > g(\mathbf{x}|\mu_\ominus, \Sigma_\ominus) \\ -1 & \text{otherwise} \end{cases}$$

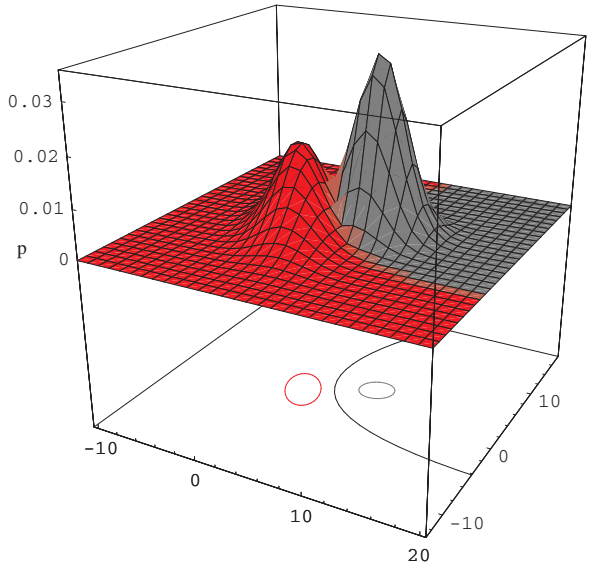
Resulting classifier

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

A Very Simple Classifier







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 two classes by a hyperplane.
- Then: Consider methods that do not require the decision surface (= the boundary between classes) to be linear (= a straight line or plane).
- Dealing with more than two classes.

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:

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

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.

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. If you prefer equations:

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

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

Recall

Two random variables are *stochastically independent*, or *independent* for short, if their joint distribution factorizes:

$$P(x, y) = P(x)P(y) \qquad \text{or} \qquad p(x, y) = p(x)p(y)$$

Dependent means *not independent*.

Intuitively

X and Y are dependent if knowing the outcome of X provides any information about the outcome of Y.

More precisely:

- If someone draws (X, Y) simultaneously, and only discloses $X = x$ to you, does that change your mind about the distribution of Y? (If so: Dependence.)
- Once X is given, the conditional distribution of Y is $P(Y|X = x)$.
- If that is still $P(Y = y)$, as before X was drawn, the two are independent. If $P(Y|X = x) \neq P(Y)$, they are dependent.

A few remarks

- Joint distributions of dependent variables can become very complicated. Dealing with joint distributions of many variables is one of the hardest problems in statistics and probability.
- The math almost always becomes easier if we assume variables are independent.
- On the other hand, assuming independence means we neglect all interactions between the effects represented by the variables.
- When we design probability models, there is usually a trade-off between simplicity (e.g. assuming everything is independent) and accuracy (trying to model all interactions precisely).

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

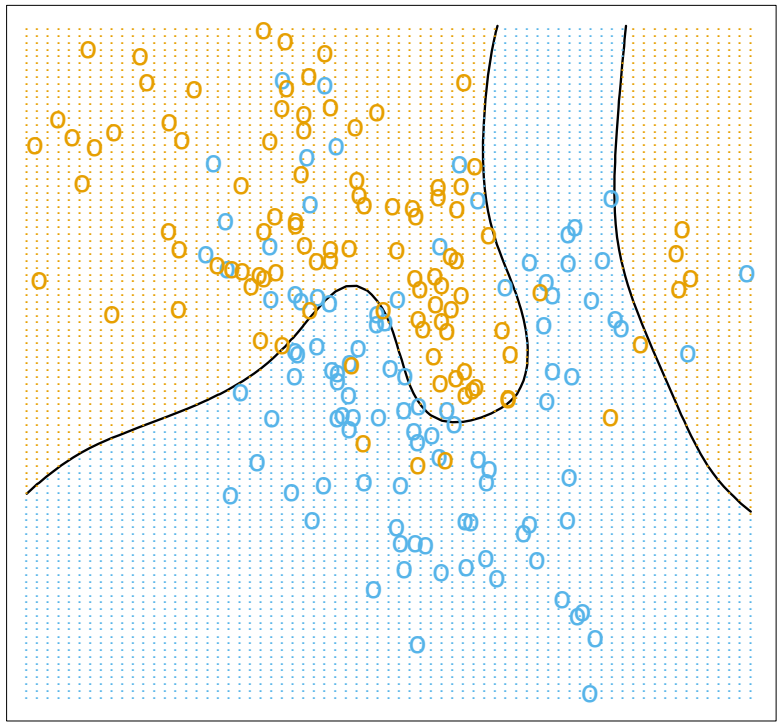
$$f(\mathbf{x}) = \arg\max_{y} \frac{p(\mathbf{x}|y)P(y)}{p(\mathbf{x})} = \arg\max_{y} p(\mathbf{x}|y)P(y)$$

If the class-conditional distribution is continuous, we use

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

Optimal classifier

- In the risk framework, the best possible classifier is the one which minimizes the 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).

BAYES-OPTIMAL CLASSIFIER

Suppose for simplicity we have to classes labeled "1" and "2", so $y \in \{1, 2\}$.

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

What do the terms mean?

- $P(y)$ = probability to observe class $Y = y$ if we draw (X, Y) from $p(x, y)$ and discard X.
- Approximately, this is the probability that a training data point is labeled y if we draw it uniformly from a very large training set (without looking at x).
- If both classes are equally probable (in terms of training data: equally large), then $P(y) = \frac{1}{2}$.
- $P(y|x)$: Fix a point x is space. What is the probability that a data point at this location belongs to class y?
- This is a number strictly between 0 and 1 if the classes "overlap" in space.

If classes are assumed equally large

$$f(\mathbf{x}) = \arg\max_y p(x|y)P(y) = \arg\max_y p(x|y)\tfrac{1}{2} = \arg\max_y p(x|y)$$

That means: The Bayes-optimal classifier is the one that assigns a point at location x to the class whose probability at x is larger, e.g. to class 1 if $P(1|x) \geq P(2|x)$.

EXAMPLE: SPAM FILTERING

Representing emails

- $\mathbf{Y} = \{$ spam, email $\}$
- $\mathbf{X} = \mathbb{R}^d$
- Each axis is labeled by one possible word.
- d = number of distinct words in vocabulary
- x_j = number of occurrences 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

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

Simplifying assumption

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

$$p(\mathbf{x}|y) = \prod_{j=1}^{d} p_j(x_j|y) \quad \text{for } \mathbf{x} = (x_1, \ldots, x_d) \ ,$$

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.

Class-conditional in the spam example

$P(x|y)$ is a multinomial (= categorical distribution). It is estimated as:

$$P(\text{word } i|y) = \frac{\#\text{ occurrences of word } i \text{ in emails of class } y}{\#\text{ occurrences of word } i \text{ in all emails}}$$

Linear Classification

Definition

For two vectors $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{R}^d$, the **scalar product** of $\mathbf{x}$ and $\mathbf{y}$ is

$$\langle \mathbf{x}, \mathbf{y} \rangle \quad := \quad x_1 y_1 + \ldots + x_d y_d \quad = \quad \sum_{i=1}^{d} x_i y_i$$

Note: $\langle \mathbf{x}, \mathbf{x} \rangle = \|\mathbf{x}\|^2$, so the Euclidean norm (= the length) of $\mathbf{x}$ is $\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$.

Linearity

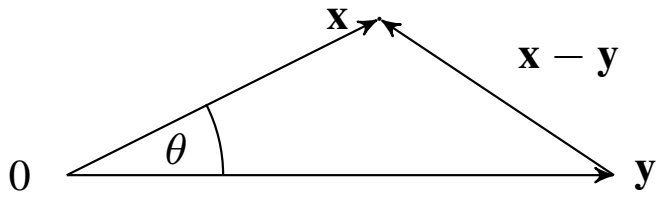
The scalar product is additive in both arguments,

$$\langle \mathbf{x} + \mathbf{z}, \mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{z}, \mathbf{y} \rangle \quad \text{and} \quad \langle \mathbf{x}, \mathbf{y} + \mathbf{z} \rangle = \langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{x}, \mathbf{z} \rangle$$

and scales as

$$\langle c \cdot \mathbf{x}, \mathbf{y} \rangle = c \cdot \langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{x}, c \cdot \mathbf{y} \rangle \qquad \text{for any } c \in \mathbb{R} \ .$$

Functions that are additive and scale-equivariant are called **linear**, so the scalar product is linear in both arguments.



Recall: The cosine rule

If two vectors $\mathbf{x}$ and $\mathbf{y}$ enclose an angle θ, then

$$\|\mathbf{x}-\mathbf{y}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\cos\theta\|\mathbf{x}\|\|\mathbf{y}\|$$

(If θ is a right angle, then $\cos\theta = 0$, and this becomes Pythogoras' $\|\mathbf{x}-\mathbf{y}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2$.)

Cosine rule for scalar products

It is easy to check that

$$\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - \|\mathbf{x}-\mathbf{y}\|^2 = 2\langle\mathbf{x},\mathbf{y}\rangle$$

Substituting gives

$$2\cos\theta\|\mathbf{x}\|\|\mathbf{y}\| = 2\langle\mathbf{x},\mathbf{y}\rangle$$

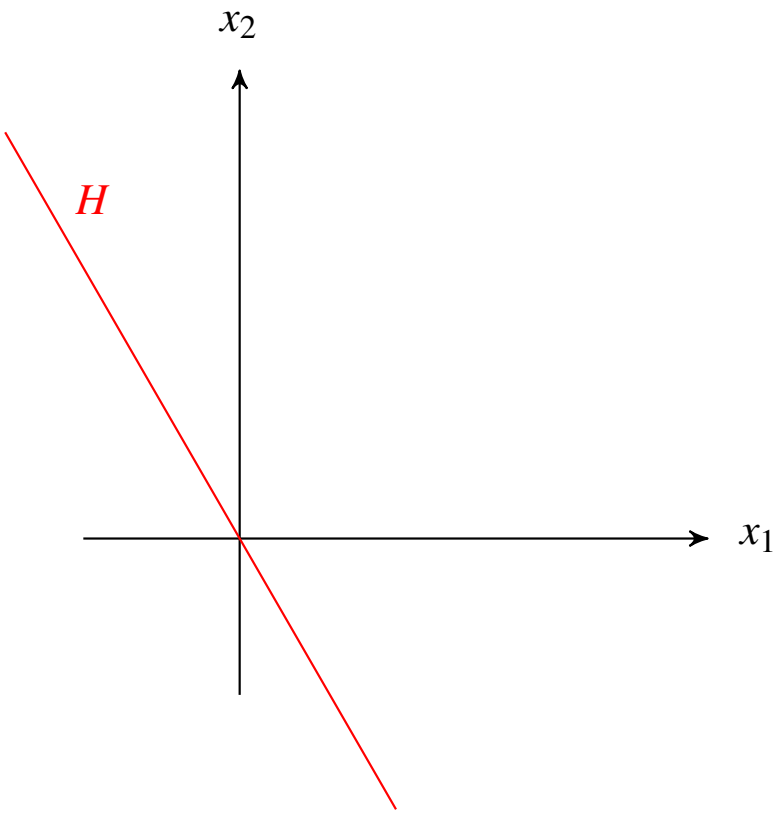
and hence

$$\cos\theta = \frac{\langle\mathbf{x},\mathbf{y}\rangle}{\|\mathbf{x}\|\|\mathbf{y}\|}$$

Consequences of the cosine rule

The scalar product satisfies $\langle \mathbf{x}, \mathbf{y} \rangle = \|\mathbf{x}\| \|\mathbf{y}\|$ if and only if $\mathbf{x}$ and $\mathbf{y}$ are parallel, and

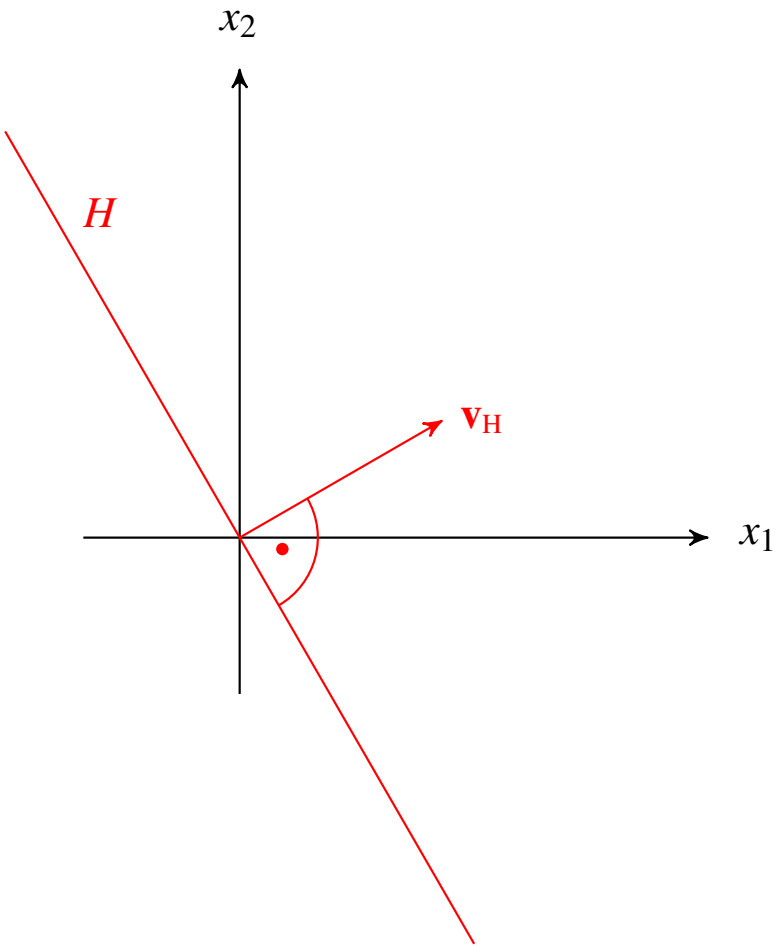
$$\langle \mathbf{x}, \mathbf{y} \rangle = 0 \qquad \text{if and only if } \mathbf{x} \text{ and } \mathbf{y} \text{ are orthogonal.}$$



Hyperplanes

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

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



Hyperplanes

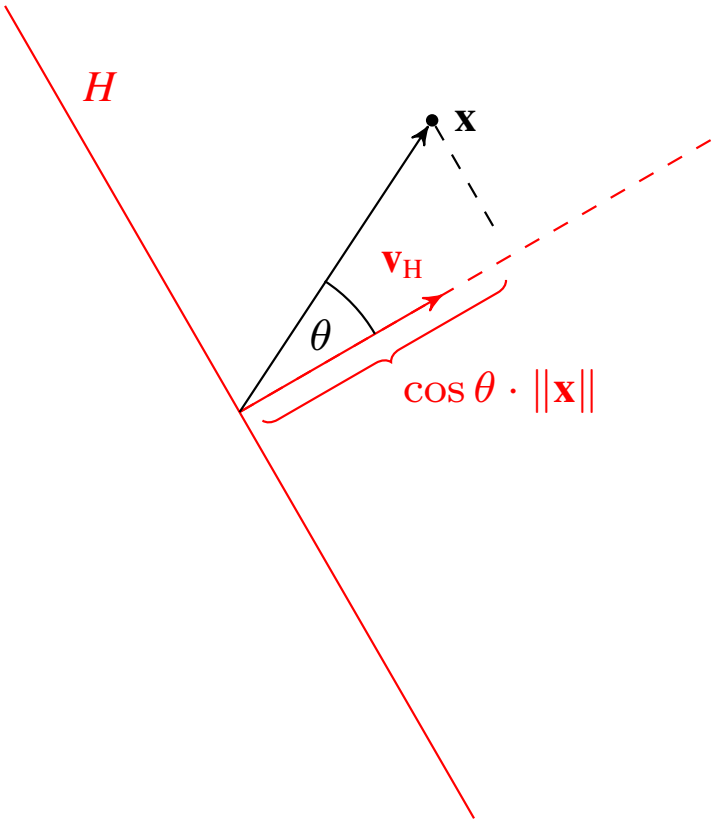
- Consider a hyperplane H in $\mathbb{R}^d$. Think of H as a set of points.
- Each point $\mathbf{x}$ in H is a vector $\mathbf{x} \in \mathbb{R}^d$.
- Now draw a vector $\mathbf{v}_H$ that is orthogonal to H.
- Then any vector $\mathbf{x} \in \mathbb{R}^d$ is a point in H if and only if $\mathbf{x}$ is orthogonal to $\mathbf{v}_H$.
- Hence:

$$\mathbf{x} \in H \qquad \Leftrightarrow \qquad \langle \mathbf{x}, \mathbf{v}_H \rangle = 0 \ .$$

- If we choose $\mathbf{v}_H$ to have length $\|\mathbf{v}_H\| = 1$, then $\mathbf{v}_H$ is called a **normal vector** of H.

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

WHICH SIDE OF THE PLANE ARE WE ON?



Distance from the plane

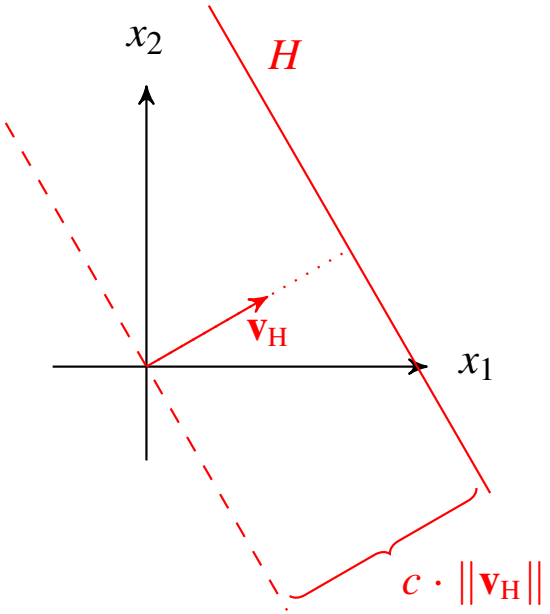
- The projection of $\mathbf{x}$ onto the direction of $\mathbf{v}_H$ has length $\langle \mathbf{x}, \mathbf{v}_H \rangle$ *measured in units of* $\mathbf{v}_H$, i.e. length $\langle \mathbf{x}, \mathbf{v}_H \rangle / \|\mathbf{v}_H\|$ in the units of the coordinates.
- By cosine rule: The distance of $\mathbf{x}$ from the plane is

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

Which side of the plane?

- The cosine satisfies $\cos\theta > 0$ iff $\theta \in (-\frac{\pi}{2}, \frac{\pi}{2})$.
- We can decide which side of the plane $\mathbf{x}$ is on using

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



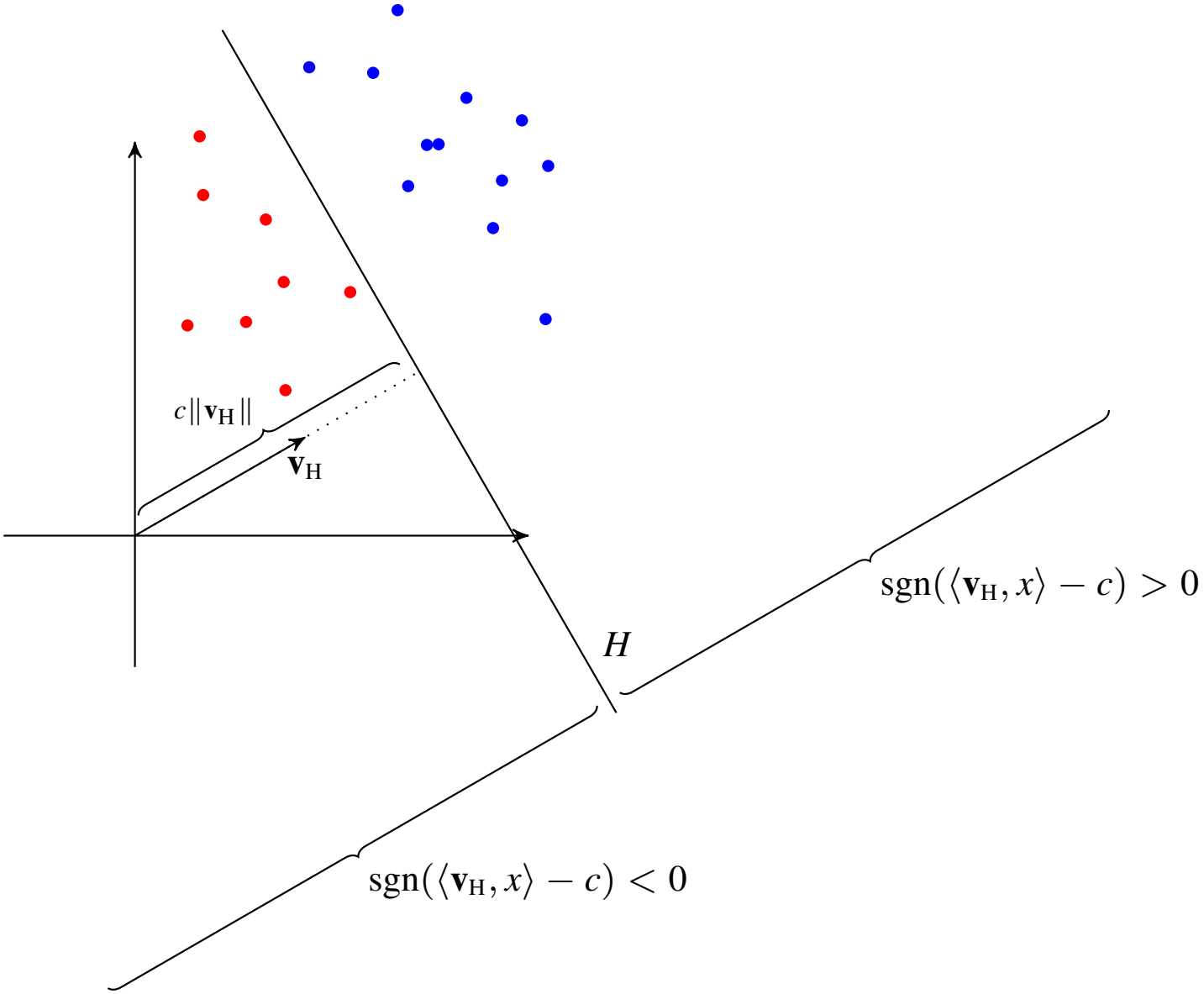
Affine Hyperplanes

- An **affine hyperplane** $H_\mathbf{w}$ is a hyperplane shifted by a vector $\mathbf{w}$,
$$H_\mathbf{w} = H + \mathbf{w} \ .$$
(That means $\mathbf{w}$ is added to each point $\mathbf{x}$ in H.)
- We choose $\mathbf{w}$ in the direction of $\mathbf{v}_H$, so
$$\mathbf{w} = c \cdot \mathbf{v}_H \quad \text{for some } c > 0 \ .$$

Which side of the plane are we on?

- 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:

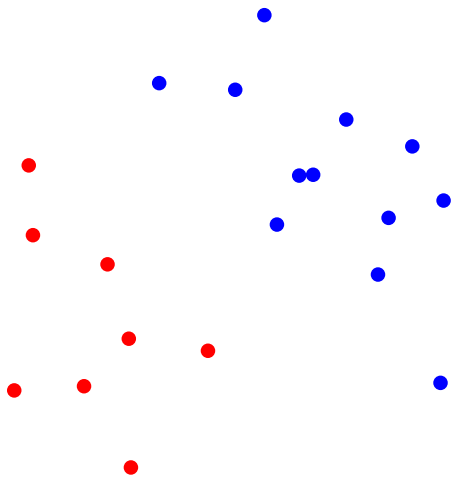
- We usually assume $\mathbf{v}_{\mathrm{H}}$ to be a unit vector. If it is not, f_{H} still defines a linear classifier, but c describes a shift of a different length.
- Specifying a linear classifier in $\mathbb{R}^d$ requires $d + 1$ scalar parameters.

Definition

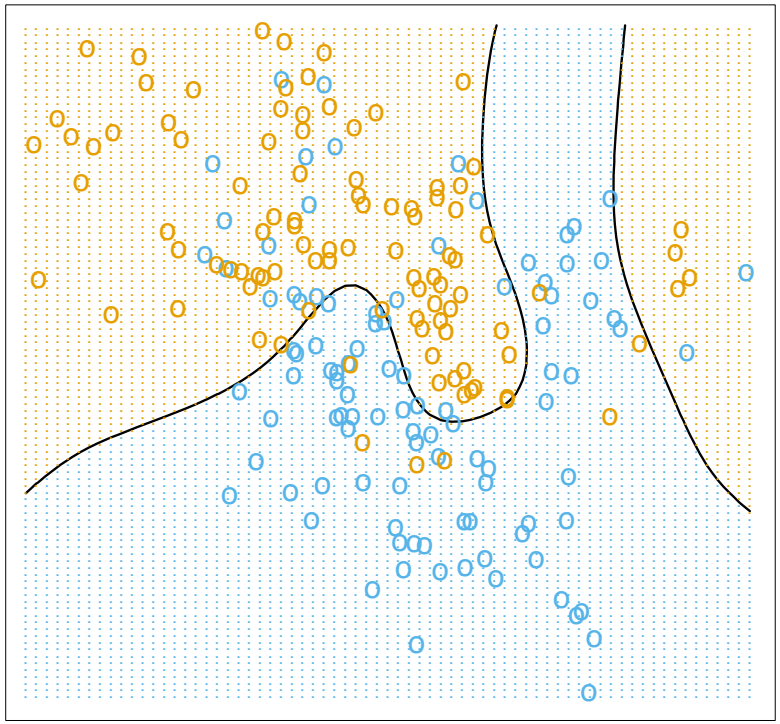
Two sets $A, B \in \mathbb{R}^d$ are called **linearly separable** if there is an affine hyperplane H which separates them, i.e. which satisfies

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

Linear Separability



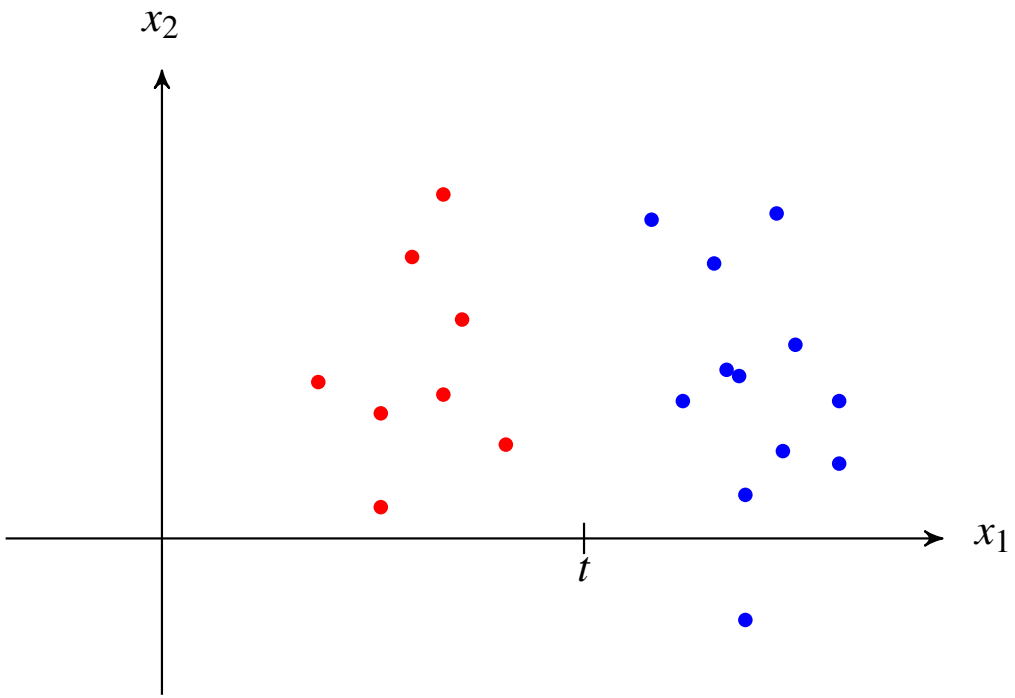
linearly separable



not linearly separable

Linear Separability

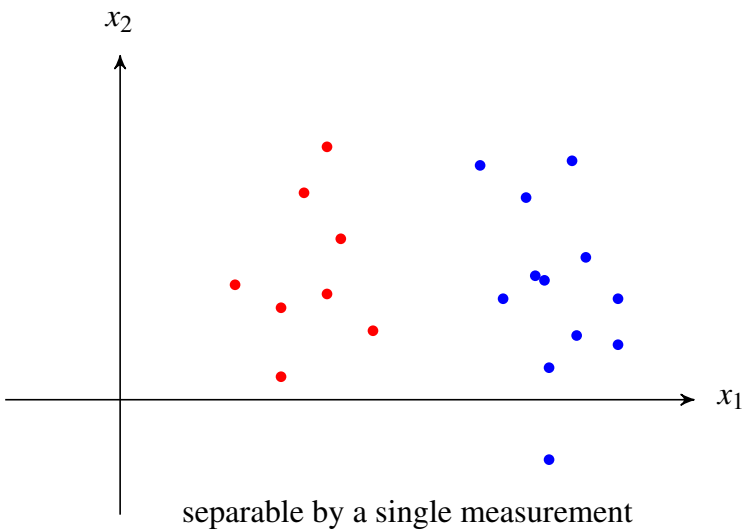
- Recall that when data is represented by points in $\mathbb{R}^d$, each axis represents a quantity that is measured (a "variable").
- If there exists a single variable that distinguishes two classes, these classes can be distinguished along a single axis.



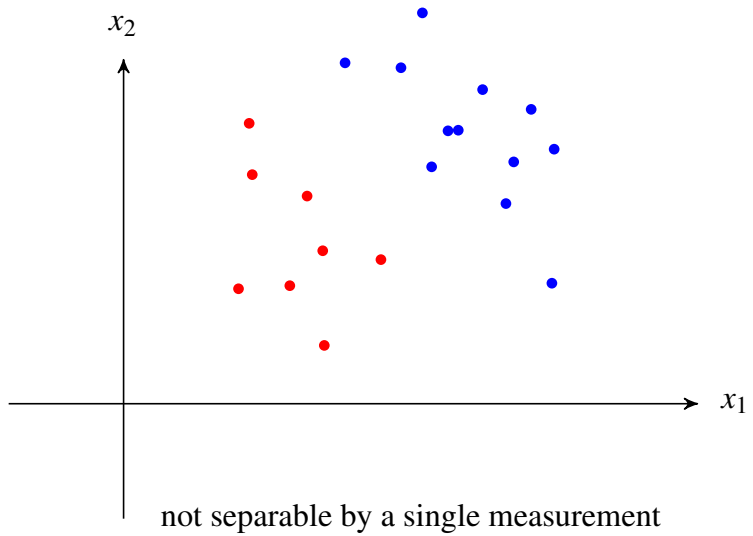
- In this illustration, we could classify by a "threshold point" t on the line.

Linear Separability

- Even if classes cannot be distinguished by a single variable, they may be distinguishable by a combination of several variables.
- That is the case for linearly separable data. The threshold point along x_1 is now a function of the threshold point along x_2, and vice versa. Linearly separable also implies this function is linear.

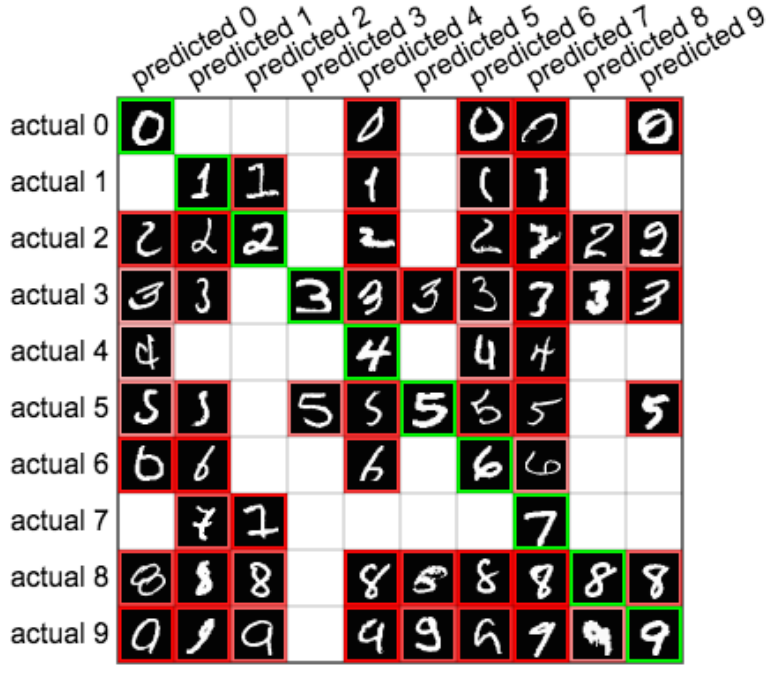


separable by a single measurement



not separable by a single measurement

but linearly separable


predicted 0
predicted 1
predicted 2
predicted 3
predicted 4
predicted 5
predicted 6
predicted 7
predicted 8
predicted 9
actual 0
actual 1
actual 2
actual 3
actual 4
actual 5
actual 6
actual 7
actual 8
actual 9

Multiple Classes

More than two classes

For some classifiers, multiple classes are natural. We have already seen one:

- Simple classifier fitting one Gaussian per class.

We will discuss more examples soon:

- Trees.
- Ensembles: Number of classes is determined by weak learners.

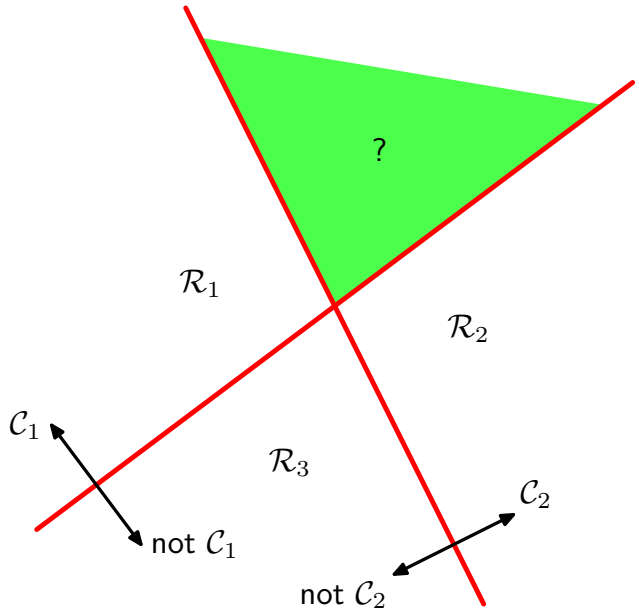
Exception: All classifiers based on hyperplanes.

Linear Classifiers

Approaches:

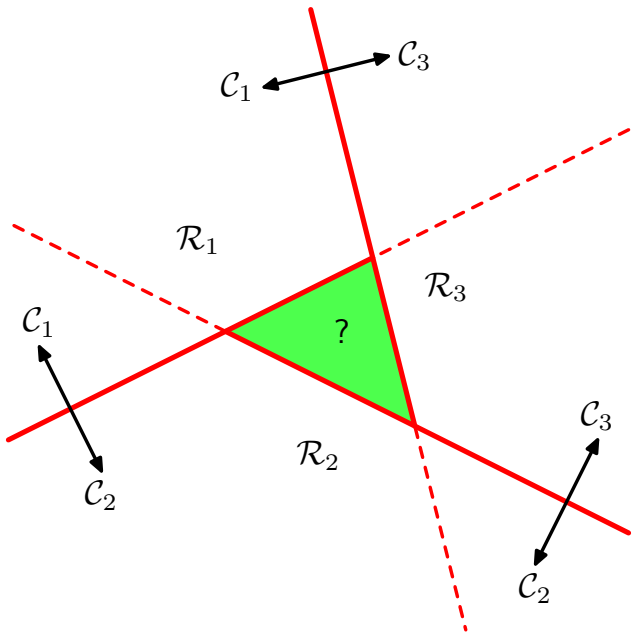
- One-versus-all (more precisely: one-versus-the-rest) classification.
- One-versus-one classification.
- Multiclass discriminants.

One-Versus-All Classification



- One linear classifier per class.
- Classifies "in class k" versus "not in class k".
- This is a two-class classifier that defines:
  - Positive class = $\mathcal{C}_k$.
  - Negative class = $\cup_{j \neq k} \mathcal{C}_j$.
- Problem: Ambiguous regions (green in figure).

One-Versus-One Classification



- One linear classifier for each pair of classes (i.e. $\frac{K(K-1)}{2}$ in total).
- Classify by majority vote.
- Problem again: Ambiguous regions.

Linear classifier

- Recall: Decision rule is $f(\mathbf{x}) = \operatorname{sgn}(\langle \mathbf{x}, \mathbf{v}_{\mathrm{H}} \rangle - c)$
- Idea: Combine classifiers *before* computing sign. Define

$$g_k(\mathbf{x}) := \langle \mathbf{x}, \mathbf{v}_k \rangle - c_k$$

Multiclass linear discriminant

- Use one classifier g_k (as above) for each class k.
- Trained e.g. as one-against-rest.
- Classify according to

$$f(\mathbf{x}) := \arg\max_k \{g_k(\mathbf{x})\}$$

- If $g_k(\mathbf{x})$ is positive for several classes, a larger value of g_k means that $\mathbf{x}$ lies "further" into class k than into any other class j.
- If $g_k(\mathbf{x})$ is negative for all k, the maximum means we classify $\mathbf{x}$ according to the class represented by the closest hyperplane.

Problem

- Multiclass discriminant idea: Compare distances to hyperplanes.
- Works if the orthogonal vectors $\mathbf{v}_H$ determining the hyperplanes are normalized.
- For some of the best training methods for linear classifiers, that does not work well.

Optimization

Recall from classification

- We "train" e.g. a linear classifier by finding the affine plane for which the empirical risk defined by a given loss function becomes as small as possible.

This is an example of phrasing a problem as an "optimization problem":

- There is a real-valued function (here: the empirical risk) that measures how good a given solution is.
- We choose that solution for which this function is minimal.

More generally

A variety of problems in statistics, machine learning and data mining are phrased as optimization problems:

- Fitting a parametric model: Maximum likelihood
- Training a classifier: Minimize an empirical risk under a given loss function
- Linear regression: Minimize a least squares error
- Sparse regression: Minimize a penalized least squares error
- Training neural networks: Minimize an empirical risk; loss can be chosen for classification or for regression task.

Min and Argmin

$$\min_x f(x) = \text{ smallest value of } f(x) \text{ for any } x$$

$$\arg\min_x f(x) = \text{ value of } x \text{ for which } f(x) \text{ is minimal}$$

Minimum with respect to subset of arguments

$$\min_x f(x, y) = \text{ smallest value of } f(x, y) \text{ for any } x \text{ if } y \text{ is kept fixed}$$

Optimization problem

For a given function $f : \mathbb{R}^d \to \mathbb{R}$, a problem of the form

$$\text{find} \quad \mathbf{x}^* := \arg\min_{\mathbf{x}} f(\mathbf{x})$$

is called an **minimization problem**. If $\arg\min$ is replaced by $\arg\max$, it is a **maximization problem**. Minimization and maximization problems are collectively referred to as **optimization problems**.

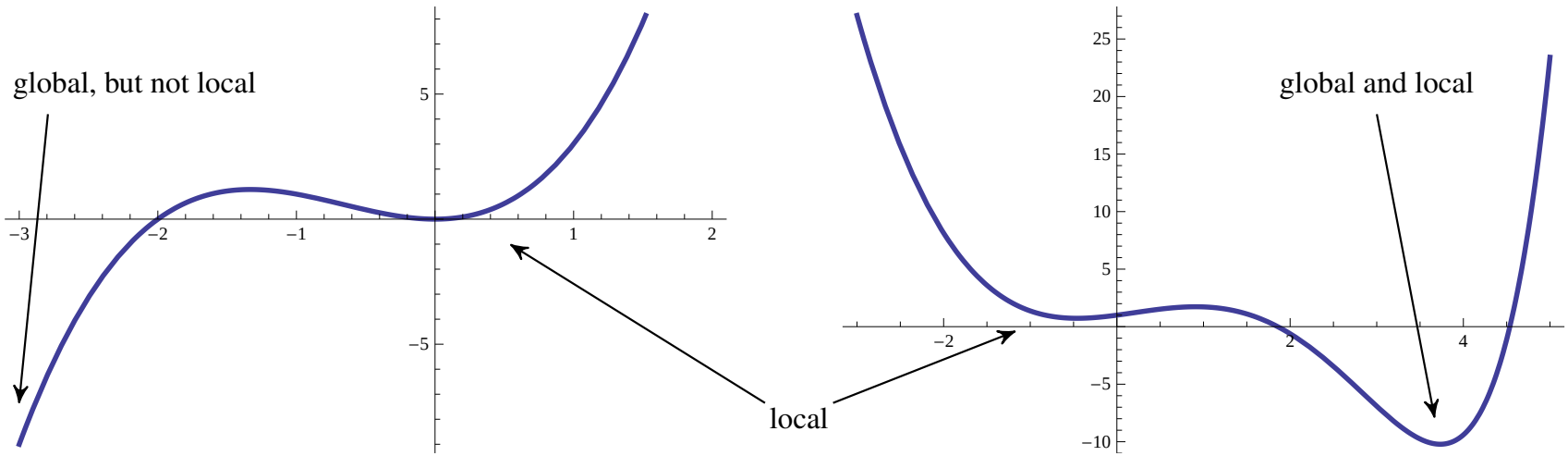
Minimization vs Maximization

For any function f, we have

$$\min f(x) = -\max(-f(x)) \qquad \text{and} \qquad \arg\min f(x) = \arg\max(-f(x))$$

That means:

- If we know how to minimize, we also know how to maximize, and vice versa.
- We do not have to solve both problems separately; we can just generically discuss minimization.



Local and global minima

A minimum of f at x is called:

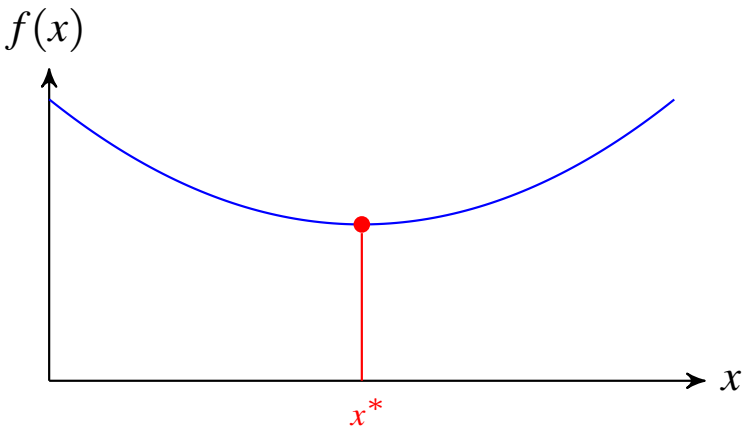
- **Global** if f assumes no smaller value on its domain.
- **Local** if there is some open interval (a, b) containing x such that $f(x)$ is a global minimum of f restricted to that interval.

Typical situation

- Given is a function $f : \mathbb{R}^d \to \mathbb{R}$.
- The dimension d is usually very large.
  (In neural network training problems: Often in the millions.)
- We cannot plot or "look at" the function.
- We can only evaluate its value $f(x)$ point by point.

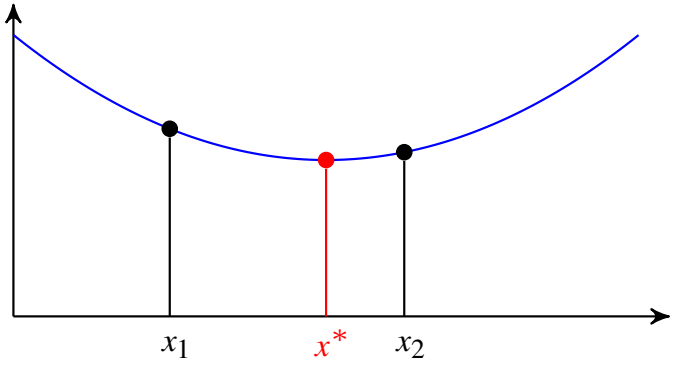
One-dimensional illustration

Here, $d = 1$ (but keep in mind we are interested in very large d.)

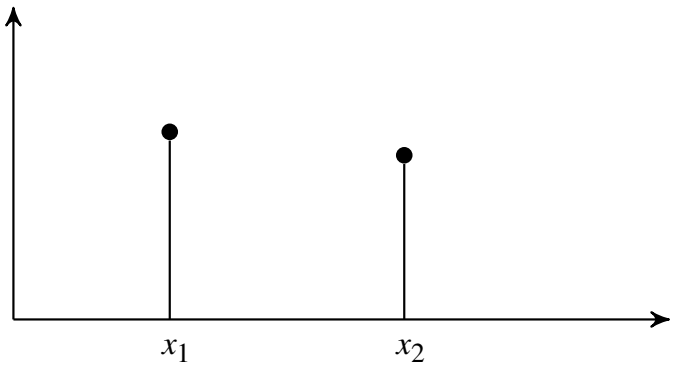


The minimizer we are interested in is x^*.

One-dimensional Illustration



- Our goal is to find x^*.
- We can evaluate the function at points of our choice, say x_1 and x_2.



- However, we cannot "see" the function.
- All we know are values at a few points.

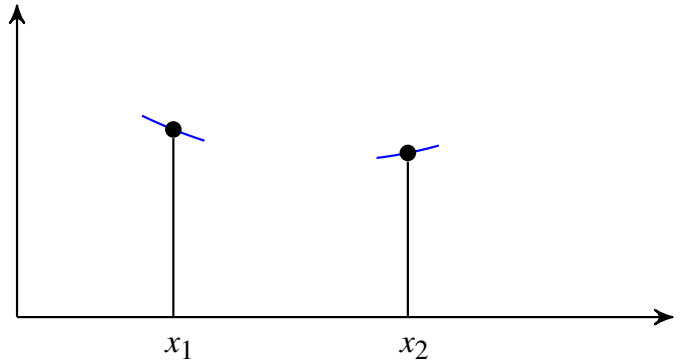
Task

Based on the values we know, we have to:

- Either make a decision what x^* is.
- Or gather more information, by evaluating f at additional points. In that case, we have to decide which point to evaluate next.

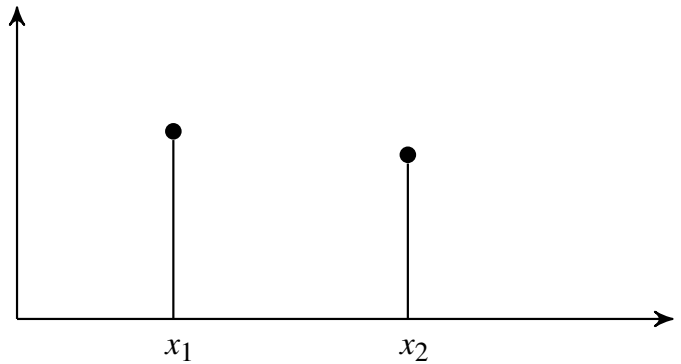
NEXT STEPS

- We will first consider how we would proceed if we had access to the entire function in a small neighborhood around each of the points $x_1, x_2, \ldots$, i.e. if we could see something like this:

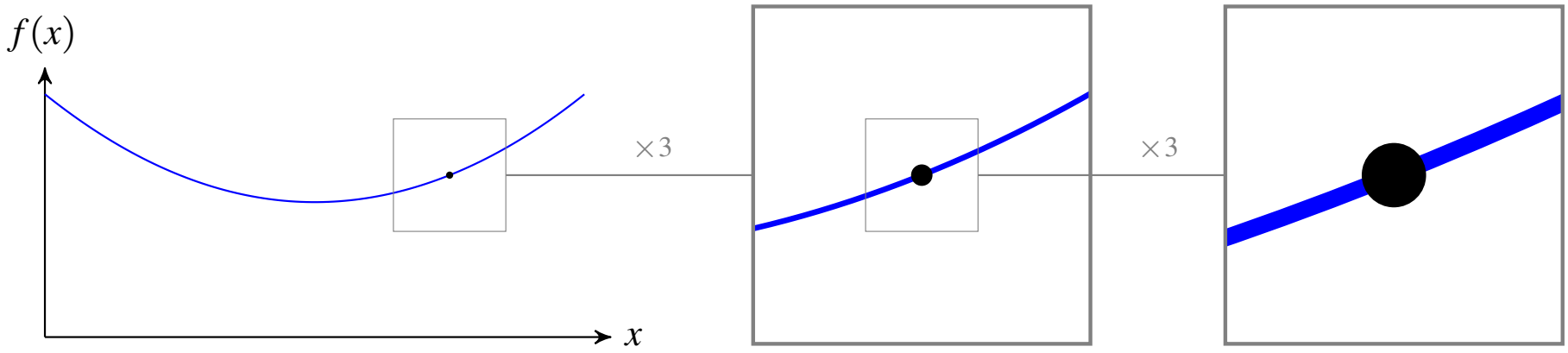


  To this end, we discuss the concept of a derivative.

- We then consider what we can actually implement on a computer, given that we only have access to point-wise information:



Zooming in on a smooth function



Observation

- Each time we zoom in, the curve looks more like a straight line.
- If we zoom in far enough, we can replace the curve in a small area around the marked point by a straight line.
- In mathematical jargon, that is called an *approximation*: We replace the curve around the marked point by a surrogate curve. If that surrogate is a straight line (i.e. a linear function), it is a *linear approximation*.

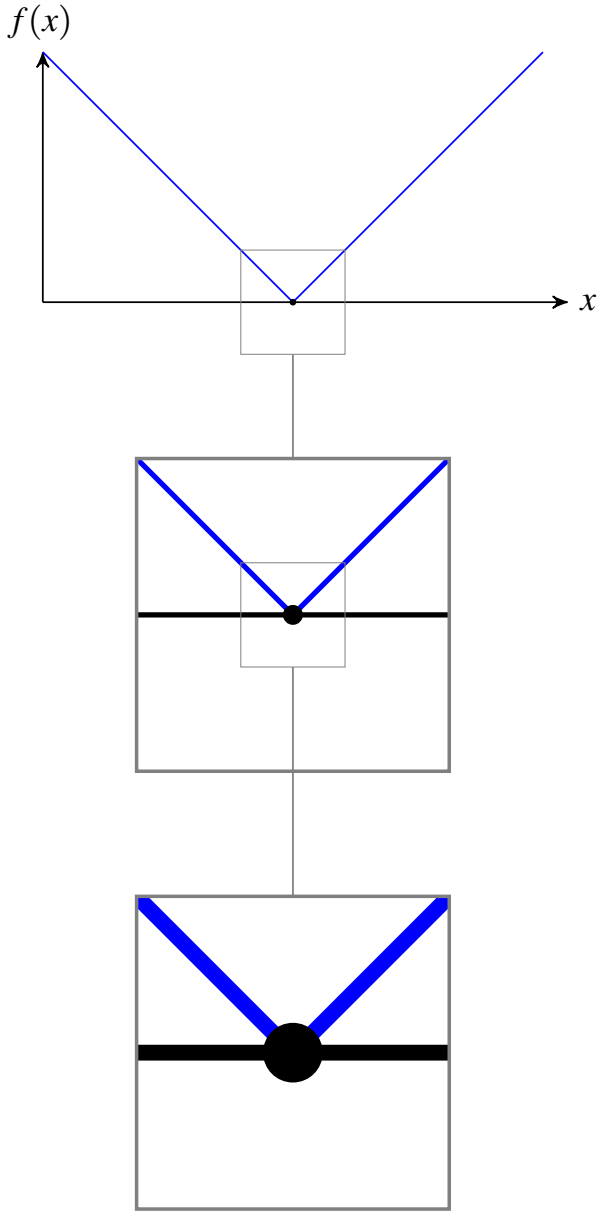
Zooming in on a smooth function

A counter example

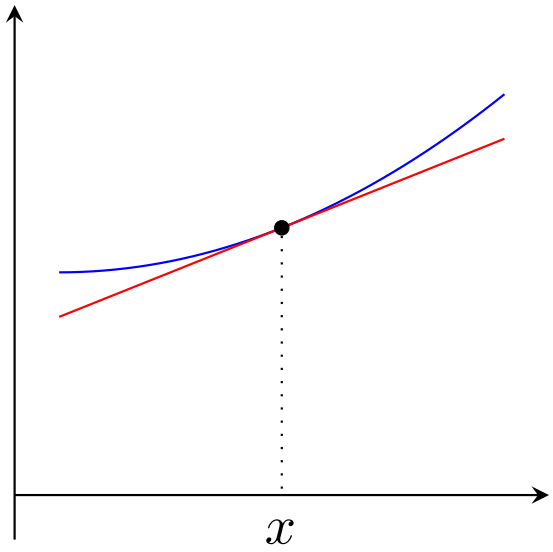
- Not every function has this property.
- Here, we consider the absolute value function $f(x) = |x|$, and zoom in on the point $x = 0$.
- In this case, the shape of f never seems to change.
- Note this would be different if we had picked any other point than $x = 0$.

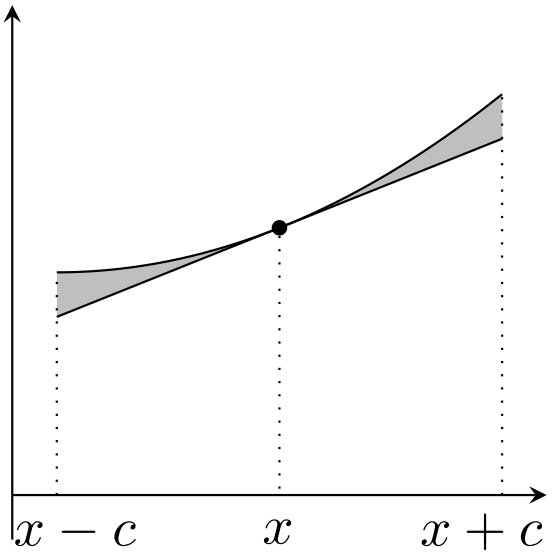
We observe

- Whether a function is "locally straight" is a property that may be true at some points, but not at others.
- Clearly it matters whether the function is "smooth" around the point we focus on.



Approximating by a Straight Line

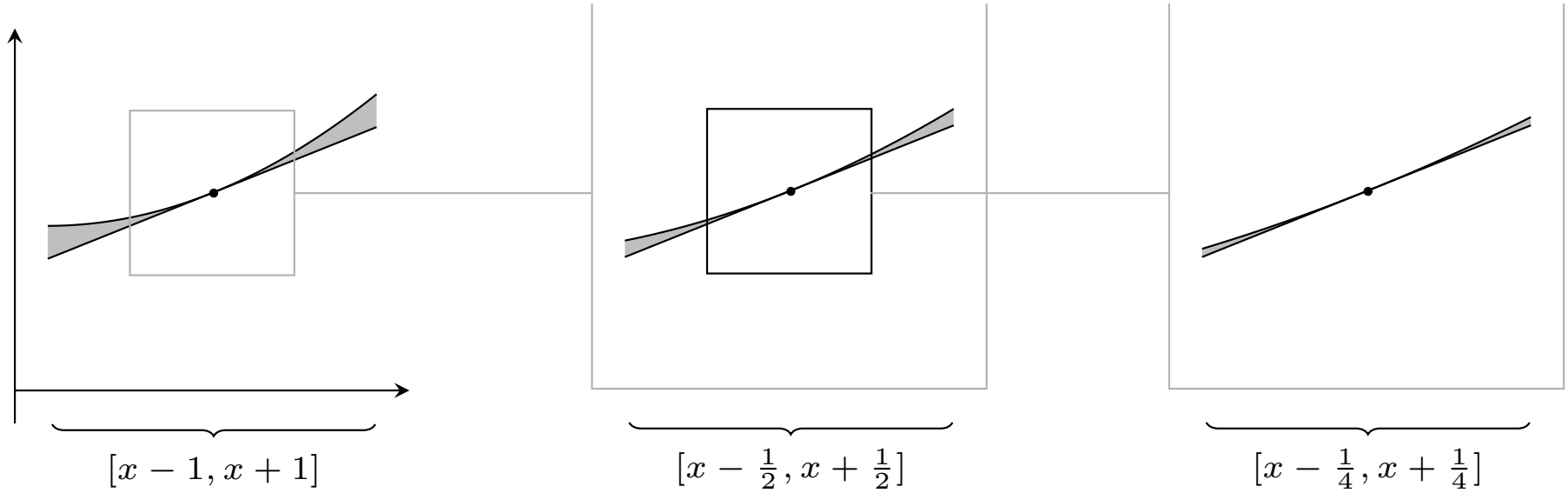




- We consider a function (blue) and approximate it at a point x by a straight line (red).
- To measure how good the approximation is, we fix a constant $c > 0$ and enclose x in the interval $[x - c, x + c]$.
- On this interval, we compute the area between the two functions (shaded in gray). Suppose this area is $A(x, c)$.
- Of course, $A(x, c)$ will grow if we make c larger. To make the area comparable for different values of c, we use the *relative* approximation error

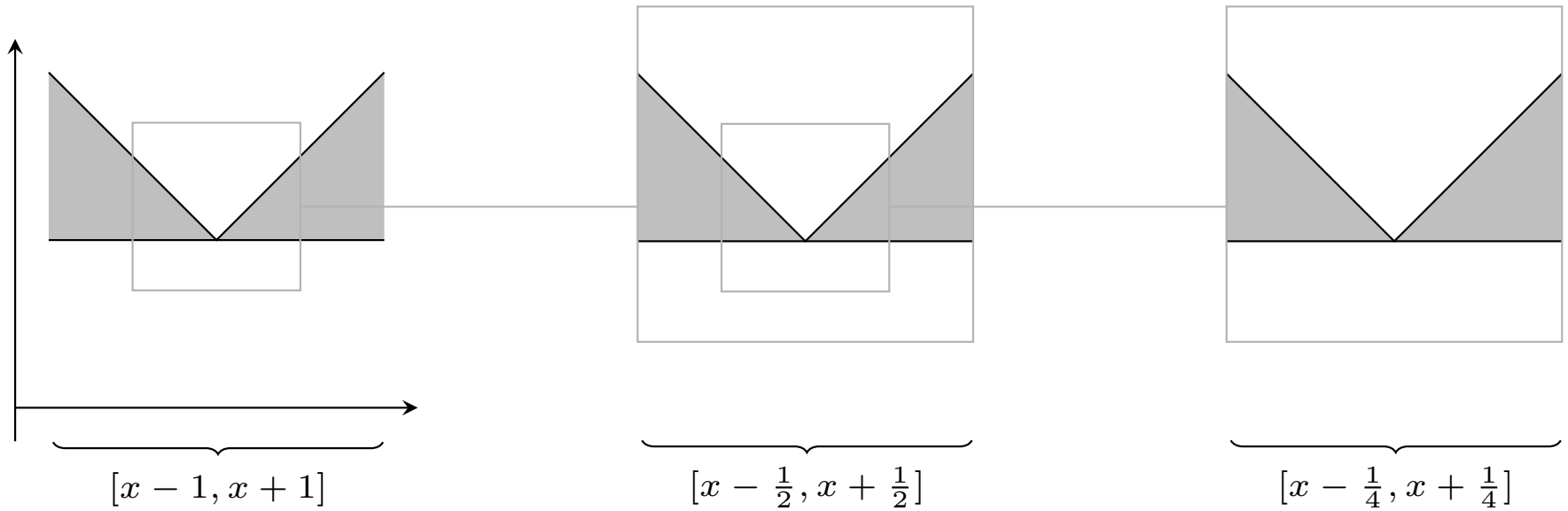
$$r(c) = \frac{A(x, c)}{|[x - c, x + c]|} = \frac{A(x, c)}{2c} \ .$$

Approximating by a Straight Line



- Now consider what happens if we zoom in, by making c smaller and smaller.
- If the function is smooth, we observe the relative error becomes smaller each time.
- The function can be approximated by the line to arbitrary precision, that is: If we are permitted *any* error $\varepsilon > 0$, we can always find a small enough c such that $r(c) < \varepsilon$.
- In this sense, the linear approximation (= approximation by a straight line) is *locally exact*.
- If a straight line can be chosen for f and x such that the relative approximation error can be made arbitrarily small by making the intervall sufficiently small, then f is called **differentiable at** x.

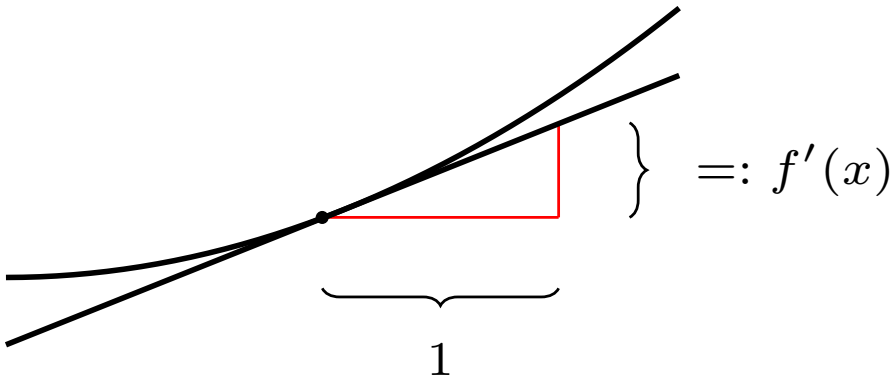
ZOOMING IN ON A NON-SMOOTH FUNCTION



Now try the same for the absolute value function:

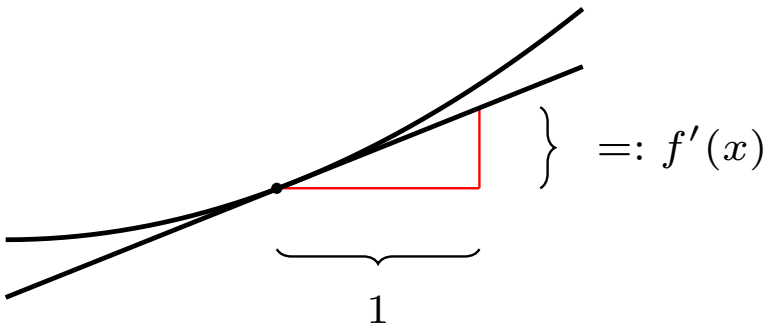
- Approximate it at $x = 0$ by a horizontal line.
- Here, the relative error around $x = 0$ remains the same regardless of how we choose c.
- We could also use an approximating line with a different slope, and would encounter the same problem.
- Thus, $|x|$ is not differentiable at $x = 0$ (although it is differentiable at every other point x).

The Derivative

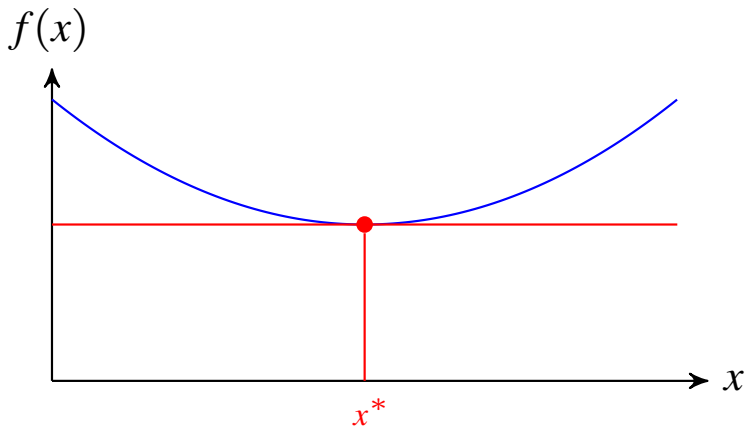


- If f is differentiable at x, there is a unique approximating line at x for which the relative error is minimal as c gets smaller.
- We can measure the slope of this line by substracting its values at $x + 1$ and x.
- We denote this slope by $f'(x)$ and call it the **derivative** of f at x.
- If f is differentiable at every point x, we can compute the value $f'(x)$ at every point, so f' is again a function. In general, it takes different values at different points x.

Some Properties of the Derivative

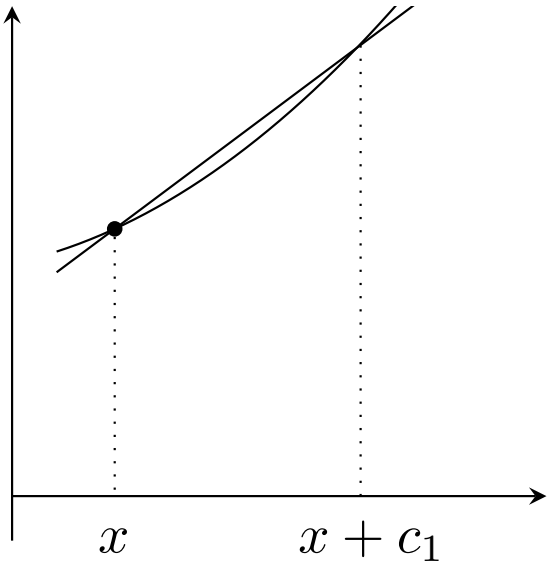


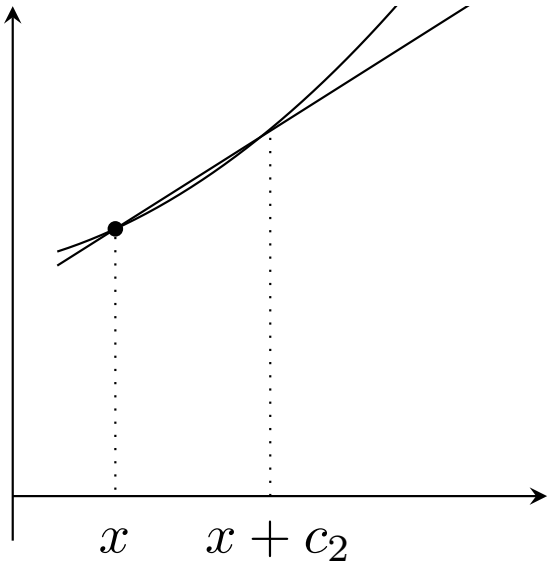
- If f increases around x, then $f'(x) > 0$. If f decreases, then $f'(x) < 0$.
- Recall that we are interested in finding minima and maxima. If f is differentiable at x and x is a local minimum or maximum, the approximating line is horizontal:



That means: At a (differentiable) maximum or minimum x^*, we have $f'(x^*) = 0$.

Finding the Derivative





- We fix a constant $c > 0$ and draw a straight line through the points $(x, f(x))$ and $(x + c, f(x + c))$. The slope of that line is

$$\frac{f(x + c) - f(x)}{c}$$

- Now make c smaller and smaller: Choose $c_1 > c_2 > \ldots$, for example $c_n = \frac{1}{n}$.
- We then ask what happens as c gets infinitely small, i.e. we try to find the limit

$$\lim_{n \to \infty} \frac{f(x + c_n) - f(x)}{c_n}$$

- If f is differentiable, this limit exists, and its slope is exactly that of the best possible linear approximation. That is, the limit is $f'(x)$.
- If the limit does not exist, f is not differentiable at x.

Not examinable.

SUMMARY

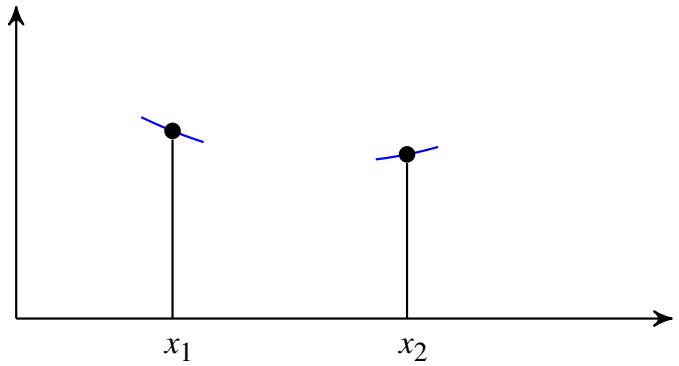
The derivative of a function f at a point x is the the slope of the locally best linear approximation to f around x.

If you are not familiar with calculus, keep in mind:

- The derivative $f'(x)$ exists if f is "sufficiently smooth" at x.
- Sign: The derivative is positive if f increases at x, negative if it decreases, and 0 if f is a maximum or minimum.
- Magnitude: The absolute value $|f'(x)|$ is the larger the more rapidly f changes around x.

Back to Optimization

Recall that we had asked: How can would we find a minimum if we had access to the entire function in a small neighborhood around points $x_1, x_2, \ldots$ that we are allowed to choose?



- If we can compute the derivatives $f'(x_1)$ and $f'(x_2)$, we have (the slope of) linear approximations to f at both points that are locally exact.
- That is: We can substitute the derivatives for the two short blue lines in the figure.
- We can tell from the sign of the derivative in which direction the function decreases.
- We also know that $f'(x) = 0$ if x is a minimum.

Basic idea

Start with some point x_0. Compute the derivative $f'(x_0)$ at x. Then:

- "Move downhill": Choose some $c > 0$, and set $x_1 = x_0 + c$ if $f'(x_0) < 0$ and $x_1 = x_0 - c$ if $f'(x_0) > 0$.
- Compute $f'(x_1)$. If it is 0 (possibly a minimum), stop.
- Otherwise, move downhill from x_1, etc.

Observations

- Since the sign of f' is determined by whether f increases or decreases, we can summarize the case distinction above by setting

$$x_1 = x_0 - \text{sign}(f'(x_0)) \cdot c$$

- If f changes rapidly, it may be a good strategy to make a large step (choose a large c), since we presumably are still far from the minimum. If f changes slowly, c should be small.
- One way of doing so is to choose c as the magnitude of f', since $|f'|$ has exactly this property. In that case:

$$x_1 = x_0 - \text{sign}(f'(x_0)) \cdot |f'(x_0)| = x_0 - f'(x_0)$$

The algorithm obtained by replying this step repeatedly is called **gradient descent**.

Gradient Descent

Gradient descent searches for a minimum of a differentiable function f.

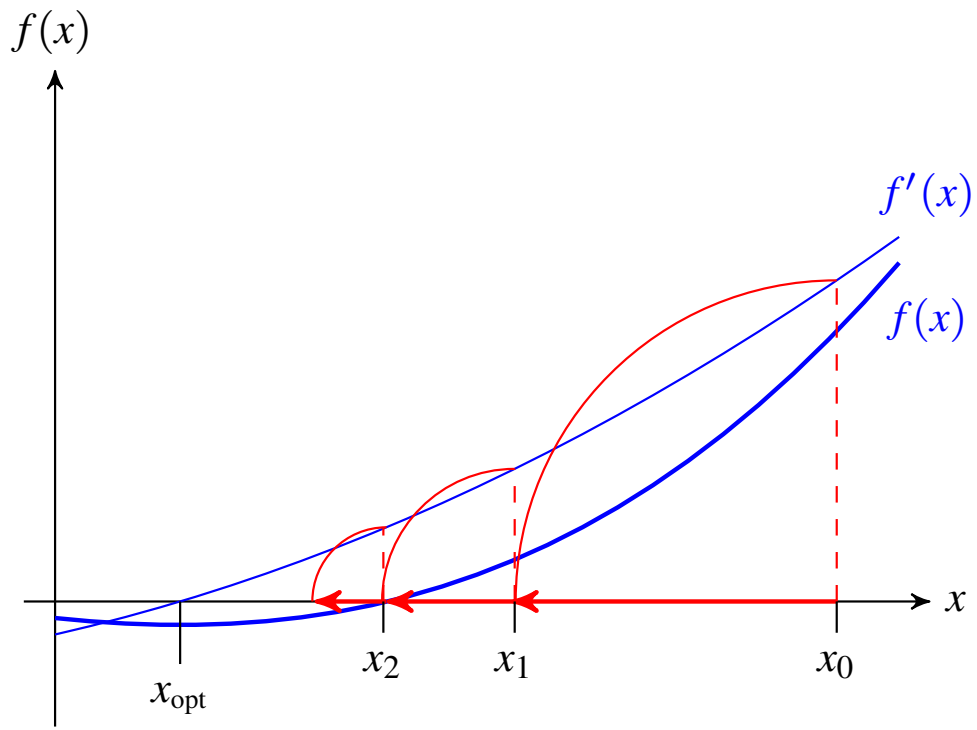
Algorithm

Start with some point $x_0 \in \mathbb{R}$ and fix a precision $\varepsilon > 0$.

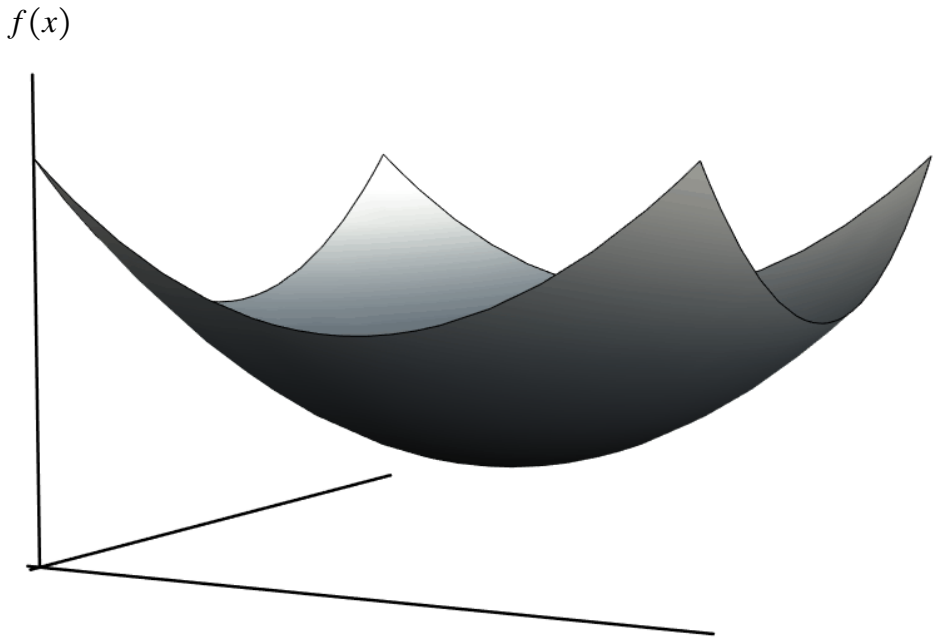
Repeat for $n = 1, 2, \ldots$:

1. Check whether $|f'(x_n)| < \varepsilon$. If so, report the solution $x^* := x_n$ and terminate.
2. Otherwise, set

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

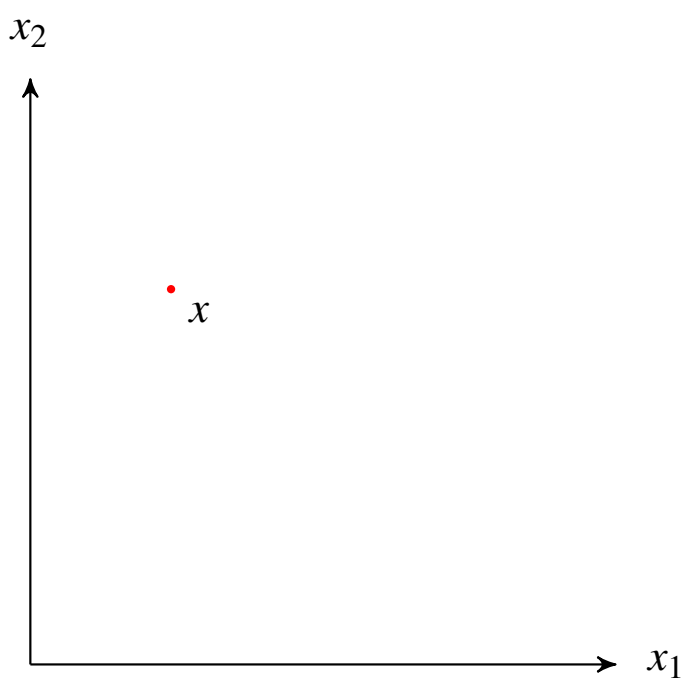


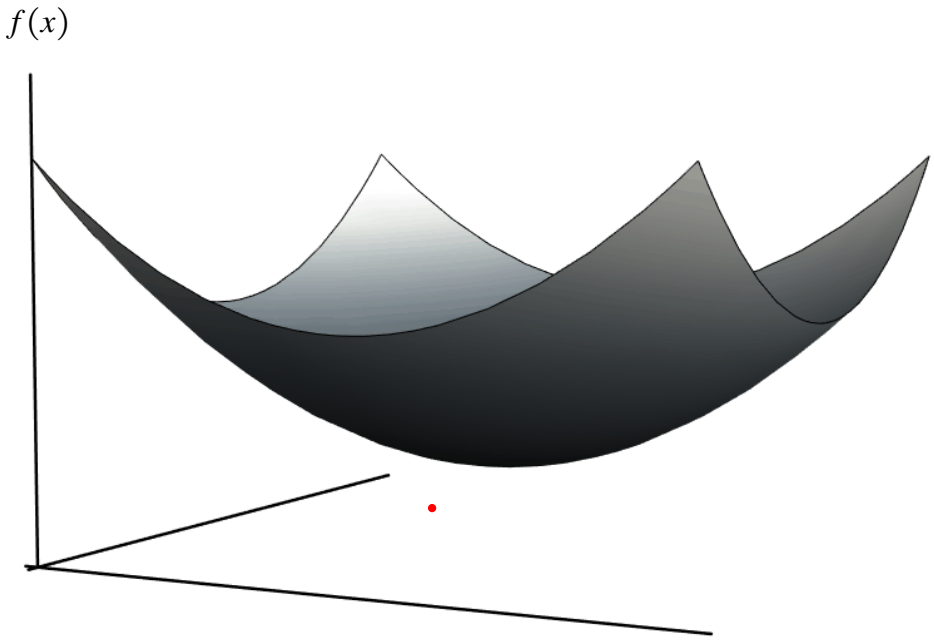
Derivatives in Multiple Dimensions



- We now ask how to define a derivative in multiple dimensions.
- Consider a function $f : \mathbb{R}^d \to \mathbb{R}$. What is the derivative of f at a point x?
- For simplicity, we assume $d = 2$ (so that we can plot the function).

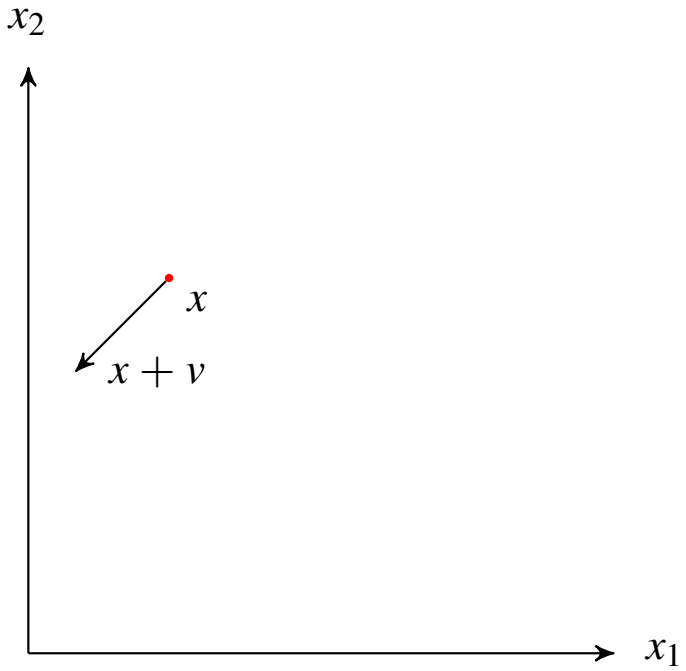
Derivatives in Multiple Dimensions

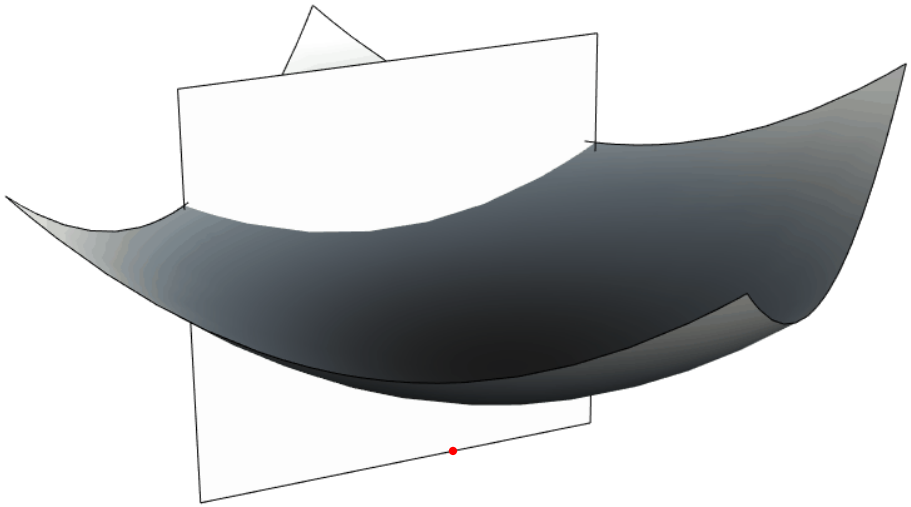




- We fix a point $x = (x_1, x_2)$ in $\mathbb{R}^2$, marked red above.
- We will try to turn this into a 1-dimensional problem, so that we can use the definition of a derivative we already know.

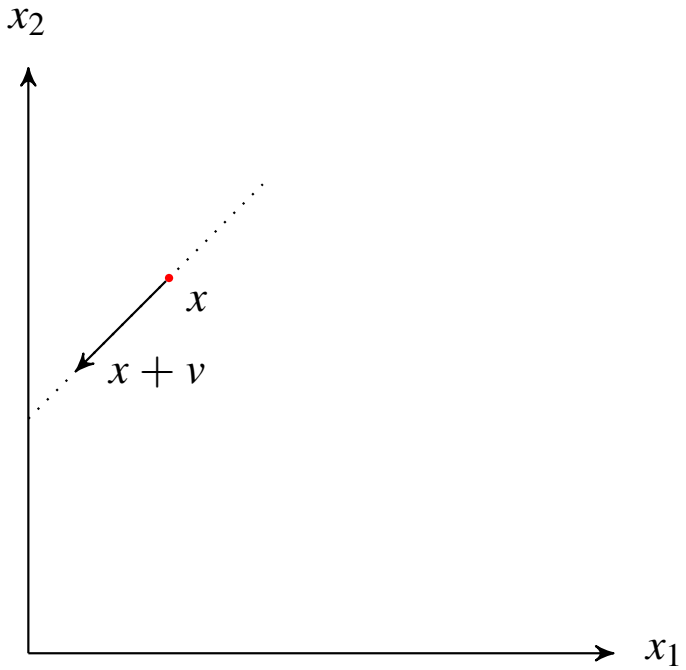
Reducing to one Dimension

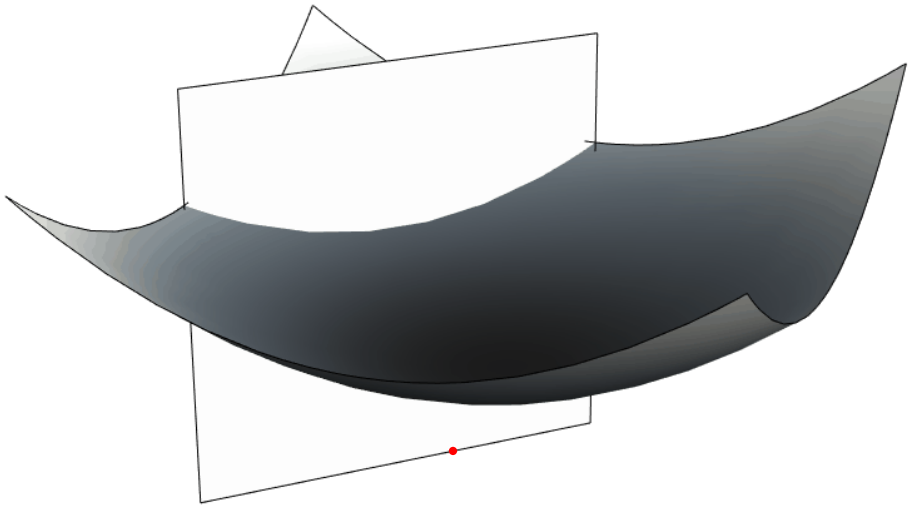




- To make the problem 1-dimensional, fix some vector $v \in \mathbb{R}_2$, and draw a line through x in direction of v.
- Then intersect f with a plane given by this line: In the coordinate system of f, choose the plane that contains the line and is orthogonal to $\mathbb{R}^2$.
- The plane contains the point x.
- Note we can do that even if $d > 2$. We still obtain a plane.

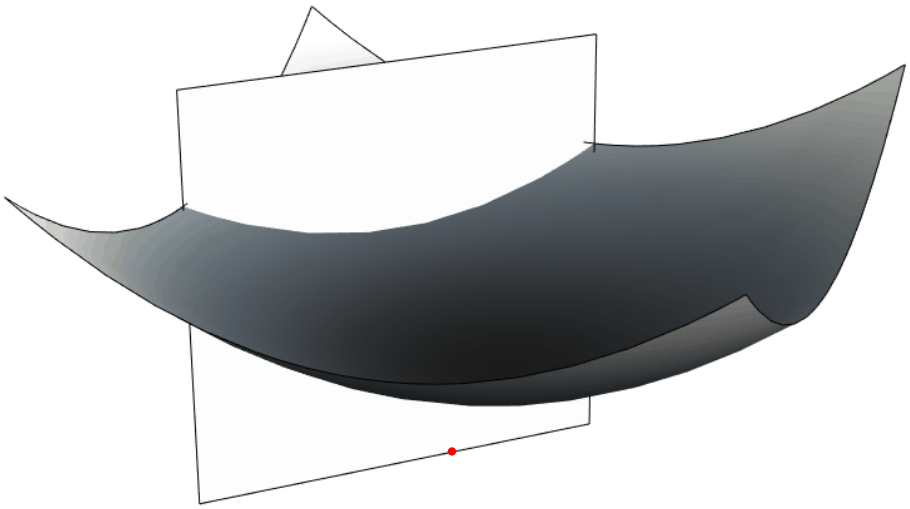
Reducing to one Dimension

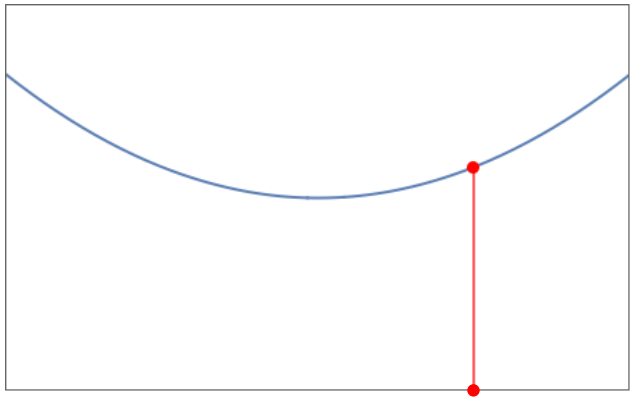




- To make the problem 1-dimensional, fix some vector $v \in \mathbb{R}_2$, and draw a line through x in direction of v.
- Then intersect f with a plane given by this line: In the coordinate system of f, choose the plane that contains the line and is orthogonal to $\mathbb{R}^2$.
- The plane contains the point x.
- Note we can do that even if $d > 2$. We still obtain a plane.

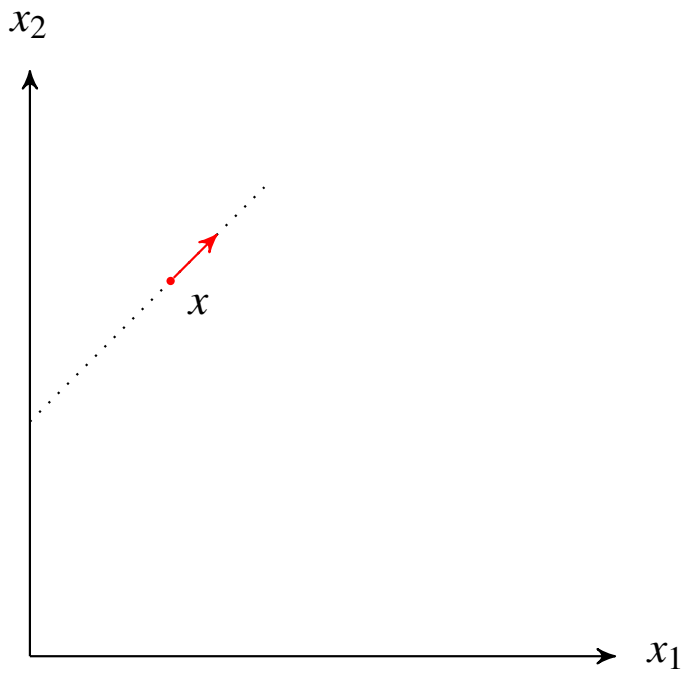
Reducing to one Dimension





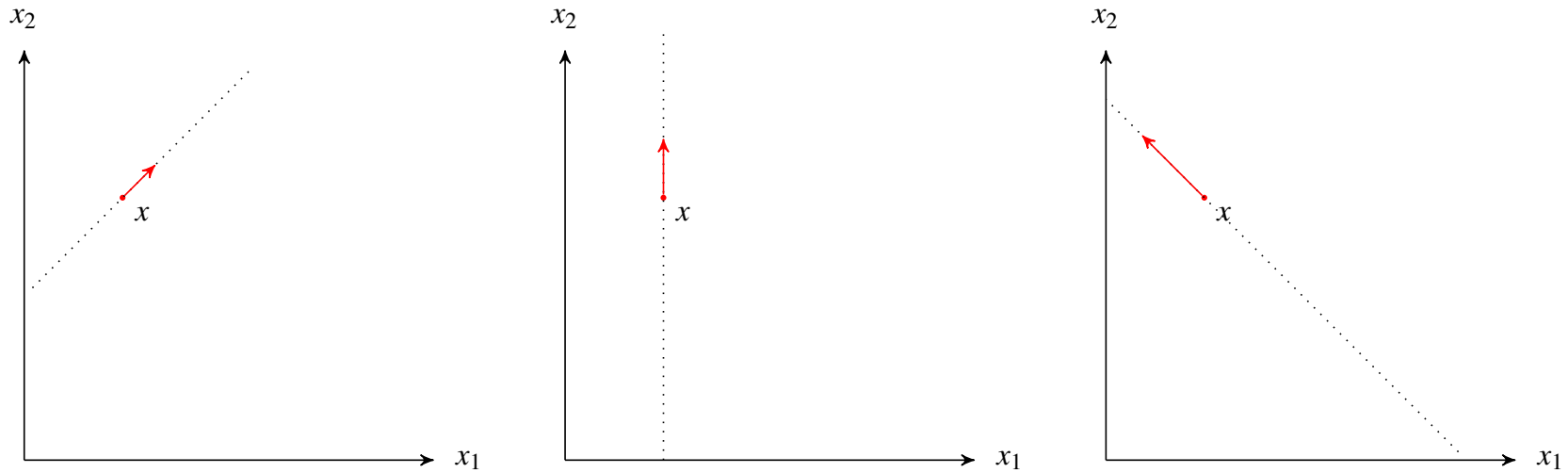
- The intersection of f with the plane is a 1-dimensional function f_H, and x corresponds to a point x_H in its domain.
- We can now compute the derivative f'_H of f_H at x_H. The idea is to use this as the derivative of f at x.

Back to Multiple Dimensions



- In the domain of f, we draw a vector from x *in direction of* H such that:
  1. The vector is oriented to point in the direction in which f_H increases.
  2. Its length is the value of the derivative $f'_H(x)$.
- That completely determines the vector (shown in red above).
- There is one problem still to be solved: f_H depends on H, that is, on the direction of the vector v. Which direction should we use?

The Gradient



- We now rotate the plane H around x. For each position of the plane, we get a new derivative $f'_H(x)$, and a new red vector.
- We choose the plane for which f'_H is largest:

$$H^* := \arg \max_{\text{all rotations of } H} f'_H(x)$$

  Provided that f_H is differentiable for all H, one can show that this is always unique (or $f'_H(x)$ is zero for all H).
- We then define the vector

$$\nabla f(x) := \text{ vector given by } H^* \text{ as above}$$

  The vector $\nabla f(x)$ is called the **gradient** of f at x.

Properties of the Gradient

The gradient $\nabla f(x)$ of $f : \mathbb{R}^d \to \mathbb{R}$ at a point $x \in \mathbb{R}^d$ is a vector in the domain $\mathbb{R}^d$ in the direction in which f most rapidly increases at x.

- The length of the gradient measures steepness: The more rapidly f increases at x, the larger $\|\nabla f(x)\|$.
- The gradient has length 0 if x is a maximum or minimum of f. A constant function has gradient of length 0 at every point x.
- The gradient operation is linear:

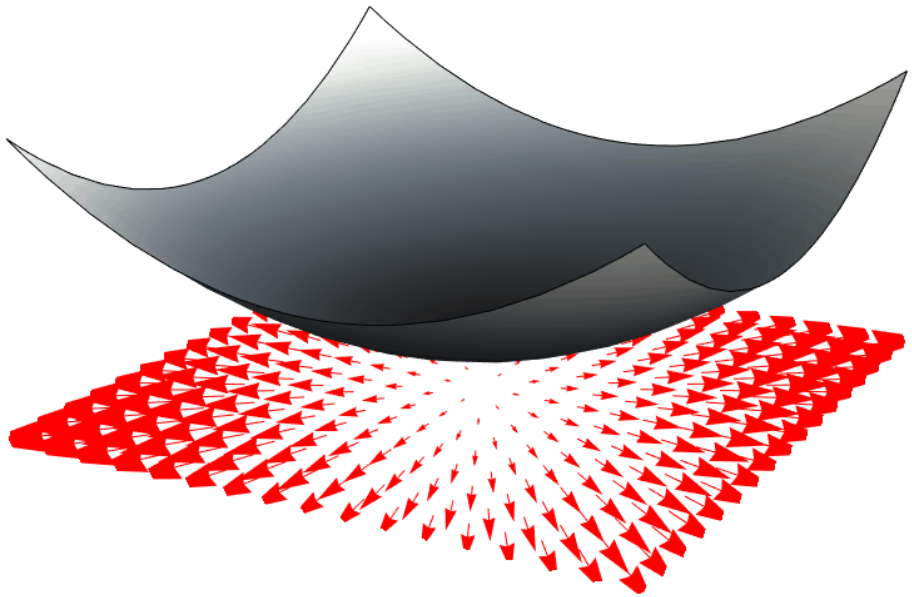
$$\nabla(\alpha f(x) + \beta g(x)) = \alpha \nabla f(x) + \beta \nabla g(x)$$

Gradients and Contour Lines

- Recall that a contour line (or contour set) of f is a set of points along which f remains constant,
$$C[f, c] := \{x \in \mathbb{R}^d \,|\, f(x) = c\} \qquad \text{for some } c \in \mathbb{R}.$$
- One can show that if $C[f, c]$ contains x, the gradient at x is orthogonal to the contour:
$$\nabla f(x) \perp C[f, c] \qquad \text{if } x \in C[f, c]\ .$$
- Intuition: The gradient points in the direction of maximal *local* change, whereas $C[f, c]$ is a direction in which there is no change. Locally, these two are orthogonal.

Gradients are orthogonal to contour lines.

Gradients and Contour Lines



- For this parabolic function, all contour lines are concentric circles around the minimum.
- The picture above shows the gradients plotted at various points in the plane.

$$f : \mathbb{R}^d \to \mathbb{R}$$

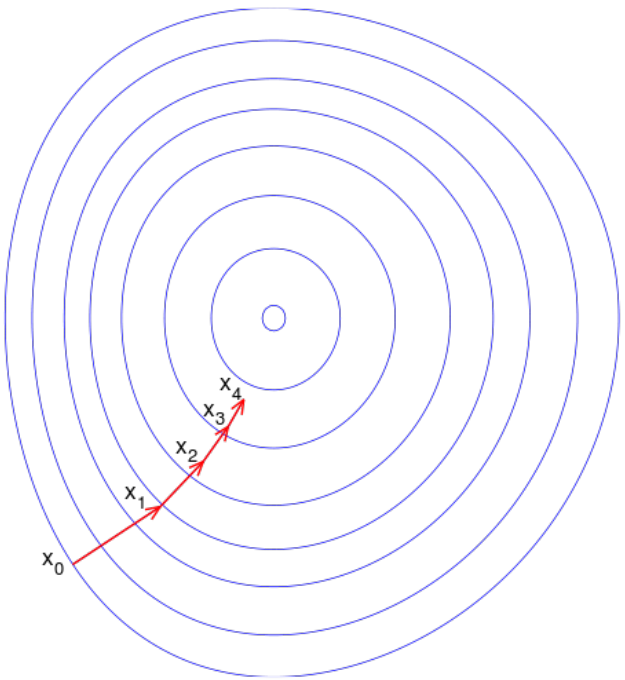
Algorithm

Start with some point $x_0 \in \mathbb{R}$ and fix a precision $\varepsilon > 0$.

Repeat for $n = 1, 2, \ldots$:

1. Check whether $\|\nabla f(x_n)\| < \varepsilon$. If so, report the solution $x^* := x_n$ and terminate.
2. Otherwise, set

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



$$f : \mathbb{R}^d \to \mathbb{R}$$

Algorithm

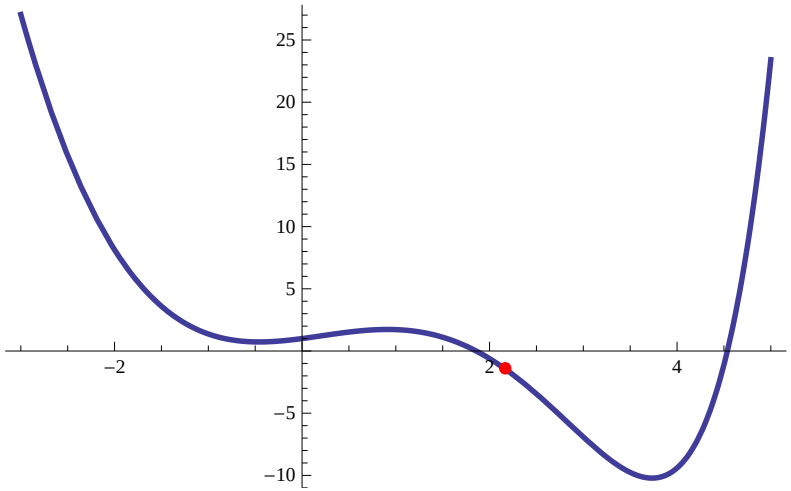
Start with some point $x_0 \in \mathbb{R}$ and fix a precision $\varepsilon > 0$.

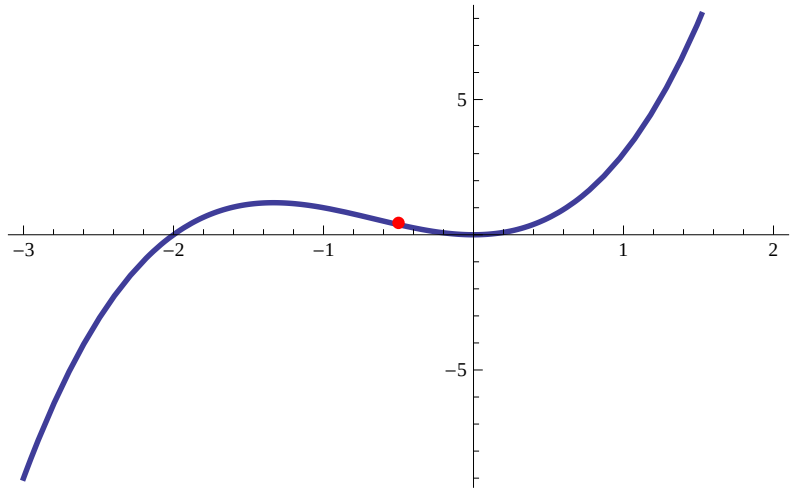
Repeat for $n = 1, 2, \ldots$:

1. Check whether $\|\nabla f(x_n)\| < \varepsilon$. If so, report the solution $x^* := x_n$ and terminate.
2. Otherwise, set
$$x_{n+1} := x_n - \alpha(n)\nabla f(x_n)$$

Here, $\alpha(n) > 0$ is a coefficient that may depend on n. It is called the **step size** in optimization, or the **learning rate** in machine learning.

Gradient Descent and Local Minima





- Suppose for both functions above, gradient descent is started at the point marked red.
- It will "walk downhill" as far as possible, then terminate.
- For the function on the left, the minimum it finds is global. For the function on the right, it is only a local minimum.
- Since the derivative at both minima is 0, gradient descent cannot detect whether they are global or local.

For smooth functions, gradient descent finds *local* minima. If the function is complicated, there may be no way to tell whether the solution is also a global minimum.

Summary so far

- The derivative/gradient provides local information about how a function changes around a point x.
- Optimization algorithms: If we know the gradient at our current location x, we can use this information to make a step in "downhill" direction, and move closer to a (local) minimum.

What we do not know yet

That assumes that we can compute the gradient. There are two possibilities:

- For some functions, we are able to derive $\nabla f(x)$ as a function of x. Gradient descent can evaluate the gradient by evaluating that function.
- Otherwise, we have to estimate $\nabla f(x)$ by evaluating the function f at points close to x.

For now, we will assume that we can compute the gradient as a function.

Recap: Optimization and Learning

Given

- Training data $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \ldots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$.

The data analyst chooses

- A class $\mathcal{H}$ of possible solutions (e.g. $\mathcal{H} =$ all linear classifier).
- A loss function L (e.g. 0-1 loss) that measures how well the solution represents the data.

Learning from training data

- Define the empirical risk of a solution $f \in \mathcal{H}$ on the training data,

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

- Find an optimal solution f^* by minimizing the empirical risk using gradient descent (or another optimization algorithm).

THE PERCEPTRON ALGORITHM

Task: Train a Linear Classifier

We want to determine a linear classifier in $\mathbb{R}^d$ using 0-1 loss.

Recall

- A solution is determined by a normal $\mathbf{v}_{\mathrm{H}} \in \mathbb{R}^d$ and an offset $c > 0$.
- For training data $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \ldots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$, the empirical risk is

$$\hat{R}_n(\mathbf{v}_{\mathrm{H}}, c) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\{\mathrm{sgn}(\langle \mathbf{v}_{\mathrm{H}}, \mathbf{x}_i \rangle - c) \neq y_i\}$$

- The empirical risk minimization approach would choose a classifer given by $(\mathbf{v}_{\mathrm{H}}^*, c^*)$ for which

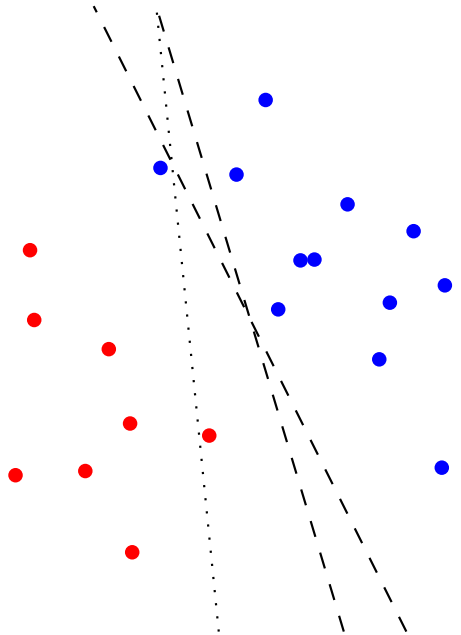
$$(\mathbf{v}_{\mathrm{H}}^*, c^*) = \arg\min_{\mathbf{v}_{\mathrm{H}}, c} \hat{R}_n(\mathbf{v}_{\mathrm{H}}, c)$$

Idea

Can we use gradient descent to find the minimizer of $\hat{R}_n$?

Example

- In the example on the right, the two dashed classifiers both get a single (blue) training point wrong.
- Both of them have different values of $(\mathbf{v}_\mathrm{H}, c)$, but for both of these values, the empirical risk is identical.
- Suppose we shift one of the to dashed lines to obtain the dotted line. On the way, the line moves over a single red point. The moment it passes that point, the empirical risk jumps from $\frac{1}{n}$ to $\frac{2}{n}$.

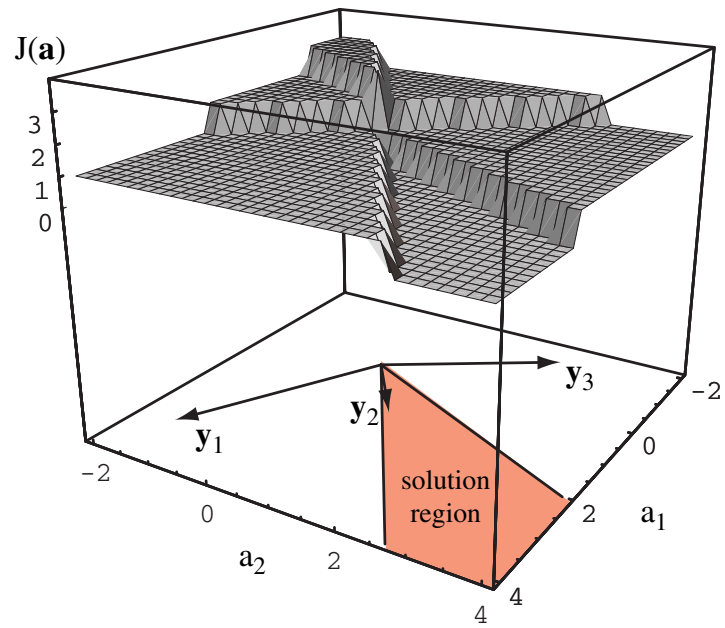


Conclusion

Consider the empirical risk function $\hat{R}_n(\mathbf{v}_\mathrm{H}, c)$:

- If $(\mathbf{v}_\mathrm{H}, c)$ defines an affine plain that contains one of the training points, $\hat{R}_n$ is discontinuous at $(\mathbf{v}_\mathrm{H}, c)$ (it "jumps"). That means it is not differentiable.
- At all other points $(\mathbf{v}_\mathrm{H}, c)$, the function is constant. It is differentiable, but the length of the gradient is 0.

The empirical risk of a linear classifer under 0-1 loss is piece-wise constant.



Formal problem

Even if we can avoid points where $\hat{R}_n$ jumps, the gradient is always 0. Gradient descent never moves anywhere.

Intuition

- Remember that we can only evaluate local information about $\hat{R}_n$ around a given point $(\mathbf{v}_\text{H}, c)$.
- In every direction around $(\mathbf{v}_\text{H}, c)$, the function looks identical.
- The algorithm cannot tell what a good direction to move in would be.
- Note that is also the case for every other optimization algorithm, since optimization algorithms depend on local information.

Solution idea

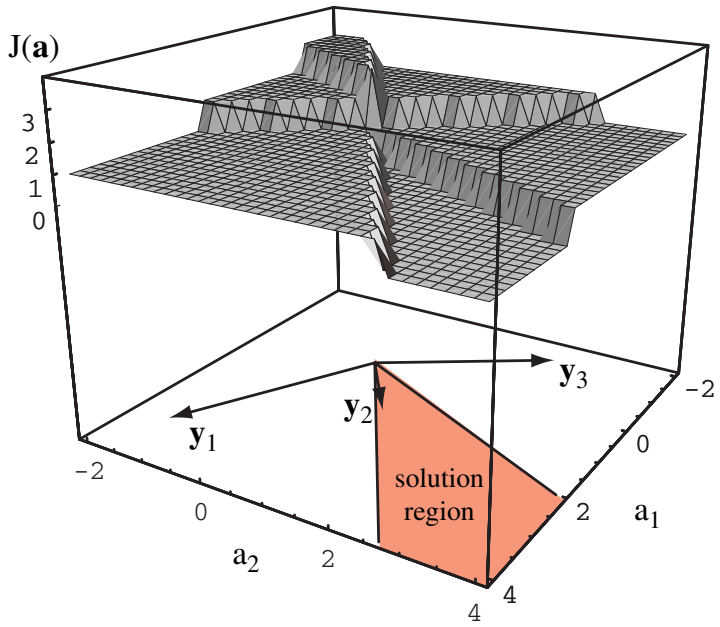
Find an approximation to $\hat{R}_n$ that is not piece-wise constant, and decreases in direction of an optimal solution. We try to keep the approximation as simple as possible.

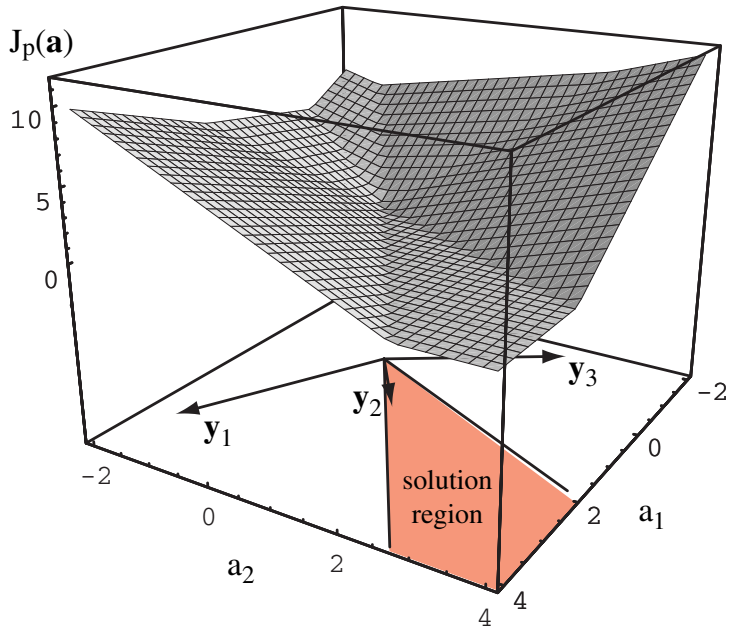
Perceptron Cost Function

We replace the empirical risk $\hat{R}_n(\mathbf{v}_H, c) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\{\text{sgn}(\langle \mathbf{v}_H, \mathbf{x}_i\rangle) - c) \neq y_i\}$ by the piece-wise linear function

$$\hat{S}_n(\mathbf{v}_H, c) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{\mathbb{I}\{\text{sgn}(\langle \mathbf{v}_H, \mathbf{x}_i\rangle) - c) \neq y_i\}}_{\text{“switches off” correctly classified points}} \cdot \overbrace{|\langle \mathbf{v}_H, \mathbf{x}\rangle - c|}^{\text{measures distance to plane}}$$

$\hat{S}_n$ is called the **perceptron cost function**.





Training

- **Given**: Training data $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \ldots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$.
- **Training**: Fix a precision ε and a learning rate α, and run gradient descent on the perceptron cost function to find an affine plane $(\mathbf{v}_{\mathrm{H}}^*, c^*)$
- Define a classifier as $f(\mathbf{x}) := \operatorname{sgn}(\langle \mathbf{v}_{\mathrm{H}}, \mathbf{x} \rangle - c)$.

(If the gradient algorithm returns $c < 0$, flip signs: Use $(-\mathbf{v}_{\mathrm{H}}^*, -c^*)$.)

Prediction

- For a given data point $\mathbf{x} \in \mathbb{R}^d$, predict the class label $y := f(\mathbf{x})$.

This classifier is called the **perceptron**. It was first proposed by Rosenblatt in 1962.

The Perceptron Gradient

One can show that the gradient of the cost function is

$$\nabla \hat{S}_n(\mathbf{v}_\mathrm{H}, c) = -\sum_{i=1}^{n} \mathbb{I}\{f(\tilde{\mathbf{x}}_i) \neq \tilde{y}_i\} \cdot \tilde{y}_i \begin{pmatrix} \tilde{\mathbf{x}}_i \\ 1 \end{pmatrix} \ .$$

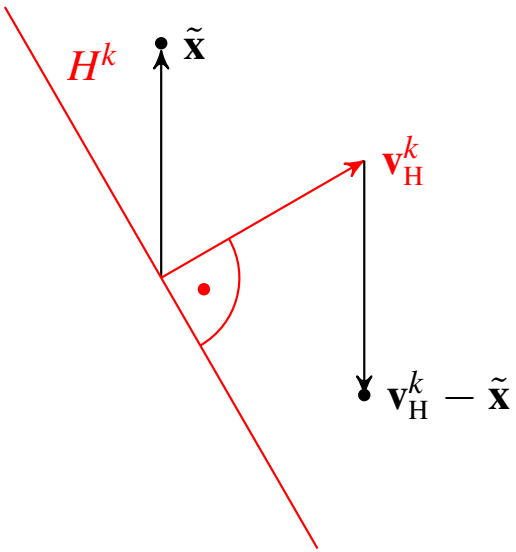
This is an example of gradient descent where we do not have to approximate the derivative numerically. Gradient descent steps then look like this:

$$\begin{pmatrix} \mathbf{v}_\mathrm{H}^{(k+1)} \\ c^{(k+1)} \end{pmatrix} := \begin{pmatrix} \mathbf{v}_\mathrm{H}^{(k)} \\ c^{(k)} \end{pmatrix} + \sum_{i|\tilde{\mathbf{x}}_i \text{ misclassified}} \tilde{y}_i \begin{pmatrix} \tilde{\mathbf{x}}_i \\ 1 \end{pmatrix}$$

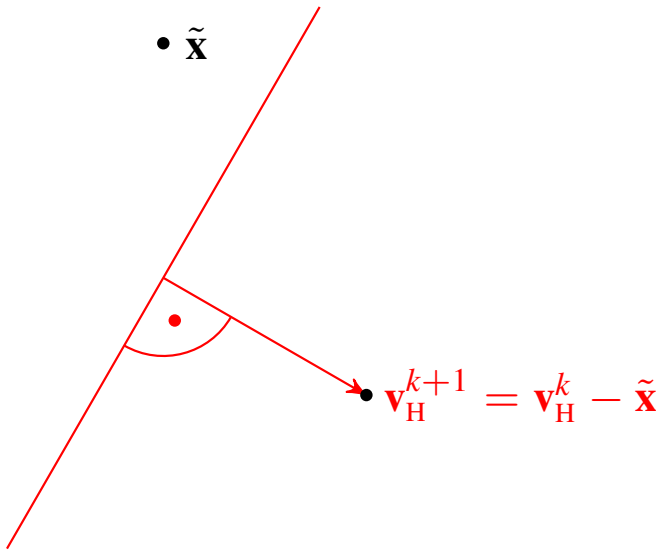
$$\begin{pmatrix} \mathbf{v}_{\mathrm{H}}^{(k+1)} \\ c^{(k+1)} \end{pmatrix} := \begin{pmatrix} \mathbf{v}_{\mathrm{H}}^{(k)} \\ c^{(k)} \end{pmatrix} + \sum_{i|\tilde{\mathbf{x}}_i \text{ misclassified}} \tilde{y}_i \begin{pmatrix} \tilde{\mathbf{x}}_i \\ 1 \end{pmatrix}$$

Effect for a single training point

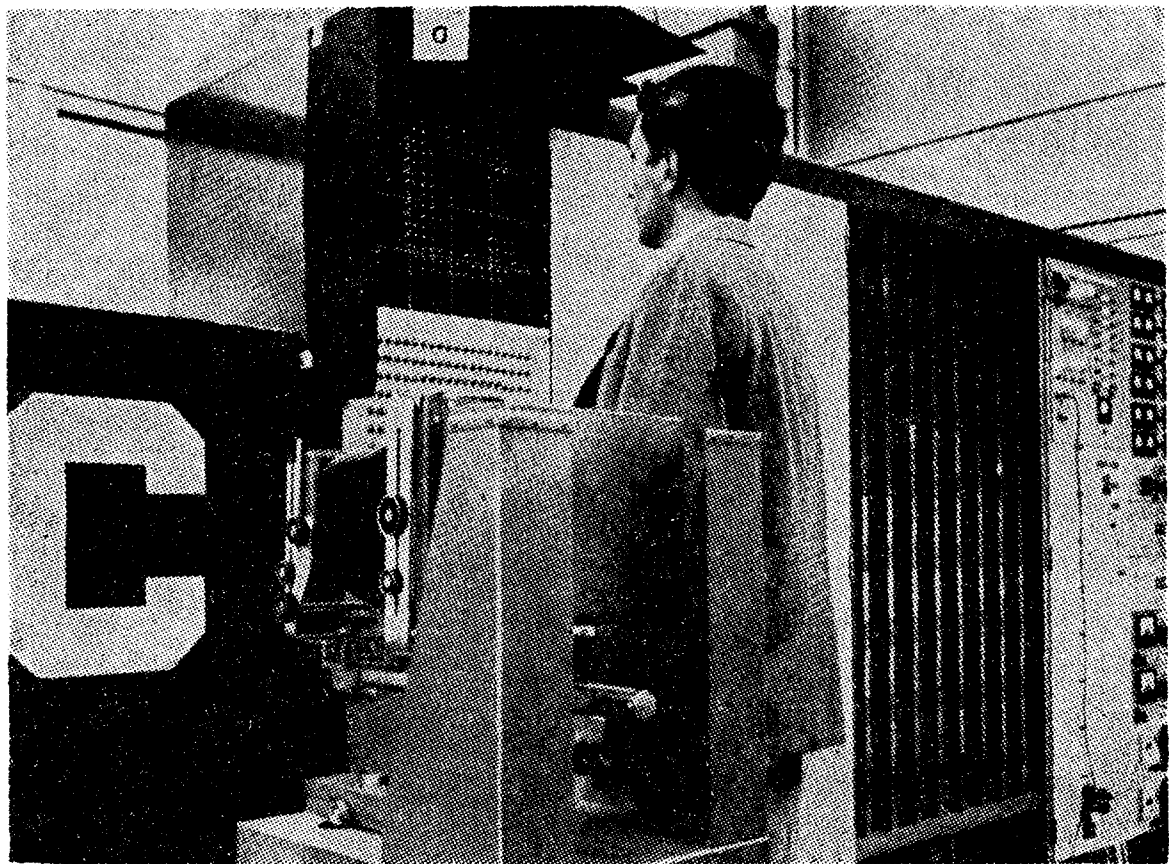
Step k: $\tilde{\mathbf{x}}$ (in class -1) classified incorrectly

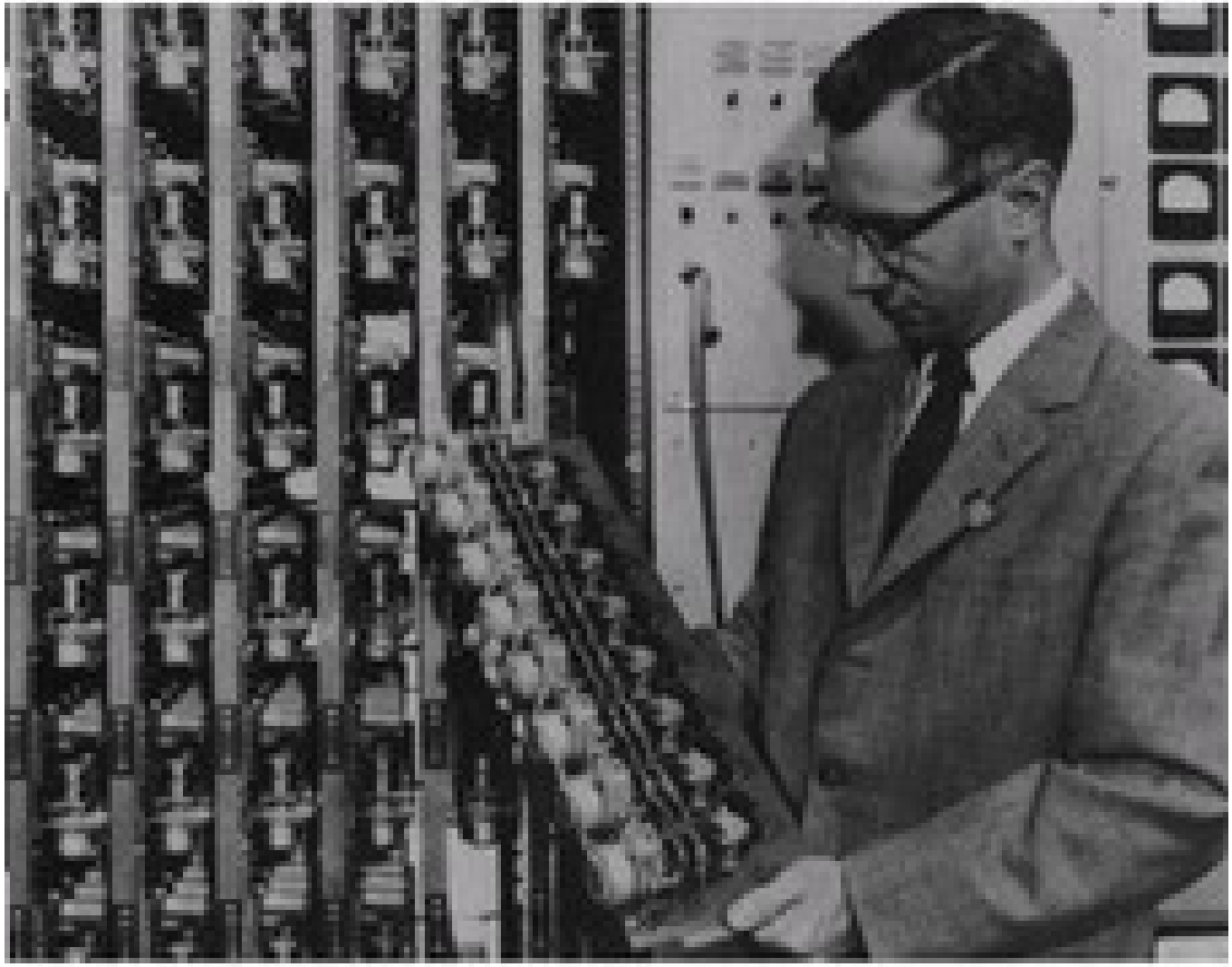


Step $k+1$



Simplifying assumption: H contains origin, so $c = 0$.



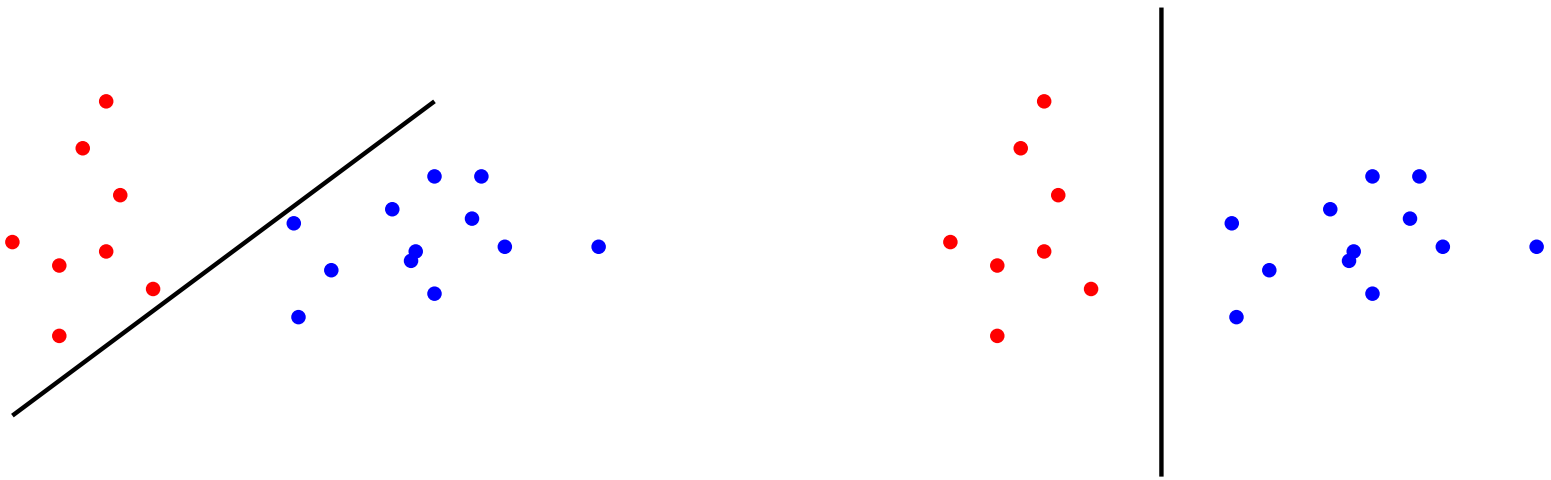


Theorem: Perceptron convergence

If the training data is linearly separable, the Perceptron learning algorithm (with fixed step size) terminates after a finite number of steps with a valid solution $(\mathbf{v}_{\mathrm{H}}, c)$ (i.e. a solution which classifies all training data points correctly).

Issues

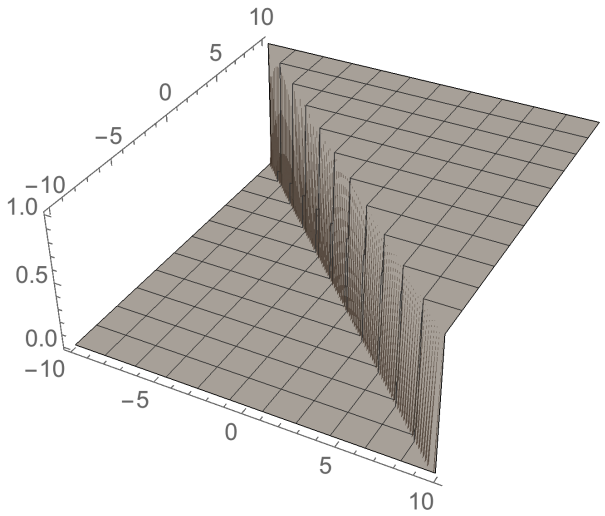
The perceptron selects *some* hyperplane between the two classes. The choice depends on initialization, step size etc.



The solution on the right will probably predict better than the one on the left, but the perceptron may return either.

Logistic Regression

A classifier is a piece-wise constant function, which means it "jumps" at the decision boundary:



- We had already noted that that is inconvenient for optimization: The function is either constant (optimization algorithms cannot extract local information) or not differentiable.
- The function does not distinguish between points close to and far from the boundary. That allows e.g. the perceptron to place the decision boundary very close to data points.

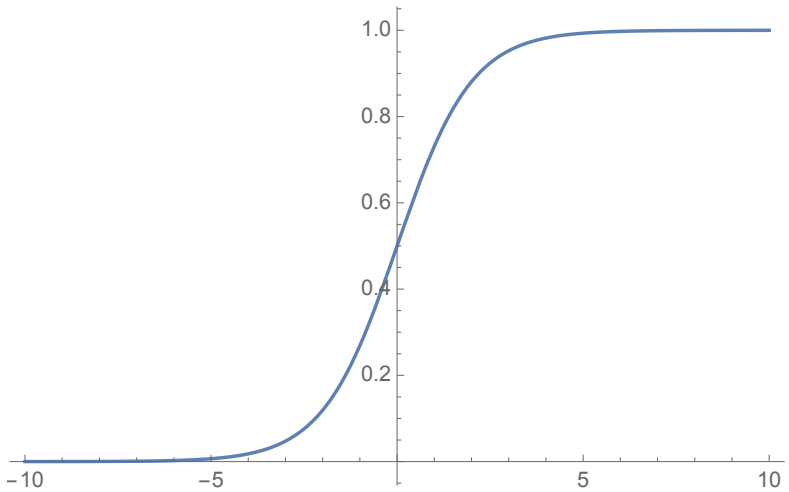
Idea

We replace the piece-wise constant function by a smooth function that otherwise looks similar. There is a canonical way of doing so, called *logistic regression*.

Keep in mind: Logistic regression is a classification method.

Sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

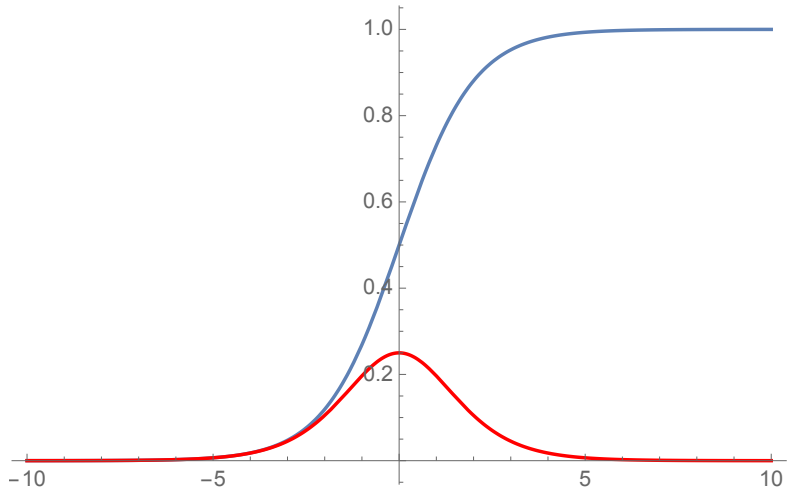


Note

$$1 - \sigma(x) = \frac{1 + e^{-x} - 1}{1 + e^{-x}} = \frac{1}{e^{x} + 1} = \sigma(-x)$$

Derivative

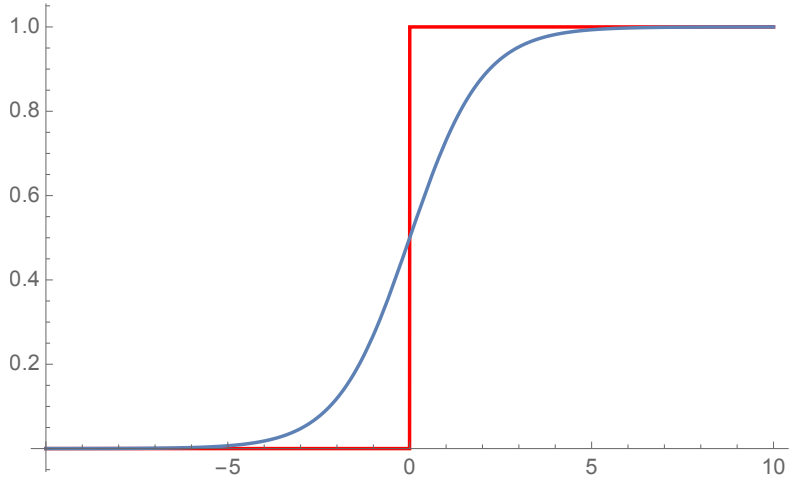
$$\frac{d\sigma}{dx}(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = \sigma(x)\big(1 - \sigma(x)\big)$$



Sigmoid (blue) and its derivative (red)

APPROXIMATING DECISION BOUNDARIES

- In linear classification: Decision boundary is a discontinuity
- Boundary is represented either by indicator function $\mathbb{I}\{\bullet > c\}$ or sign function $\text{sign}(\bullet - c)$
- These representations are equivalent: Note $\text{sign}(\bullet - c) = 2 \cdot \mathbb{I}\{\bullet > c\} - 1$



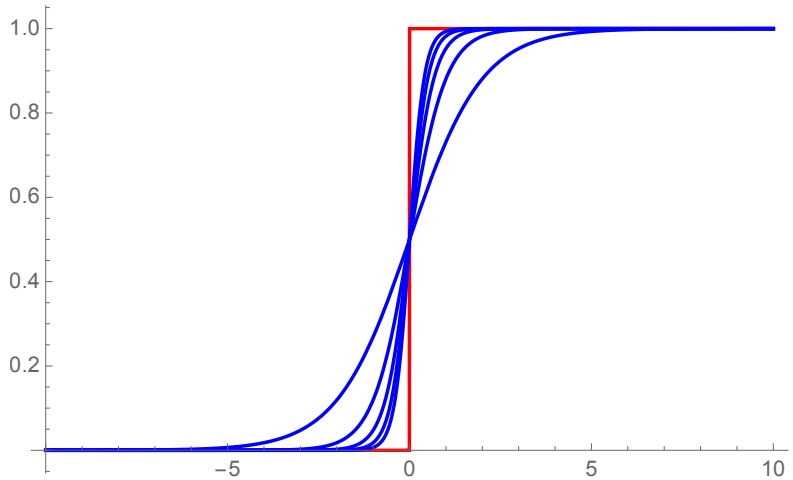
The most important use of the sigmoid function in machine learning is *as a smooth approximation to the indicator function.*

Given a sigmoid σ and a data point x, we decide which side of the approximated boundary we are own by thresholding

$$\sigma(x) \geq \frac{1}{2}$$

We can add a scale parameter by definining

$$\sigma_\theta(x) := \sigma(\theta x) = \frac{1}{1 - e^{-\theta x}} \qquad \text{for } \theta \in \mathbb{R}$$



Influence of θ

- As θ increases, σ_θ approximates $\mathbb{I}$ more closely.
- For $\theta \to \infty$, the sigmoid converges to $\mathbb{I}$ pointwise, that is: For every $x \neq 0$, we have

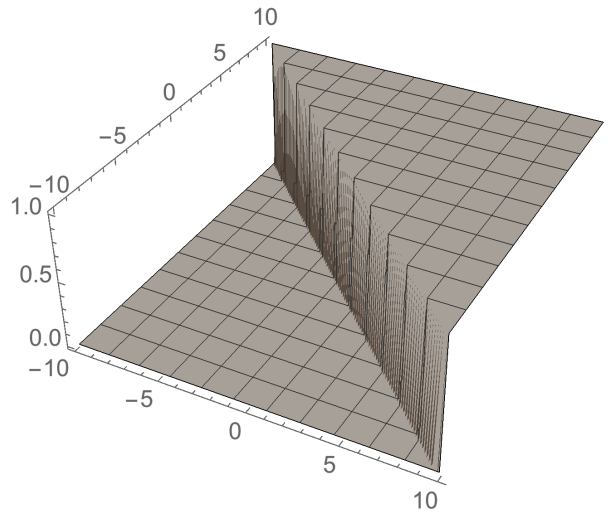
$$\sigma_\theta(x) \to \mathbb{I}\{x > 0\} \qquad \text{as } \theta \to +\infty \ .$$

- Note $\sigma_\theta(0) = \frac{1}{2}$ always, regardless of θ.

Approximating a Linear Classifier

So far, we have considered $\mathbb{R}$, but linear classifiers usually live in $\mathbb{R}^d$.

The decision boundary of a linear classifier in $\mathbb{R}^2$ is a discontinuous ridge:

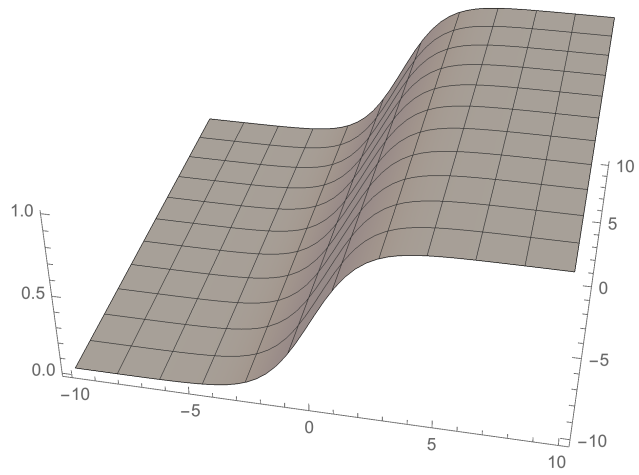


- This is a linear classifier of the form

$$\mathbb{I}\{\langle \mathbf{v}, \mathbf{x} \rangle - c\}.$$

- Here: $\mathbf{v} = (1, 1)$ and $c = 0$.

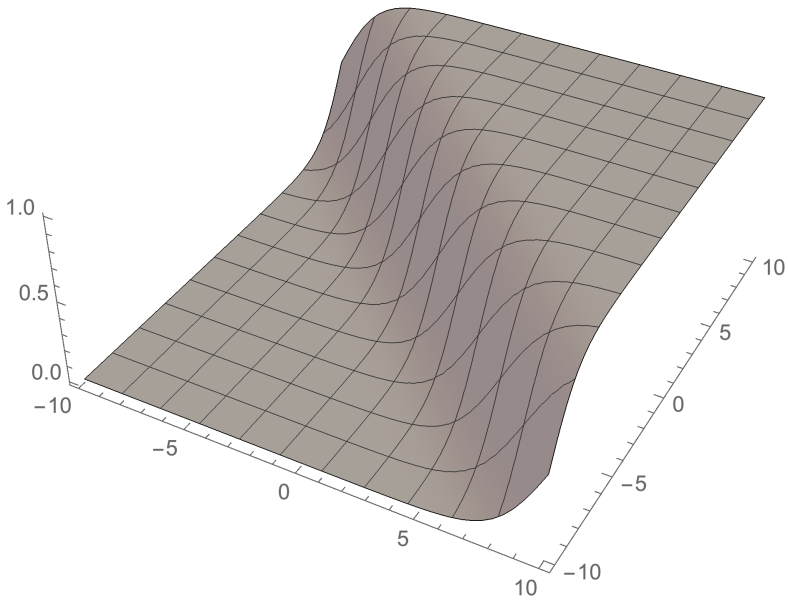
We can "stretch" σ into a ridge function on $\mathbb{R}^2$:

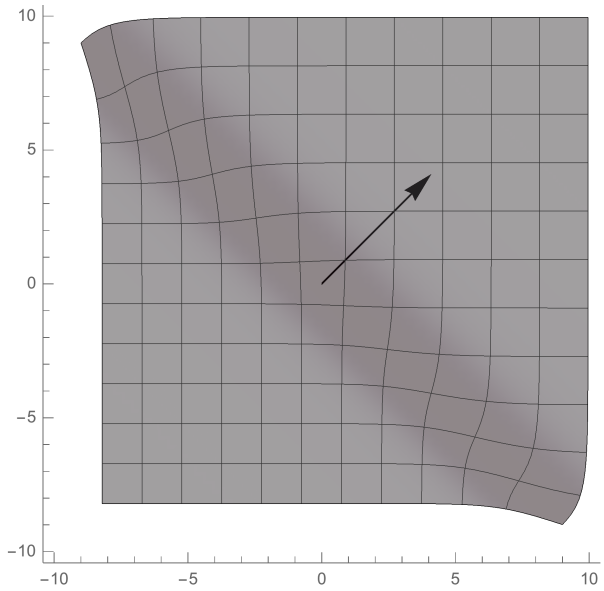


- This is the function $\mathbf{x} = (x_1, x_2) \mapsto \sigma(x_1)$.
- The ridge runs parallel to the x_2-axes.
- If we use $\sigma(x_2)$ instead, we rotate by 90 degrees (still axis-parallel).

Steering a Sigmoid

Just as for a linear classifier, we use a normal vector $\mathbf{v} \in \mathbb{R}^d$.





- The function $\sigma(\langle \mathbf{v}, \mathbf{x} \rangle - c)$ is a sigmoid ridge, where the ridge is orthogonal to the normal vector $\mathbf{v}$, and c is an offset that shifts the ridge "out of the origin".
- The plot on the right shows the normal vector (here: $\mathbf{v} = (1, 1)$) in black.
- The parameters $\mathbf{v}$ and c have the same meaning for $\mathbb{I}$ and σ, that is, $\sigma(\langle \mathbf{v}, \mathbf{x} \rangle - c)$ approximates $\mathbb{I}\{\langle \mathbf{v}, \mathbf{x} \rangle \geq c\}$.

Logistic regression is a classification method that approximates decision boundaries by sigmoids.

Setup

- Two-class classification problem
- Observations $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, class labels $y_i \in \{0, 1\}$.

The logistic regression model

We model the conditional distribution of the class label given the data as

$$P(y|\mathbf{x}) := \text{Bernoulli}\big(\sigma(\langle \mathbf{v}, \mathbf{x} \rangle - c)\big) \ .$$

- Recall $\sigma(\langle \mathbf{v}, \mathbf{x} \rangle - c)$ takes values in $[0, 1]$ for all θ, and value $\frac{1}{2}$ on the class boundary.
- The logistic regression model interprets this value as the probability of being in class y.

Since the model is defined by a parametric distribution, we can apply maximum likelihood.

Likelihood function of the logistic regression model

$$\prod_{i=1}^{n} \sigma(\langle \mathbf{v}, \tilde{\mathbf{x}}_i \rangle - c)^{y_i} \left(1 - (\sigma(\langle \mathbf{v}, \tilde{\mathbf{x}}_i \rangle - c))\right)^{1-y_i}$$

Negative log-likelihood

$$L(\mathbf{w}) \quad := \quad -\sum_{i=1}^{n} \Big(y_i \log \sigma(\langle \mathbf{v}, \tilde{\mathbf{x}}_i \rangle - c) + (1 - y_i) \log \big(1 - \sigma((\langle \mathbf{v}, \tilde{\mathbf{x}}_i \rangle - c))\big) \Big)$$

$$\nabla L(\mathbf{v}, c) \quad = \quad \sum_{i=1}^{n} \big(\sigma(\langle \mathbf{v}, \tilde{\mathbf{x}}_i \rangle - c) - y_i\big) \begin{pmatrix} \tilde{\mathbf{x}}_i \\ 1 \end{pmatrix}$$

Note

- Each training data point $\mathbf{x}_i$ contributes to the sum proportionally to the approximation error $\sigma(\langle \mathbf{v}, \tilde{\mathbf{x}}_i \rangle - c) - y_i$ incurred at $\mathbf{x}_i$ by approximating the linear classifier by a sigmoid.

Learning logistic regression

To learn a logistic regression classifier from training data, we minimize $L(\mathbf{v}, c)$ using gradient descent or another optimization algorithm.

- The function L is convex (= $\cup$-shaped). That means there is only a single local minimum, which is also the global minimum.
- FYI: You may encounter an algorithm called *iteratively reweighted least squares* for training logistic regression in the literature. The algorithm is obtained by applying a more sophisticated version of gradient descent (called *Newton's method*) to minimize L.

Bernoulli and multinomial distributions

- The mulitnomial distribution of N draws from K categories with parameter vector $(\theta_1, \ldots, \theta_K)$ (where $\sum_{k \leq K} \theta_k = 1$) has probabililty mass function

$$P(m_1, \ldots, m_K | \theta_1, \ldots, \theta_K) = \frac{N!}{m_1! \cdots m_K!} \prod_{k=1}^{K} \theta_k^{m_k} \qquad \text{where } m_k = \#\text{ draws in category } k$$

- Note that $\text{Bernoulli}(p) = \text{Multinomial}(p, 1-p; N = 1)$.

Logistic regression

- Recall two-class logistic regression is defined by $P(Y|\mathbf{x}) = \text{Bernoulli}(\sigma(\langle \mathbf{v}, \mathbf{x} \rangle - c))$.
- Idea: To generalize logistic regression to K classes, choose a separate weight vector $\mathbf{v}_k$ and offset c_k for each class k, and define $P(Y|\mathbf{x})$ by

$$\text{Multinomial}\big(\tilde{\sigma}(\langle \mathbf{v}_1, \mathbf{x} \rangle - c_1), \ldots, \tilde{\sigma}(\langle \mathbf{v}_K, \mathbf{x} \rangle - c_K)\big)$$

where $\tilde{\sigma}(\langle \mathbf{v}_k, \mathbf{x} \rangle - c_k) = \frac{\sigma(\langle \mathbf{v}_k, \mathbf{x} \rangle - c_k)}{\sum_{j=1}^{K} \sigma(\langle \mathbf{v}_j, \mathbf{x} \rangle - c_j)}$. This definition ensures the $\tilde{\sigma}$-values add up to 1 over all classes.

Not examinable.

Logistic regression for K classes

The label y now takes values in $\{1, \ldots, K\}$.

$$P(y|\mathbf{x}) = \prod_{k=1}^{K} \tilde{\sigma}(\langle \mathbf{v}_k, \mathbf{x} \rangle - c_k)^{\mathbb{I}\{y=k\}}$$

The negative log-likelihood becomes

$$L(\mathbf{v}_1, c_1, \ldots, \mathbf{v}_K, c_K) \quad = \quad - \sum_{i \leq n,\, k \leq K} \mathbb{I}\{y = k\} \log \tilde{\sigma}(\langle \mathbf{v}_k, \tilde{\mathbf{x}}_i \rangle - c_k)$$

This can again be optimized numerically.

Comparison to two-class case

- Recall that $1 - \sigma(x) = \sigma(-x)$, and
  $\text{Bernoulli}(p) = \text{Multinomial}(p, 1-p)$ (with $N = 1$ draws)
- That means

$$\text{Bernoulli}\big(\sigma(\langle \mathbf{v}, \mathbf{x} \rangle - c)\big) \equiv \text{Multinomial}\big(\sigma(\langle \mathbf{v}, \mathbf{x} \rangle - c, \sigma(\langle -\mathbf{v}, \mathbf{x} \rangle + c)\big)$$

- That is: Two-class logistic regression as above is equivalent to multiclass logistic regression with $K = 2$ *provided* we choose $\mathbf{w}_2 = -\mathbf{w}_1$.

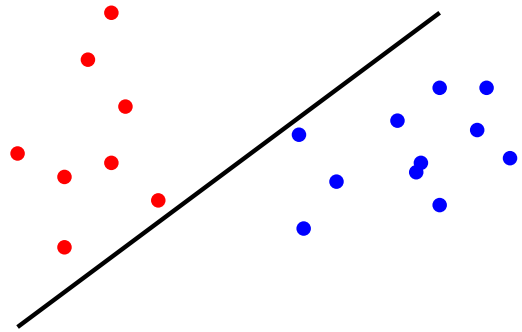
Not examinable.

MAXIMUM MARGIN CLASSIFIERS

Maximum margin idea

Setting

Linear classification, two linearly separable classes.



Recall Perceptron

- Selects *some* hyperplane between the two classes.
- Choice depends on initialization, step size etc.

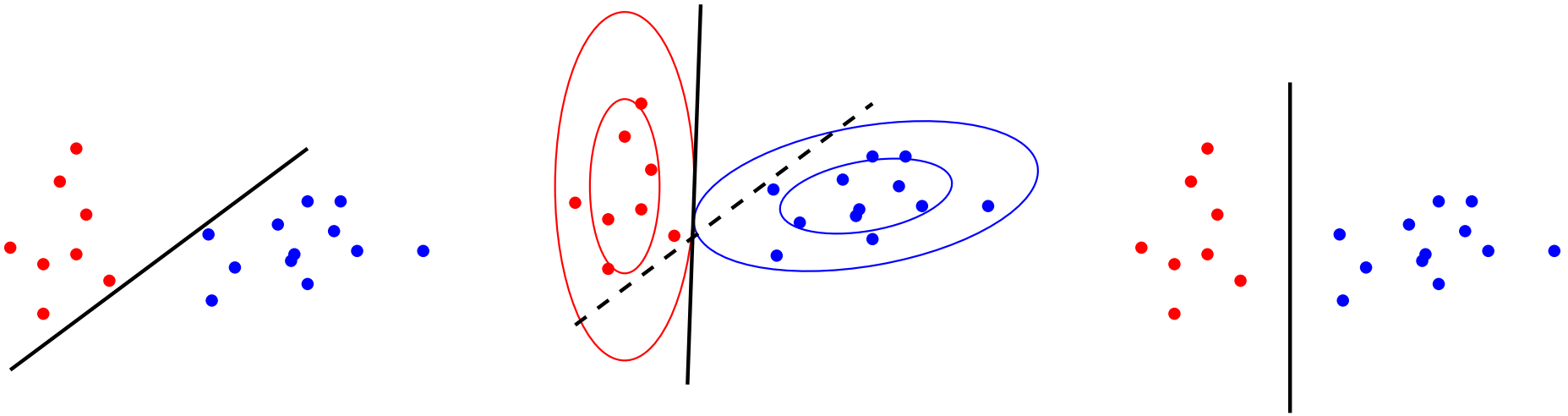
Maximum margin idea

To achieve good generalization (low prediction error), place the hyperplane "in the middle" between the two classes.

More precisely

Choose plane such that distance to closest point in each class is maximal. This distance is called the *margin*.

GENERALIZATION ERROR



Possible Perceptron solution

Good generalization under a specific distribution (here: Gaussian)

Maximum margin solution

Example: Gaussian data

- The ellipses represent lines of constant standard deviation (1 and 2 STD respectively).
- The 1 STD ellipse contains $\sim 68.3\%$ of the probability mass ($\sim 95.5\%$ for 2 STD; $\sim 99.7\%$ for 3 STD).

Optimal generalization: Classifier should cut off as little probability mass as possible from either distribution.

Without distributional assumption: Max-margin classifier

- Philosophy: Without distribution assumptions, best guess is symmetric.
- In the Gaussian example, the max-margin solution would *not* be optimal.

Next: Two Tools

Convex sets

- There is an inherent relationship between linear classification and a geometric property of shapes called *convexity*.
- Convex shapes have very useful properties, and we can use those for classification.

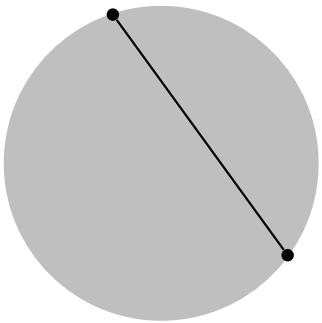
Constrained optimization

- The optimization problems we have considered before asked: What is the value of x for which $f(x)$ is as small as possible?
- A *constrained* optimization problem asks: Among all x which satisfy the property, which value makes $f(x)$ as small as possible?
- We use that to formulate the maximum margin problem as: Among all classifiers that separate the two classes, which one makes the margin as large as possible?

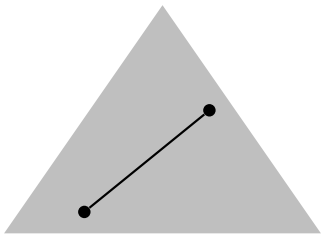
Definition

A set $A \subset \mathbb{R}^d$ is called **convex** if, for every two points $\mathbf{x}, \mathbf{y} \in A$, the straight line connecting $\mathbf{x}$ and $\mathbf{y}$ is completely contained in A.

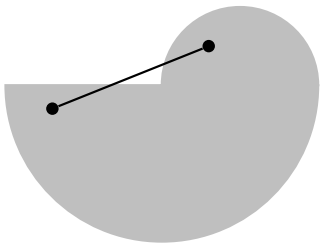
Examples



convex



convex



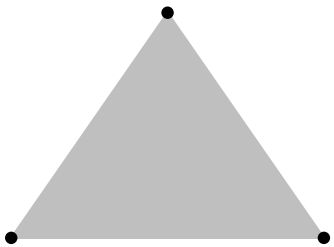
not convex

Extreme points

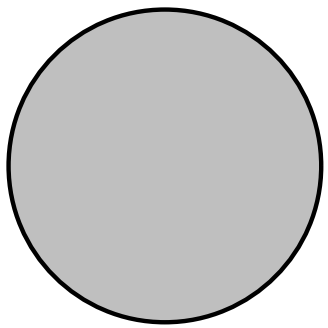
Let A be a convex set and $x \in A$. If x can be removed from A and $A \setminus \{x\}$ is still convex, then x is called an **extreme point** of A.

Examples

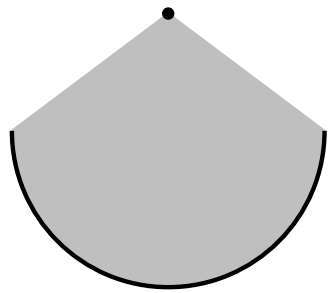
Extreme points are marked black.



finitely many extreme points



infinitely many extreme points



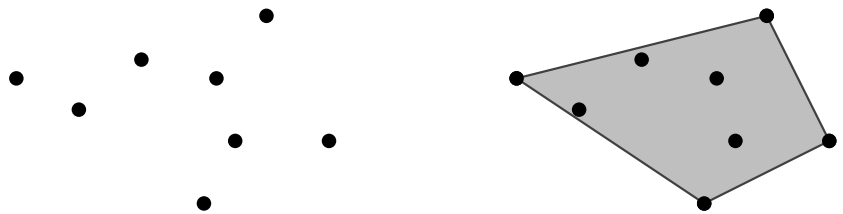
removing a point from the straight part of the boundary would leave a “hole”, and the set would not be convex anymore.

Informally

- If all segments of the boundary are straight lines or planes, the extreme points are exactly the “corner points” of the set.

Definition

If C is a finite set of points in $\mathbb{R}^d$, the **convex hull** $\text{conv}(C)$ of C is the smallest convex set that contains all points in C.

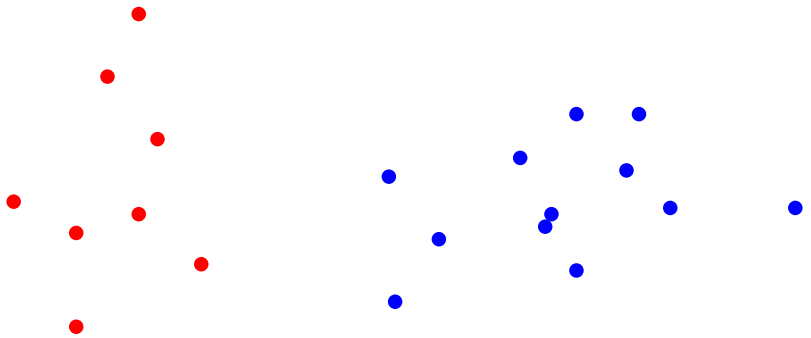


Note

- Each extreme point of $\text{conv}(C)$ is a point in the original set C.
- The convex hull is uniquely determined by C. (Every other convex in $\mathbb{R}^d$ either contains $\text{conv}(C)$, or does not contain all points in C.)
- Think of the convex hull as the shape we get by connecting the "outer" point of C.
- The importance of the convex hull for classification is that it defines which points in each training class are "outer" points (namely those which are extreme points of the convex hull).

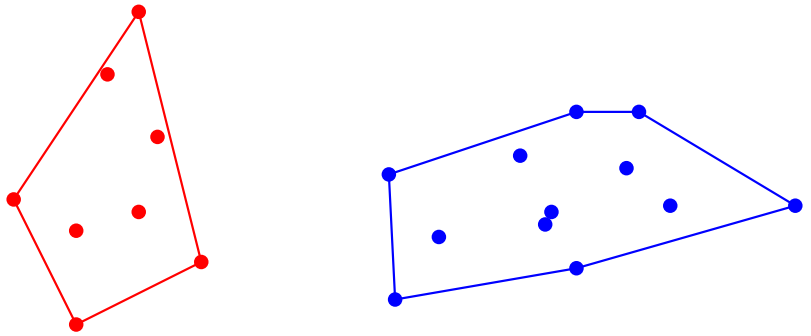
Observation

Where a separating affine plane may be placed depends on the "outer" points of the sets. Points in the center do not matter.



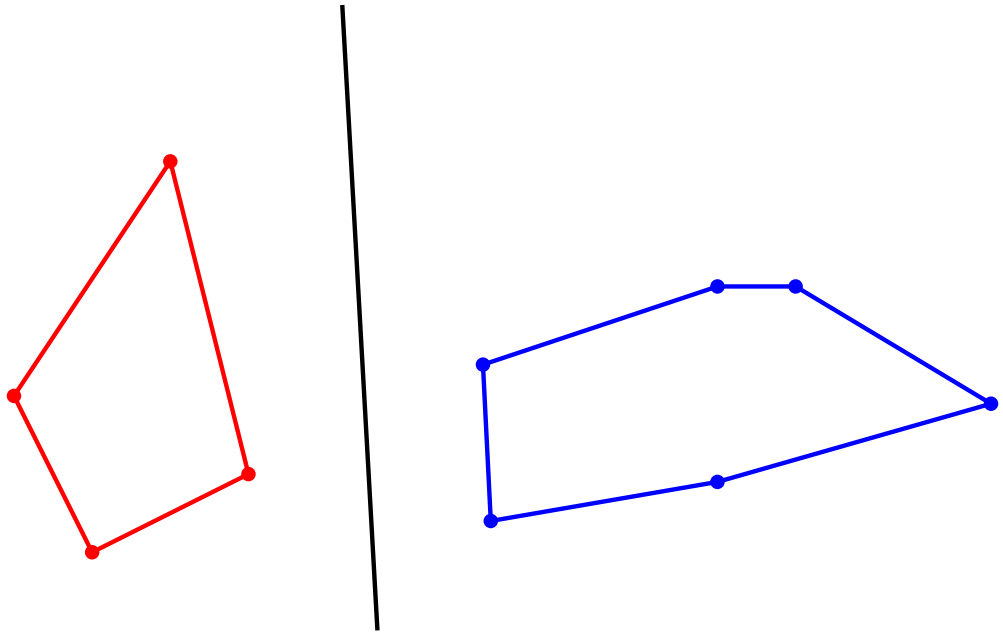
In geometric terms

Substitute each class by its convex hull:



Key idea

There is an inherent relationship between convexity and linear classification: An affine plane separates two classes if and only if it separates their convex hulls.



Next

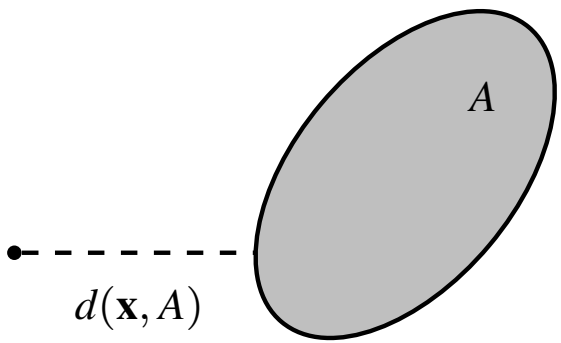
We have to formalize what it means for a hyperplane to be "in the middle" between two classes.

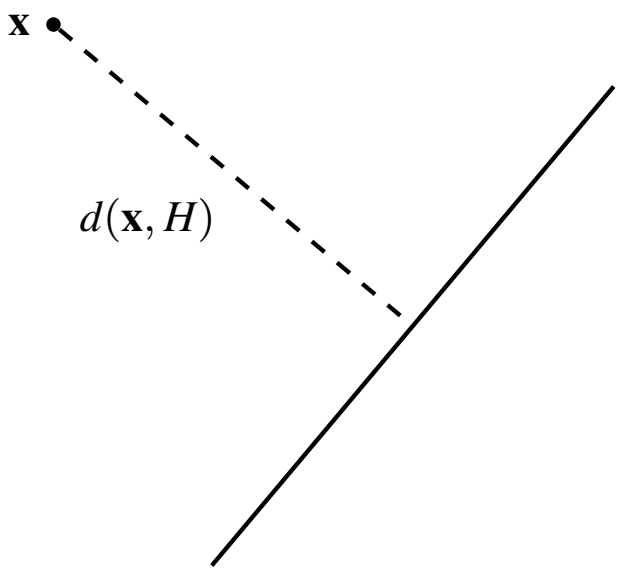
Definition

The **distance** between a point $\mathbf{x}$ and a set A the Euclidean distance between x and the closest point in A:

$$d(\mathbf{x}, A) := \min_{\mathbf{y} \in A} \|\mathbf{x} - \mathbf{y}\|$$

In particular, if $A = H$ is a hyperplane, $d(\mathbf{x}, H) := \min_{\mathbf{y} \in H} \|\mathbf{x} - \mathbf{y}\|$.



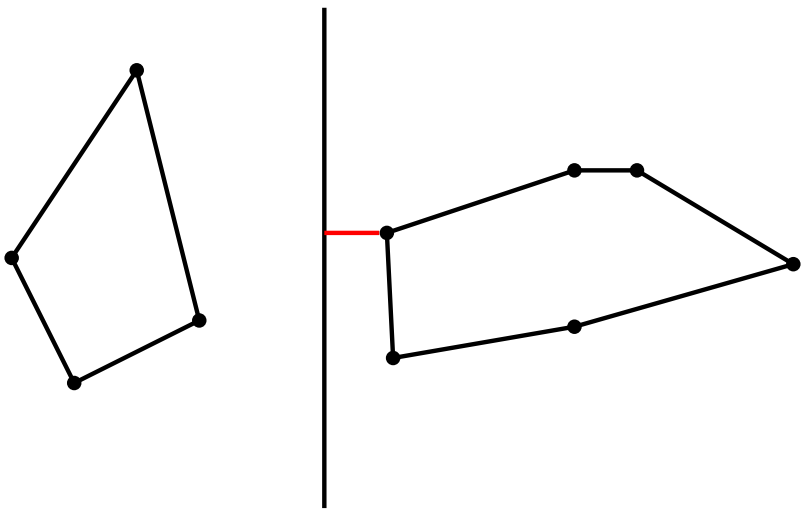


MARGIN

The **margin** of a classifier hyperplane H given two training classes is the shortest distance between the plane and any point in either set:

$$\text{margin} = \min_{x \in \text{ training data}} d(x, H)$$

Equivalently: The shortest distance to either of the convex hulls.



Idea in the following: H is "in the middle" when margin maximal.

Basic problem

We want to find the affine plane $H(\mathbf{v}, c)$ that maximizes the distance to both data sets:

$$\text{maximize } d(H(\mathbf{v}, c), \text{training data}) \text{ over all } \mathbf{v}, c$$

Problem: The optimization algorithm can just move the plane further and further away from the data. We have to make sure H is "between the classes".

Maximum margin optimization problem

The problem we actually solve is

$$\begin{aligned} &\text{maximize } d(H(\mathbf{v}, c), \text{training data}) \text{ over all } \mathbf{v}, c \\ &\text{such that } H(\mathbf{v}, c) \text{ separates the training data classes} \end{aligned}$$

We can express that as:

$$\begin{aligned} &\text{maximize } d(H(\mathbf{v}, c), \text{training data}) \text{ over all } \mathbf{v}, c \\ &\text{such that } \tilde{y}_i \text{sgn}(\langle \mathbf{v}, \tilde{\mathbf{x}}_i \rangle - c) > 0 \end{aligned}$$

This is an example of a so-called *constrained optimization problem*.

Recall that a basic optimization problem searches for an argmument x^* that makes $f(x^*)$ as small as possible.

Contstrains

Suppose we fix some property of x that is either true or false (e.g. "$x > 0$"). The problem

> among all x that satisfy the property, find the one that makes f as small as possible

is called a **constrained optimization problem**. The property is called the **constrained**.

Customary notation

If we call the property A, say, this is often written as:

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & x \text{ satisifies } A \end{aligned}$$

Idea

- An optimization algorithm tries to make f as small as possible.
- We have exclude values of x that violate the constraint.
- Solution: Change the function f so that it is very large at values x that should be excluded.

Implementation

- Choose a function $\beta(x)$ that is very large for all x that violate the constraint, and 0 at those x that are permitted.
- Add β to f: Minimize $f + \beta$ instead of f.
- Remember: We should not introduce jumps, so g should transition smoothly from 0 to "very large".

For example

Say we want to minimize f. For another function g, we impose the constraint $g(x) < 0$.

$$\min f(x) \qquad \text{s.t.} \quad g(x) < 0$$

The constraint $g(x) < 0$ be expressed as an indicator function of $g(x) \geq 0$:

$$\min f(x) + const. \cdot \mathbb{I}_{[0,\infty)}(g(x))$$

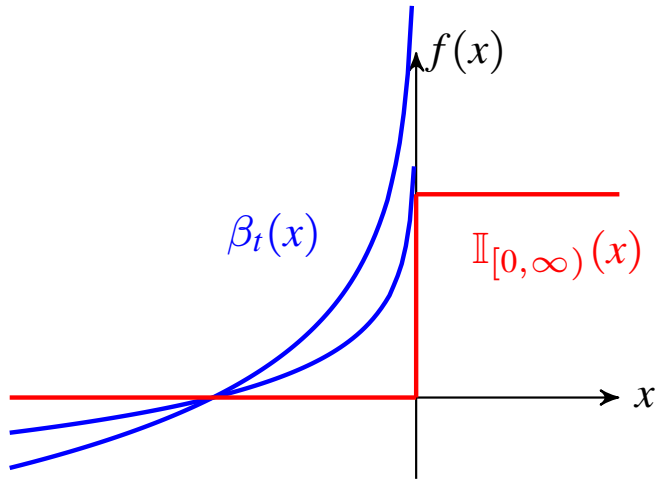
The constant must be chosen large enough to enforce the constraint.

Adding a Smooth Function

Choice of the function we add

- The indicator function jumps, which we know is not useful for optimization. We replace it by a smooth function.
- A common choice is

$$\beta_t(x) := -\frac{1}{t}\log(-x) \ .$$



In the example above

To solve $\min f(x)$ subject to $g(x) < 0$, we apply gradient descent to

$$f(x) + \beta_t(g(x)) \ .$$

The value t is a "tuning parameter" of the optimization method.

Remarks

- If the constraint changes (e.g. to "$g(x) > -3$"), it is easier to modify g than to tinker with β.
- The method above is an example of a principle we have seen before: We express what we want to do in terms of an indicator function, then replace it by something smooth, and apply graident descent.
- Data mining, statistics and machine learning are only a few examples of applications of constrained optimization methods. Much of the research on constrained optimization is driven by operations research and financial engineering.

Maximum margin optimization problem

For n training points $(\tilde{\mathbf{x}}_i, \tilde{y}_i)$ with labels $\tilde{y}_i \in \{-1, 1\}$, solve optimization problem:

$$\begin{aligned} \text{maximize} \quad & d\big(H(\mathbf{v}, c), \{\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n\}\big) \\ \text{s.t.} \quad & \tilde{y}_i(\langle \mathbf{v}_\mathrm{H}, \tilde{\mathbf{x}}_i \rangle - c) > 0 \quad \text{for } i = 1, \ldots, n \end{aligned}$$

- The first line says: Make sure the plane is a far away from every data point as possible.
- The second line says: Only planes that classify the training data correctly are permitted.

Remarks

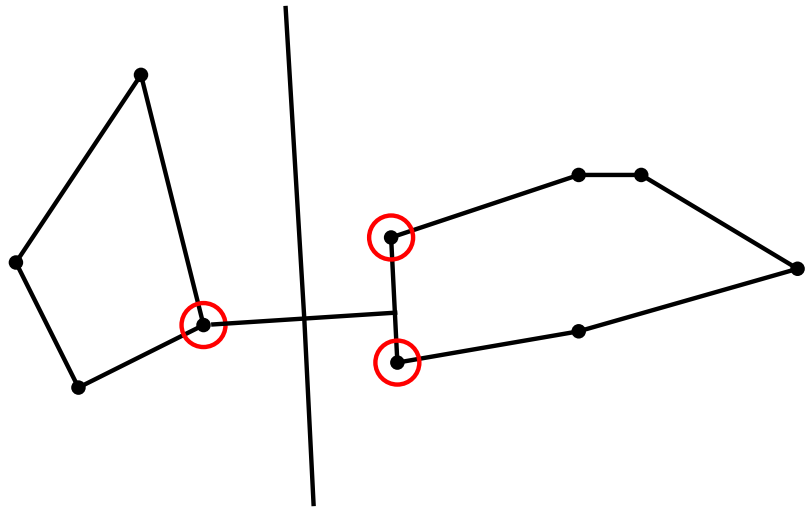
- The classifier obtained by solving this optimization problem is called a **support vector machine**.
- If training data is separable: There is a *unique* solution (in contrast to the perceptron, whose solution is not unique).

Support Vectors

Definition

Those extreme points of the convex hulls which are closest to the hyperplane are called the **support vectors**.

There are at least two support vectors, one in each class.



Implications

- The maximum-margin criterion focuses all attention to the area closest to the decision surface.
- One can show that the computational cost of solving the optimization problem grows quadratically in the number of data points.

Advantages

- The SVM often works well for high-dimensional classification.
- It can be generalized to non-linear decision boundaries using a method called the *kernel trick*.
- It can also be generalized to overlapping classes.

Disadvantages

- The quadratic training cost means SVMs cannot be trained on very large data sets.

The support vector machine (with kernel trick) is, aside from a method called a *random forest*, probably the most widely used classifier for non-vision/audio data. For vision and audio data, neural networks dominate applications.

NEAREST NEIGHBOR CLASSIFICATION

Given: Training data $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \ldots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$.

m-nearest neighbor rule

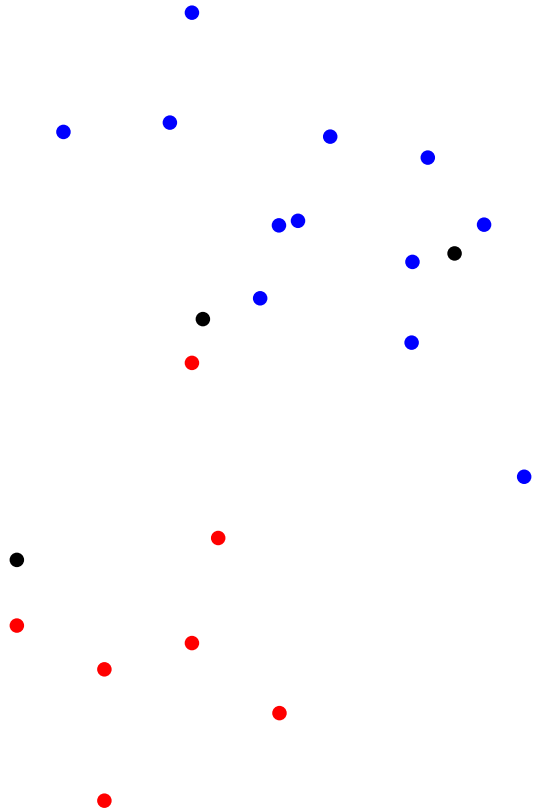
Fix $m \in \mathbb{N}$.

Classify data point $\mathbf{x}$ as:

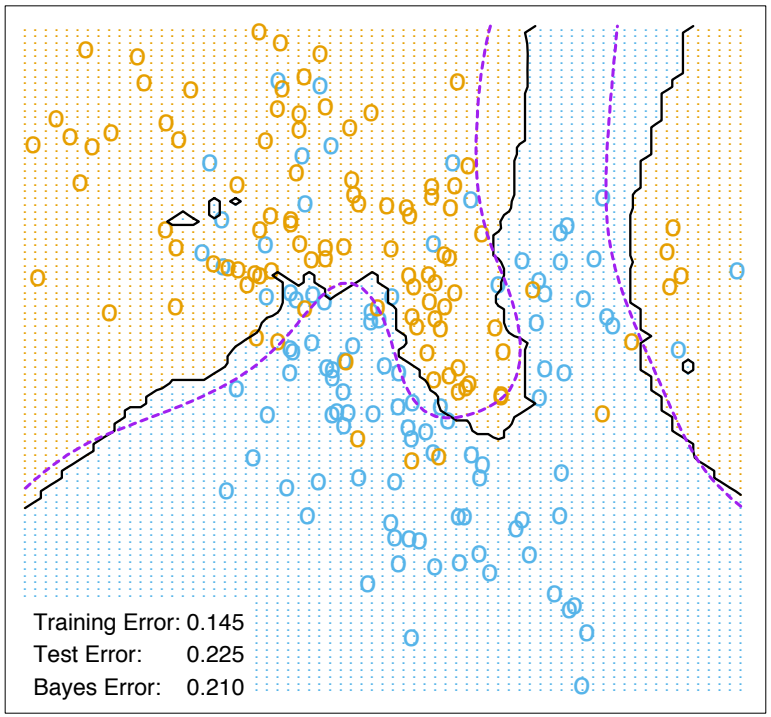
1. Find m training data points that are closest to $\mathbf{x}$ in $\mathbb{R}^d$.
2. Assign $\mathbf{x}$ to the class the majority of these m points belong to.

Remarks

- Works for any number of classes.
- For two classes, m is usually chosen as an odd number to avoid ties. For more than two classes, one has to decide on a tie-breaking strategy in case no single class produces a majority (e.g. choose one of the classes that are in majority at random).
- There is no training algorithm. The training data is used directly to compute the prediction.



EXAMPLE: TWO CLASSES

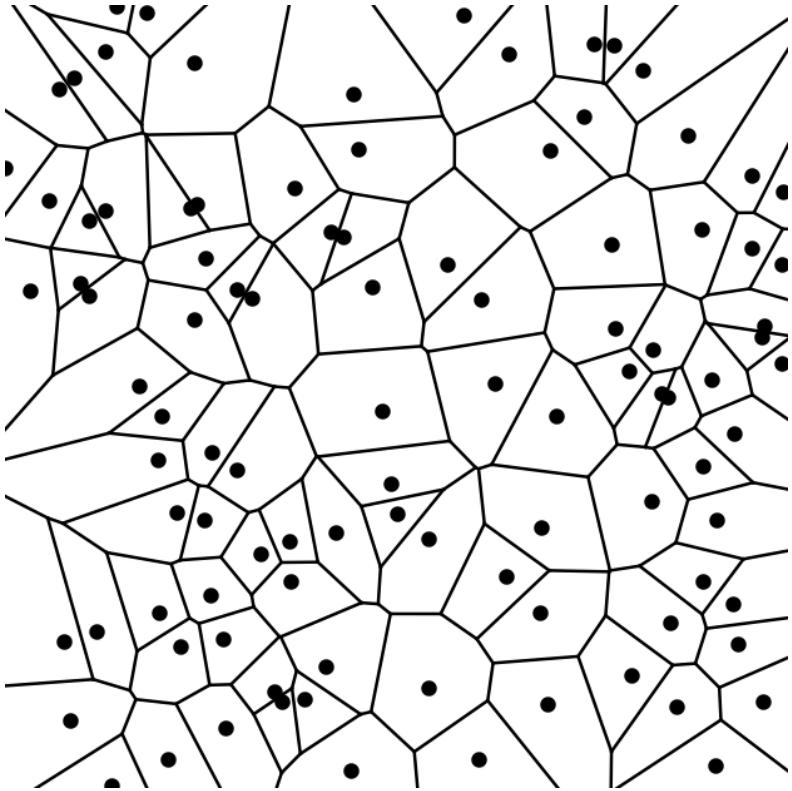


7-NN solution

Source: Hastie/Tibshirani/Freedman, "The elements of statistical learning"

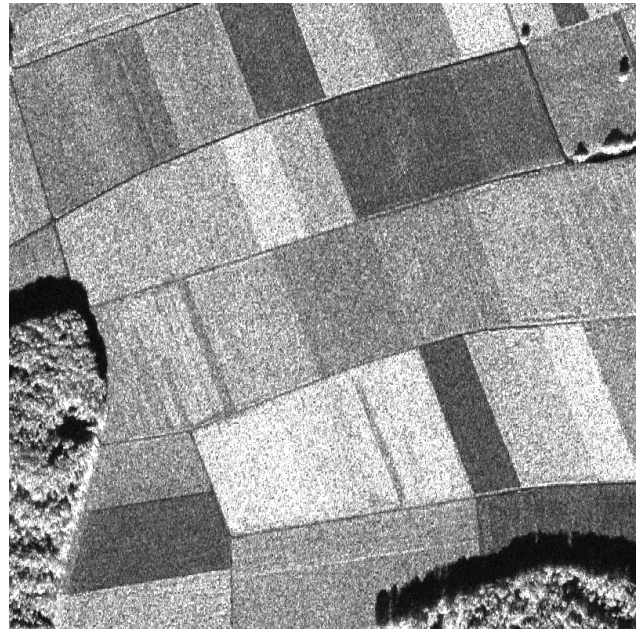
Neighborhood Regions

For $m = 1$, one can plot subdivide $\mathbb{R}^d$ into regions closest to each training point:

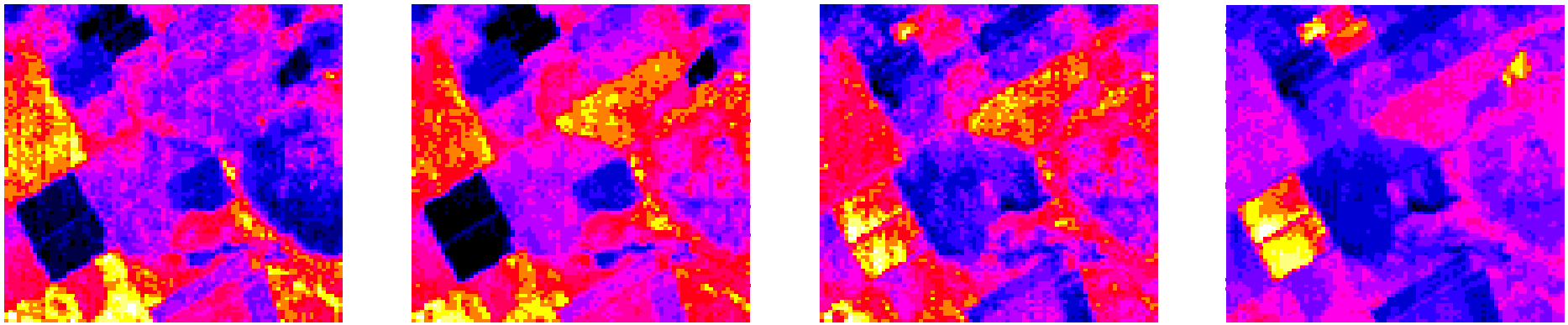


If a data point $\mathbf{x}$ falls into one of the cells, the 1-NN rule will assign it to the class label of the point defining the cell.

For $m > 1$, this becomes harder to plot.



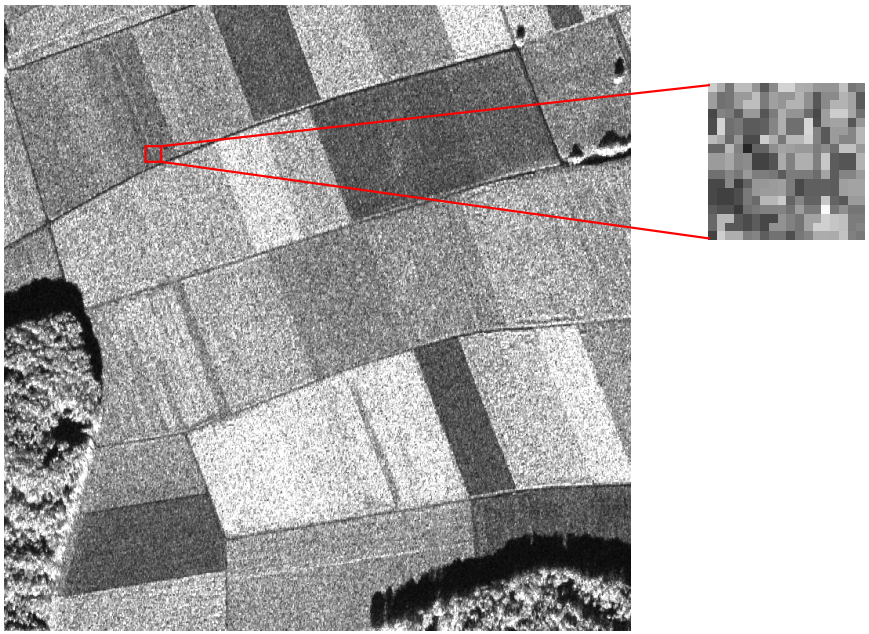
EXAMPLE: LAND USAGE CLASSIFICATION



- These are the four “channels” (spectral bands) of a LANDSAT image.
- The land it shows is used for agriculture.
- There are 7 types of land usage (red soil, cotton, . . .).
- For some images, training data is available.
- The goal is to build a classifier that can classify land use in new images.

Source: Hastie/Tibshirani/Freedman, “The elements of statistical learning”

Extracting local image statistics



1. Place a small window (size $l \times l$) at the location.
2. Extract the pixel values inside the window. Write them into a vector ($\rightarrow$ dimension l^2).

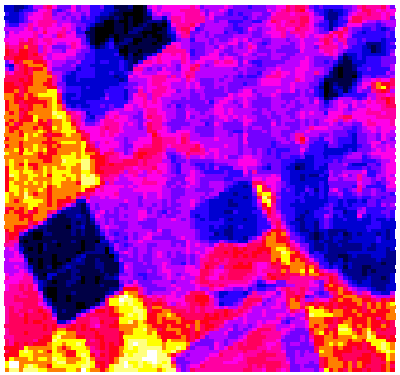
Resulting data

- We use $l = 3$, so each window contains $3 \times 3 = 9$ pixels.
- Since there are four channels we obtain $9 \times 4 = 36$ scalars characterizing each location.
- We use a nearest neighbor classifier on $\mathbb{R}^{36}$.
- To classify locations in a new image: Again extract a vector using a window, and feed that vector into the NN classifier.

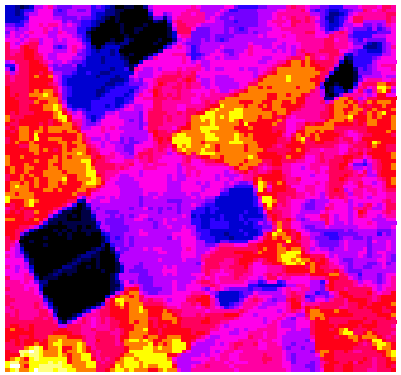
Source: Hastie/Tibshirani/Freedman, "The elements of statistical learning"

Land Usage Prediction

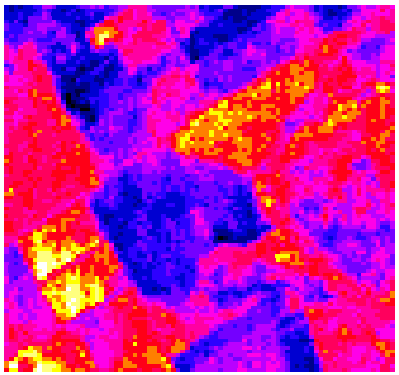
Spectral Band 1



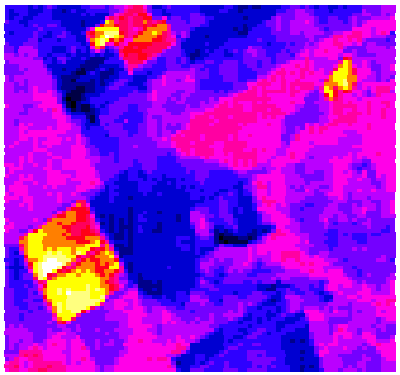
Spectral Band 2



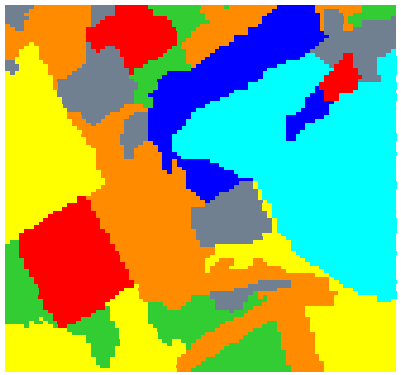
Spectral Band 3



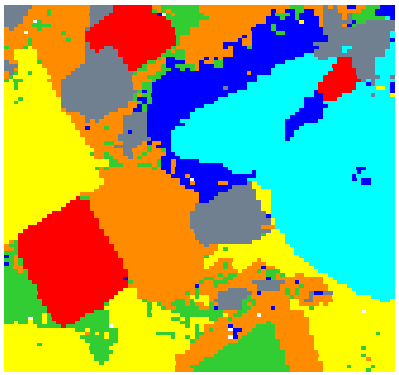
Spectral Band 4



Land Usage

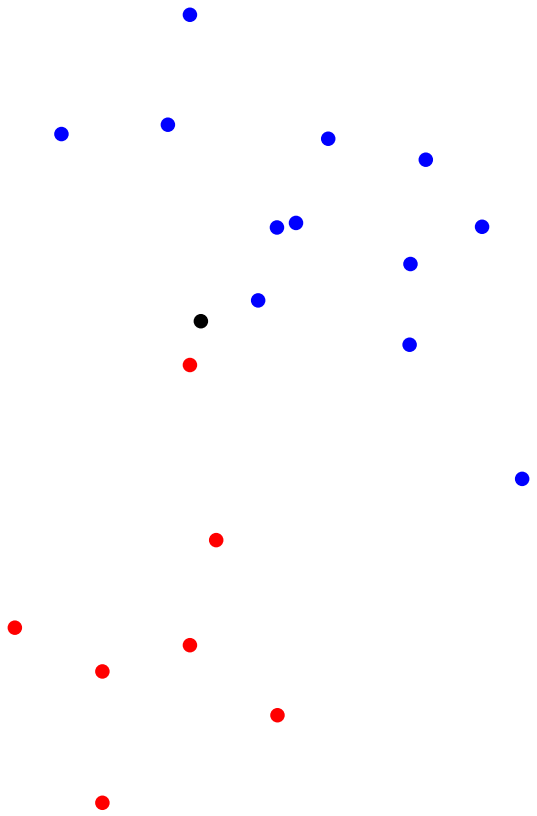


Predicted Land Usage



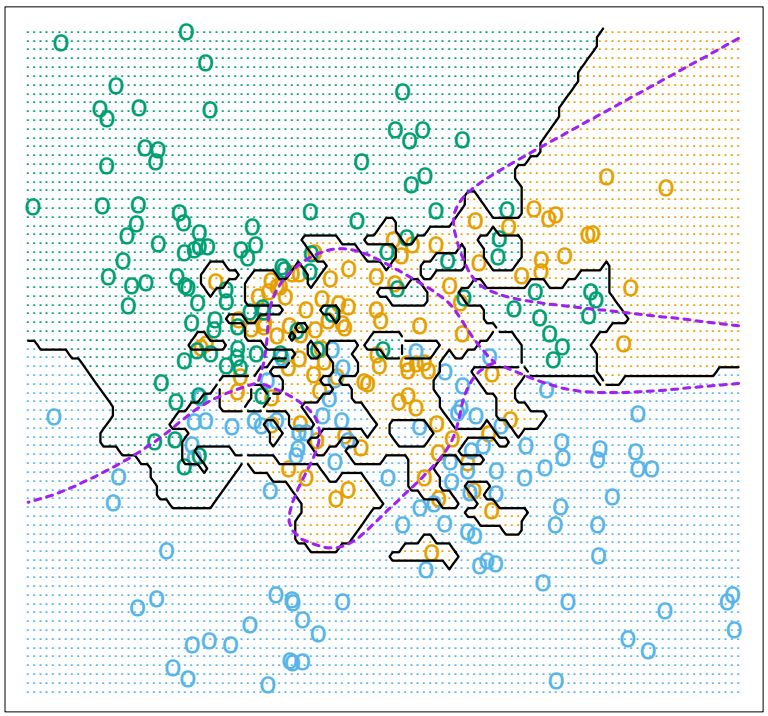
Source: Hastie/Tibshirani/Freedman, "The elements of statistical learning"

EXAMPLE

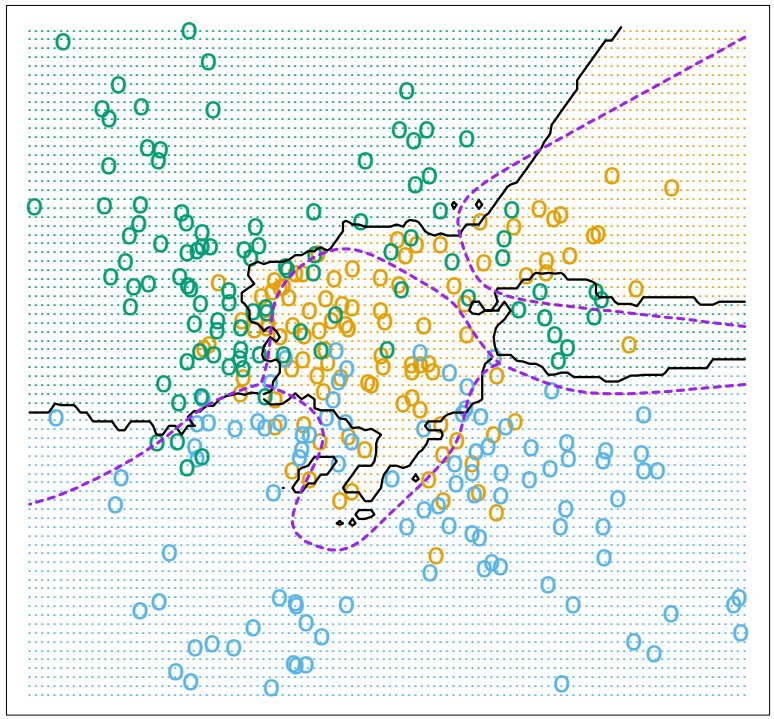


- 1-NN: Classified as “red”.
- 2-NN: Tie.
- m-NN with $m > 2$: Classified as “blue”.

INFLUENCE OF m



1-NN solution



15-NN solution

Source: Hastie/Tibshirani/Freedman, "The elements of statistical learning"

Advantages

- Simple.
- Can be applied to any number of classes.
- Often works very well.

Disadvantages

For large training data sets:

- Requires a lot of memory.
- The entire training set has to be searched for each decision.

Also:

- We are not “learning” anything, even though we can predict.

Evaluating and Tuning a Classifier

What Next

We will consider two problems:

1. Once we have trained a classifier, how do we decide whether it is "good"?
2. We have already seen classifiers with a tuning parameter (e.g. the number of neighbors m in m-NN.) How do we choose the parameter?

Suppose we have trained a classifier f on training data $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \ldots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$.
(We use 0-1 loss, so we simply count mistakes.)

Measuring performance

We can measure performance as an "error rate":

error rate of f = percentage of data points produced by the data source that f misclassifies

Note: For 0-1 loss and i.i.d. data, this coincides with the risk of f.

Interpreting the error rate

Consider a two class problem, where each class is equally probable.

- The "baseline" error rate is 50%. Classifiers that do worse than that are irrelevant.

Explanation:

- If we predict by flipping a fair coin (and completely ignore the data), we will achieve 50% error rate.
- If f has error rate $> 50\%$, we can turn it into a classifier with error rate $< 50\%$ by swapping the classes.

How do we measure the error rate in practice?

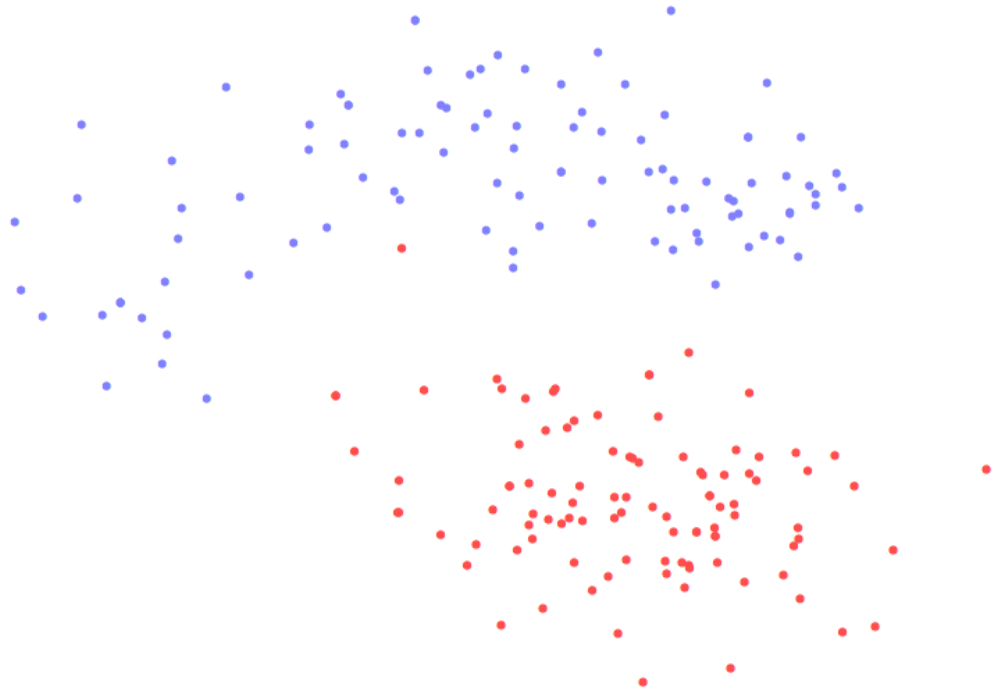
- We don't have access to the data source itself.
- We only have access to data from the source.
- We can estimate the error rate by measuring it on data: If $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ are data points, we can compute

$$\text{error rate of } f \;\approx\; \frac{\sum_{i=1}^{m} \mathbb{I}\{f(\mathbf{x}_i) \neq y_i\}}{m} = \frac{\text{number of misclassified data point}}{\text{number of data points}}$$

- To do so, we need *labeled* data points.

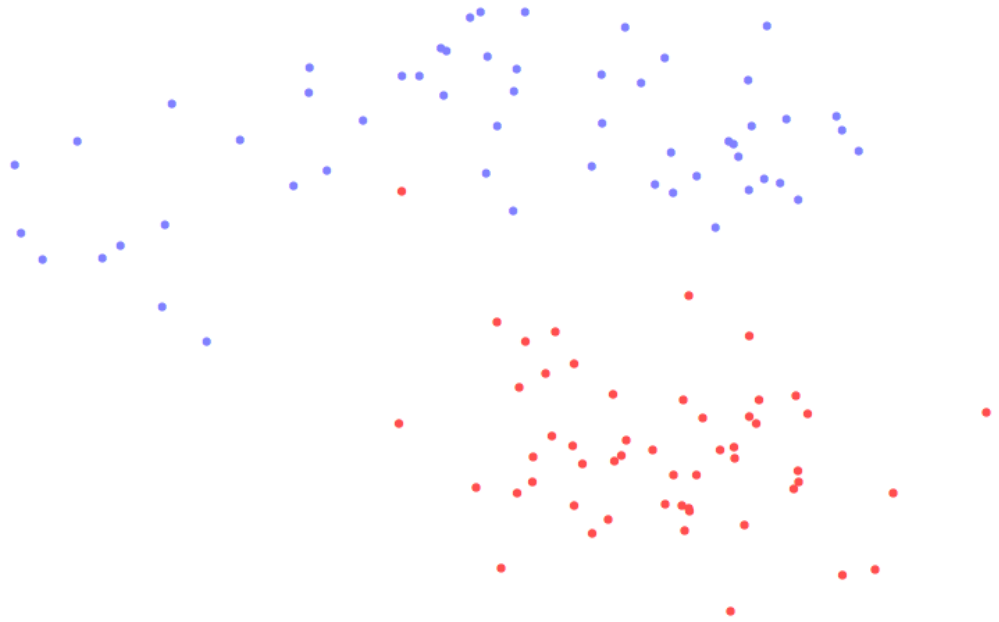
Can we use the training data?

Sample Properties



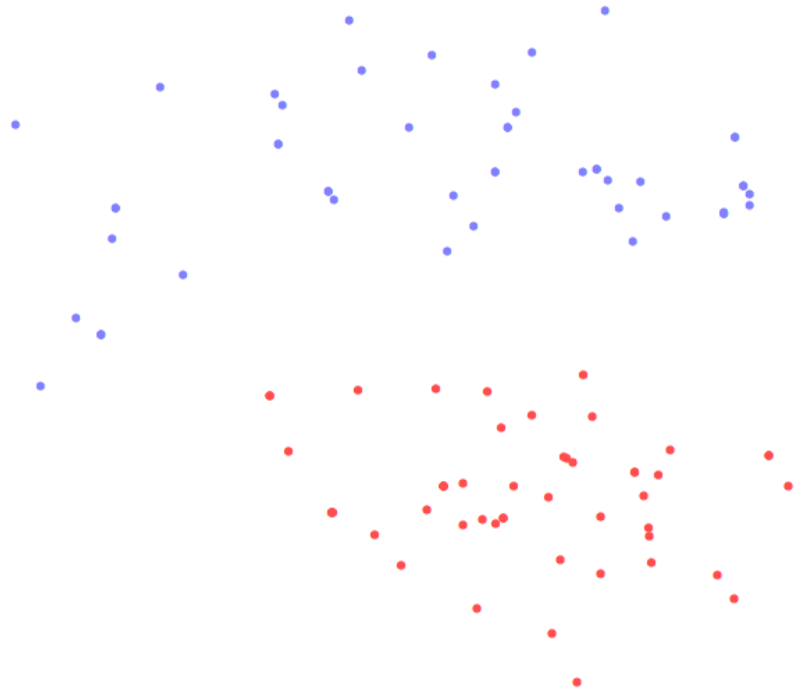
- We observe a sample of n observations from a given data source.
- If we observe a pattern in the data, it can reflect a property of the data source, or it can be a random effect.
- When we train a classifier, it should ideally adapt to properties of the source, but ignore random effects.
- We cannot distinguish the two cases without looking at another sample.

Sample Properties



- We observe a sample of n observations from a given data source.
- If we observe a pattern in the data, it can reflect a property of the data source, or it can be a random effect.
- When we train a classifier, it should ideally adapt to properties of the source, but ignore random effects.
- We cannot distinguish the two cases without looking at another sample.

Sample Properties



- We observe a sample of n observations from a given data source.
- If we observe a pattern in the data, it can reflect a property of the data source, or it can be a random effect.
- When we train a classifier, it should ideally adapt to properties of the source, but ignore random effects.
- We cannot distinguish the two cases without looking at another sample.

What happens if we measure the error rate on the training data?

- If the classifier has over-adapted to the idiosyncrasies of the training data, it will perform better on the training data then on new data from the same source.
- Estimates of error rates computed on the training data tend to *underestimate* the actual error rate.

Solution: Data splitting

- *Before* we train the classifier, we split the labeled data into two parts.
- We call these **training data** and **test data**.
- We use the training data to train the classifier.
- We then use the test data to estimate the error rate.

Types of errors

- The error rate (or, more generally, the empirical risk) evaluated on the training data is called the **training error**.
- The error rate or empirical risk evaluated on the test data is the **test error**.

The distinction between these quantities is crucial.

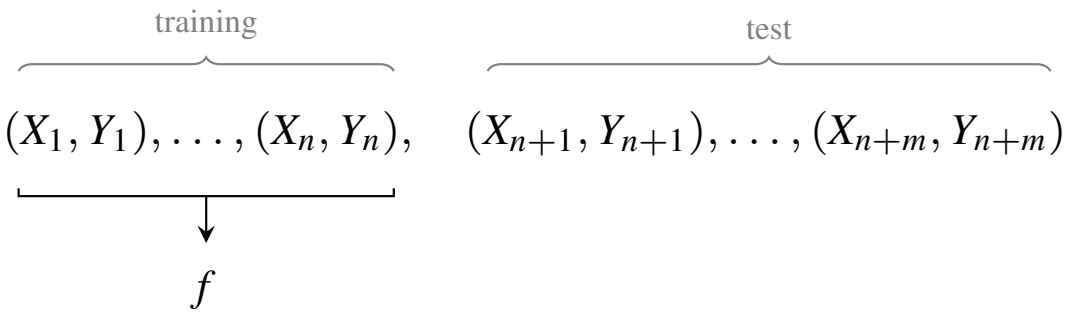
Interpretation

- The training error measures how well the classifier fits the training data.
- The test error estimates how well the classifier predicts.
- Note this is an *estimate* rather than a *measurement*. Measuring the test error would require access to the data distribution. Since we do not have that distribution, we estimate the error from data.

Important

The test data must not be used for training in any way.

Once the training method has used *any* information extracted from the test data, the test error estimate is confounded.

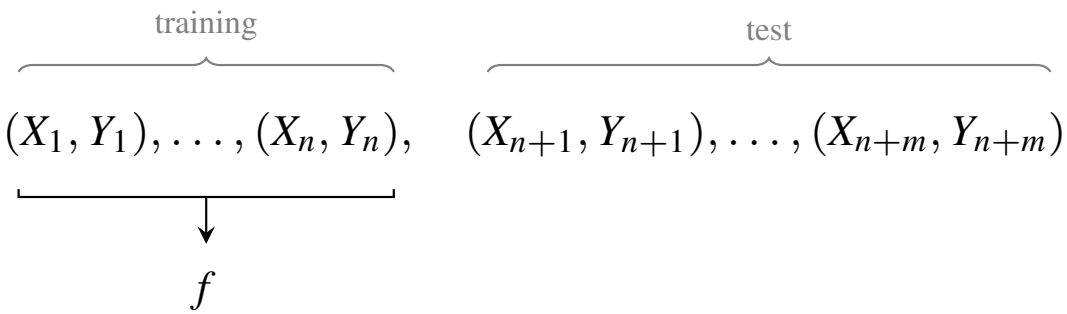


Data

- Suppose a data source generates $n + m$ labelled data points.
- We split these into n training and m test points:

$$(X_1, Y_1), \ldots, (X_n, Y_n), (X_{n+1}, Y_{n+1}), \ldots, (X_{n+m}, Y_{n+m})$$

- We assume that (X_i, Y_i) is independent of (X_j, Y_j), for $i \neq j$.
- That means the data are i.i.d., since they have the same distribution (the data source).



Using the data

- We train a classifier f on the training data. The classifier is obtained from the data by a deterministic procedure. Since the data is random, the classifier is random.

$$(X_1, Y_1), \ldots, (X_n, Y_n), (X_{n+1}, Y_{n+1}), \ldots, (X_{n+m}, Y_{n+m})$$

- Since the test data is independent of the training data, the classifier is stochastically independent of the test data.
- That means an estimate of the classifier's error obtained from the test data is unbiased.
- If training uses *any* information from the test data, the classifier and the test data become dependent.
- Typically, the effect of this dependence is that the test error *systematically underestimates* the actual prediction error on data from the source.

TREE CLASSIFIERS

Idea

- Recall: Classifiers classify according to location in $\mathbb{R}^d$
- Linear classifiers: Divide space into two halfspaces
- What if we are less sophisticated and divide space only along axes? We could classify e.g.

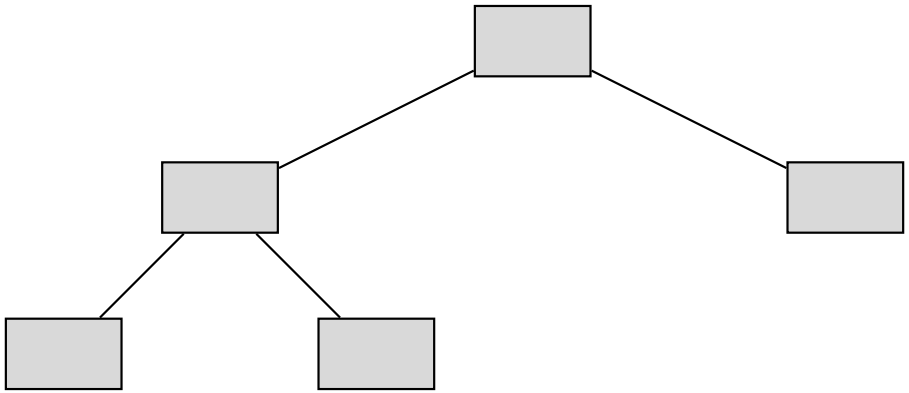
$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \qquad \text{according to} \qquad \mathbf{x} \in \begin{cases} \text{Class +} & \text{if } x_3 > 0.5 \\ \text{Class -} & \text{if } x_3 \leq 0.5 \end{cases}$$

- This decision would correspond to an affine hyperplane perpendicular to the x_3-axis, with offset 0.5.

Tree classifier

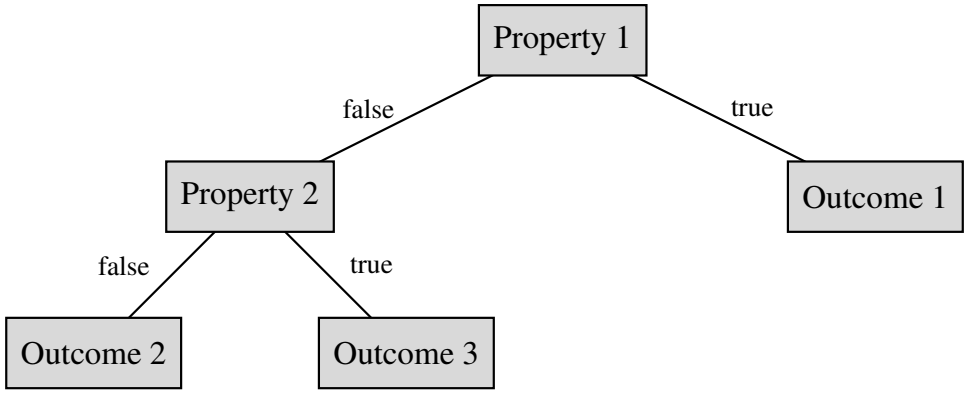
- A tree classifier combines several simple decision rules as the one above into a classifier using a so-called *decision tree*.

TREES



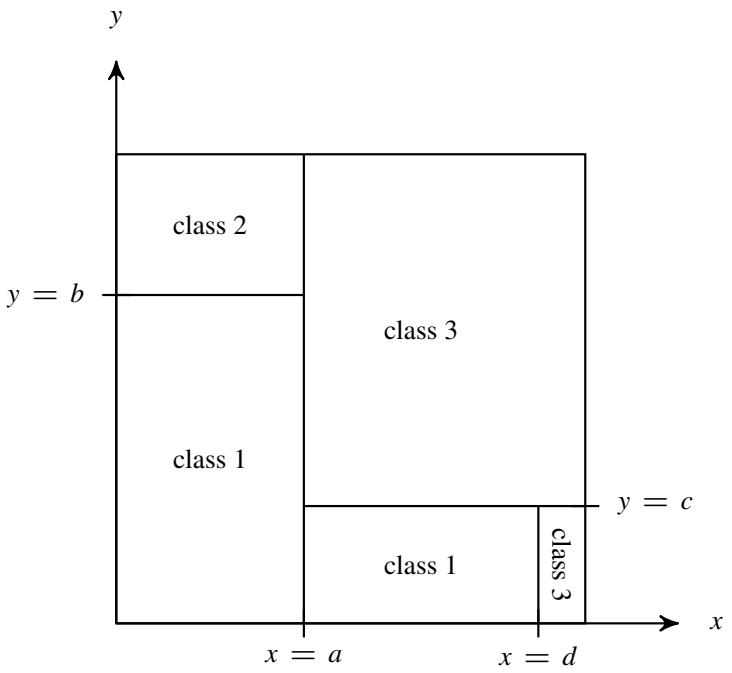
- A **tree** is a diagram consisting of **nodes** (marked as gray boxes above) and **edges** (the connecting lines).
- The topmost node is called the **root**. Each (except the root) is connected to exactly one node above it, called its **parent**.
- Nodes can be connected other nodes below them, called their **children**.
- Nodes at the bottom (those with no children) are called **leaves**.
- If each node has either two or no children, the tree is called a **binary tree**.

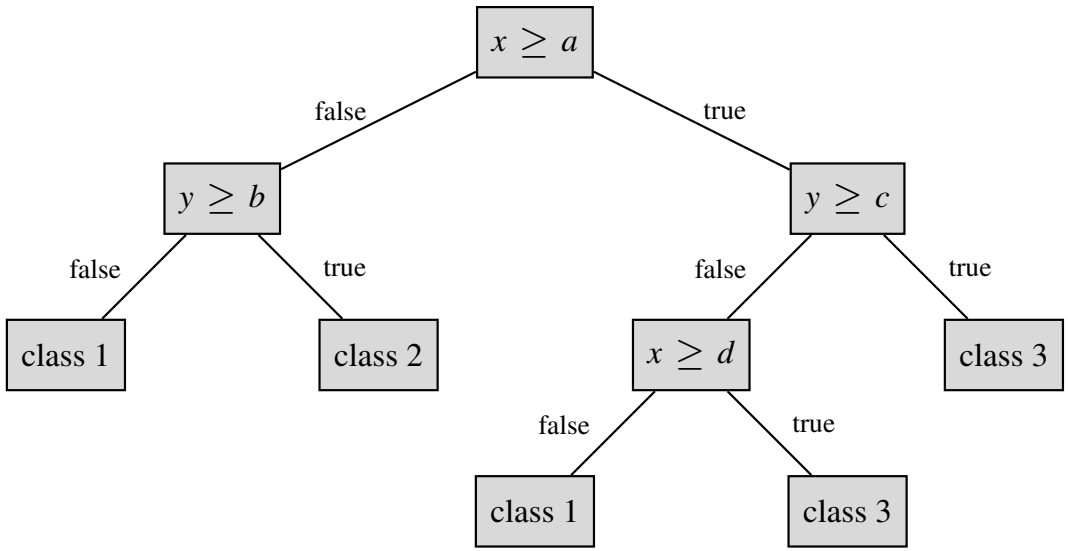
Decision Trees



- Binary trees can be used as decision diagrams.
- Each inner node (a node that is not a leaf) represents a property.
- The two children of the node represent the cases “property is false” or “property is true”.
- Each leaf represents an outcome. That means: An outcome is a combination of true and false properties.
- Such a tree is called a **decision tree**.

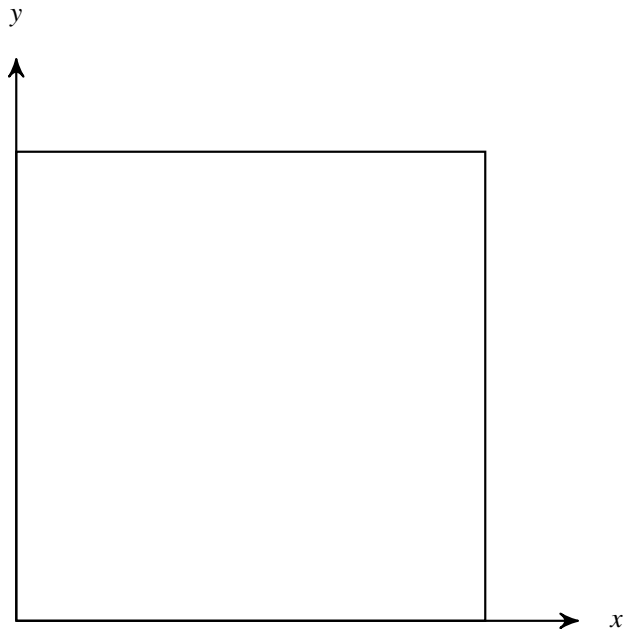
TREE CLASSIFIER





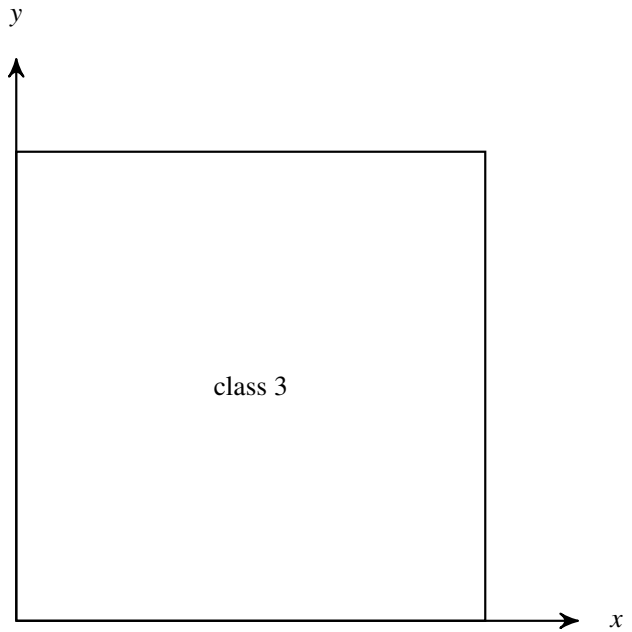
- A **tree classifier** in $\mathbb{R}^d$ is a decision tree.
- Each property at an inner node corresponds to a decision of the form $x_j \geq c$, where $j \in \{1, \ldots, d\}$ is one of the coordinates, and $c \in \mathbb{R}$ is a constant.
- Each leaf corresponds to a class.
- We classify a data point $\mathbf{x} \in \mathbb{R}^d$ by starting at the root, and following the decisions through the diagram until we reach a leaf. We then assign $\mathbf{x}$ to the class inscribed at that leaf.

Example: Data in quadratic domain, three classes, class 3 is the largest class.



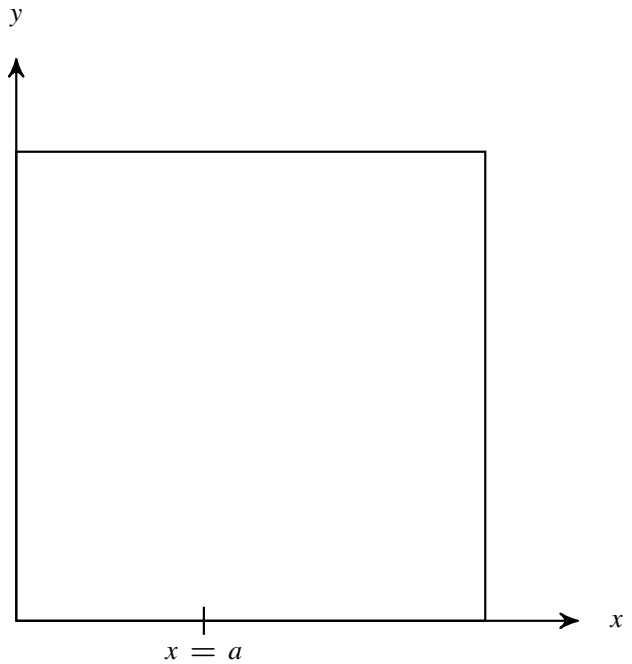
Growing a Tree

Example: Data in quadratic domain, three classes, class 3 is the largest class.



class 3

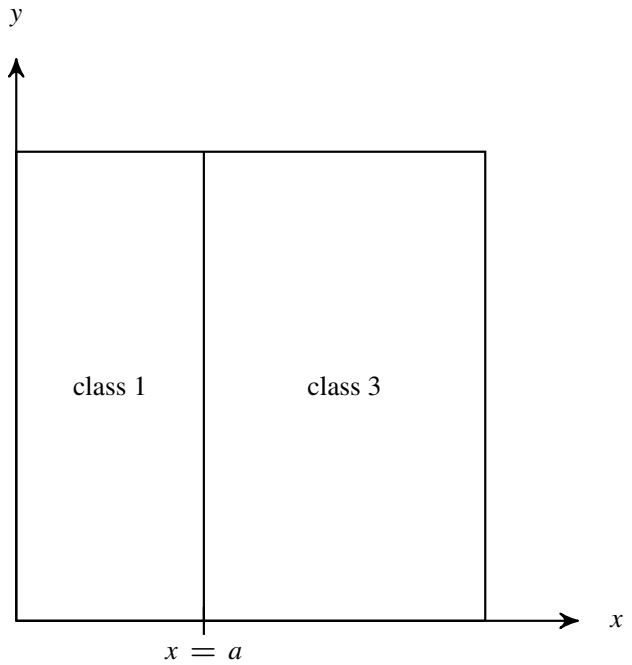
Example: Data in quadratic domain, three classes, class 3 is the largest class.

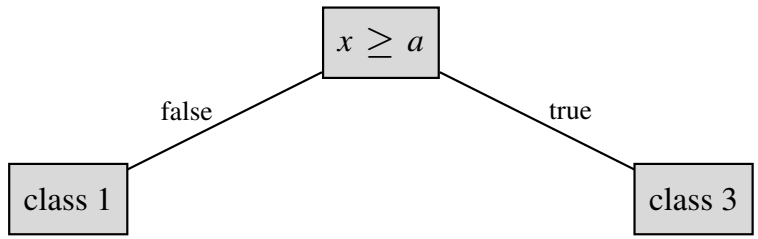


$x \geq a$

Growing a Tree

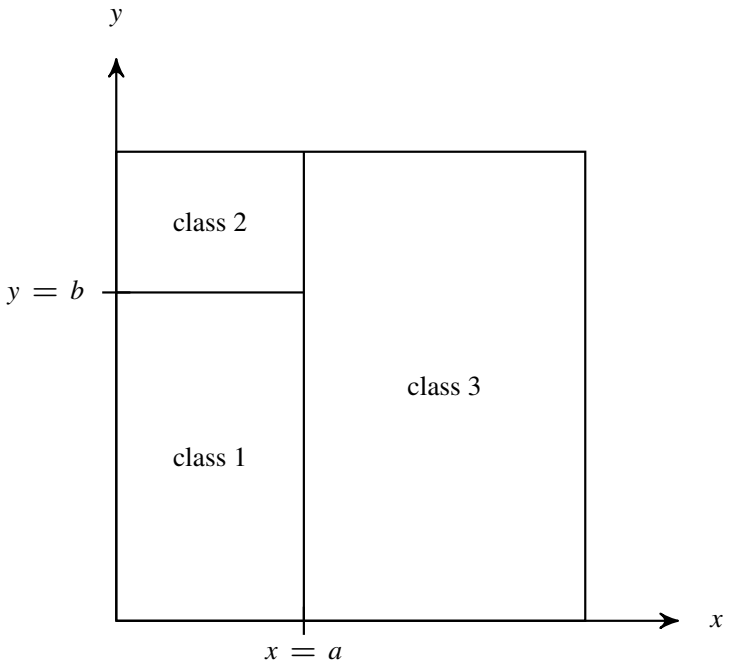
Example: Data in quadratic domain, three classes, class 3 is the largest class.

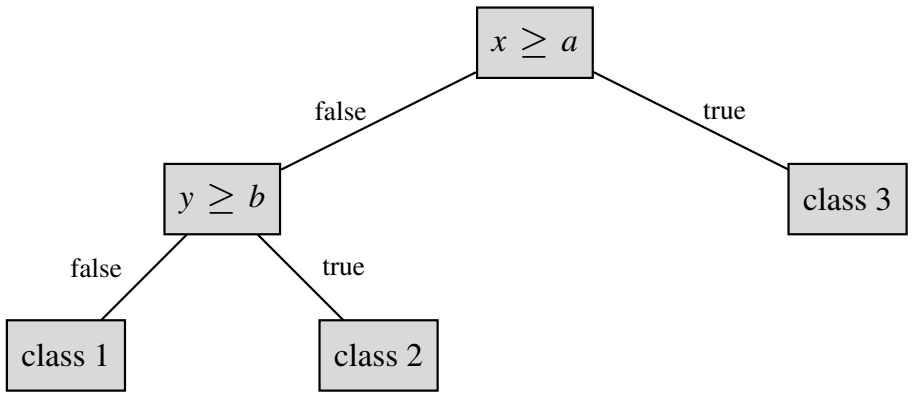




Growing a Tree

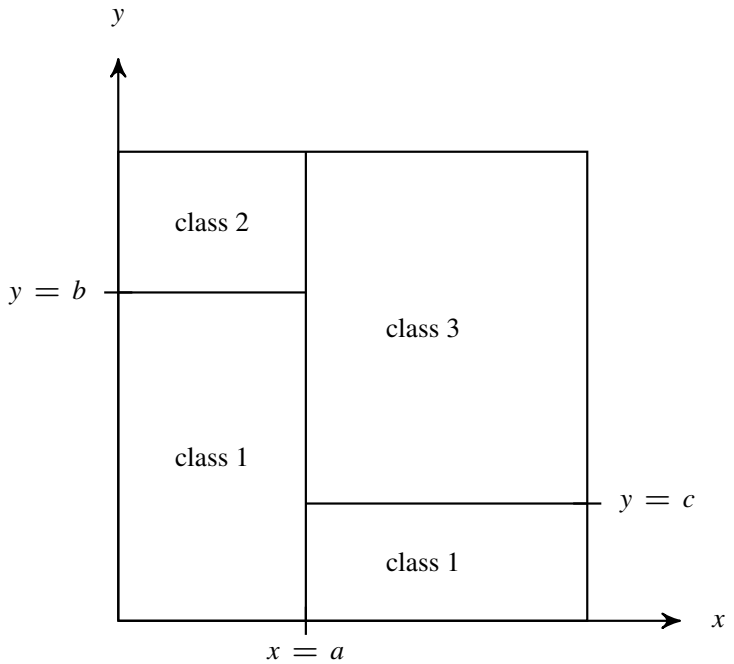
Example: Data in quadratic domain, three classes, class 3 is the largest class.

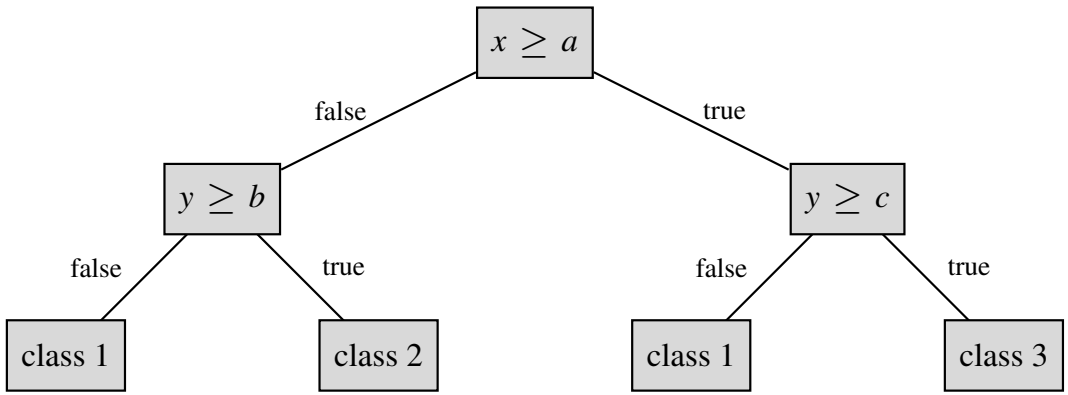




Growing a Tree

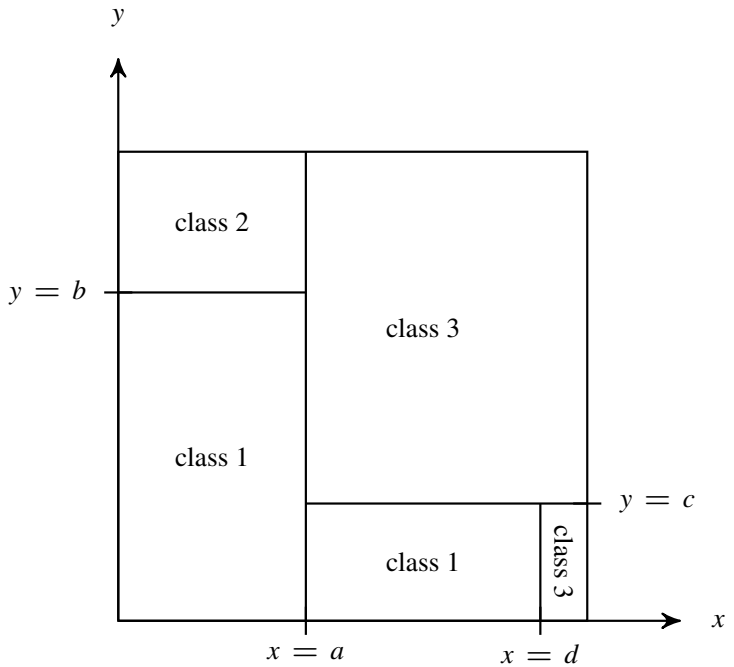
Example: Data in quadratic domain, three classes, class 3 is the largest class.

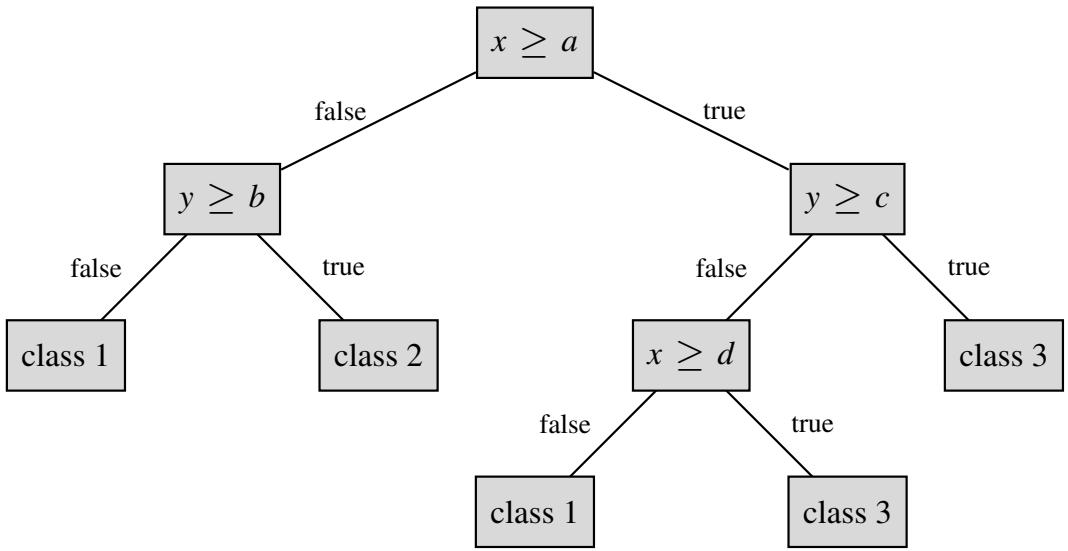




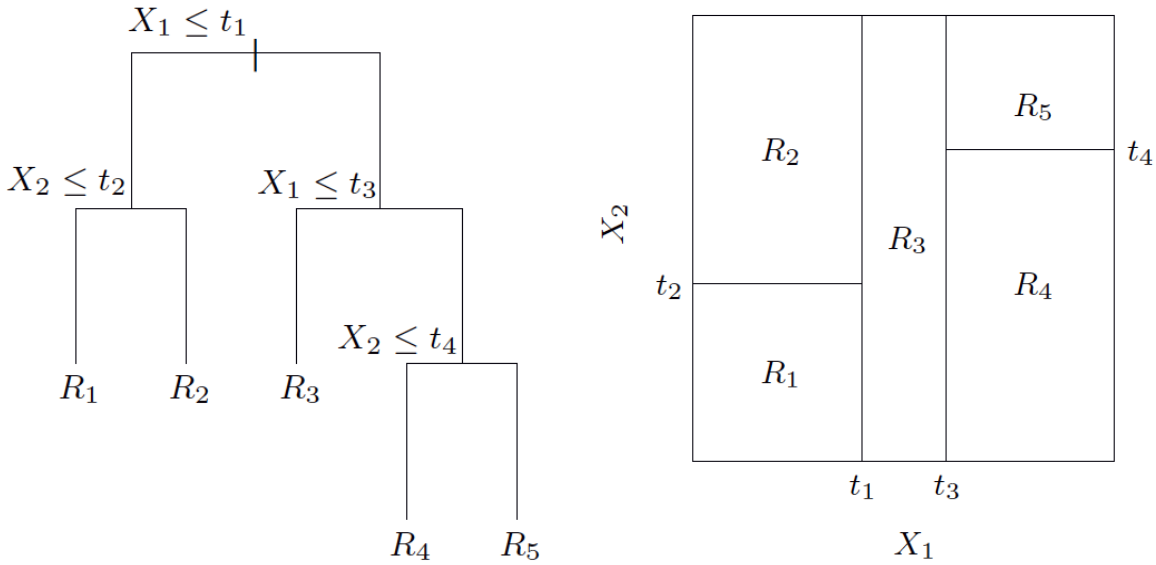
Growing a Tree

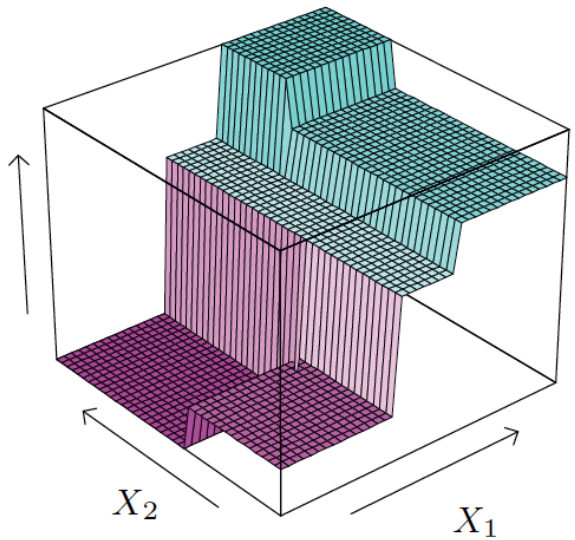
Example: Data in quadratic domain, three classes, class 3 is the largest class.





TREES





- Each leaf of the tree corresponds to a region R_m of $\mathbb{R}^d$.

Figures: Hastie, Tibshirani, Friedman, "The elements of statistical learning"

Approach

The basic strategy is very simple:

- At each step, decide where to place the next split.
- That replaces one of the regions represented by the tree by two new regions.
- Assign each new region by majority vote among the training data points in that region.

Where do we split?

We have to decide:

- Which region should be split.
- Along which axis.
- At which split point.

Idea: Find the split that results in the largest reduction in training error.

Cost of a split

- Suppose we split region $\mathcal{R}_m$ along axis j at value t.
- That results in two new regions, say $\mathcal{R}_m^1$ and $\mathcal{R}_m^2$.
- In the tree, that means we replace the node $\mathcal{R}_m$ by the criterion $x_j \geq t$, and add $\mathcal{R}_m^1$ and $\mathcal{R}_m^2$ as child nodes.
- We define the **cost** of this split as

$$\text{cost}(m, j, t) := \quad \#\text{ of misclassified points in } \mathcal{R}_m^1 \quad + \quad \#\text{ of misclassified points in } \mathcal{R}_m^2$$

(That means: We assign $\mathcal{R}_m^1$ and $\mathcal{R}_m^2$ class labels by majority vote, and check how many training points are misclassified by these class labels.)

Training a tree classifier

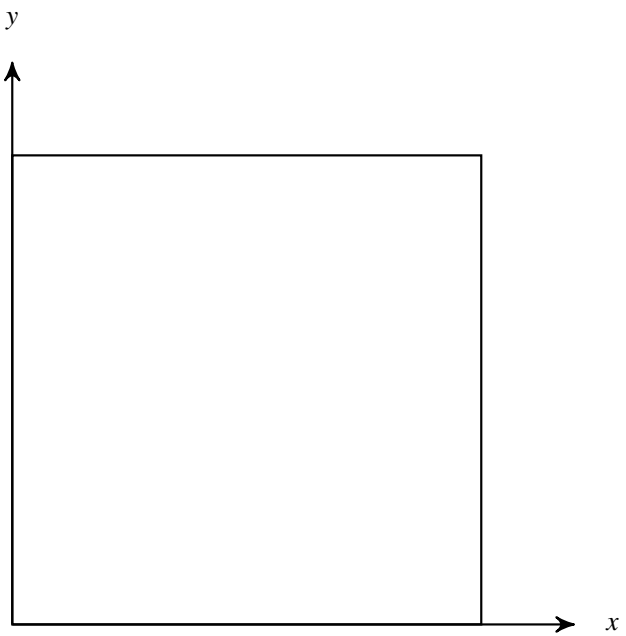
- For each region m and each axis j, find the split point t that minimizes $\text{cost}(m, j, t)$,

$$t_{mj} := \arg\min_{t \in \mathbb{R}} \text{cost}(m, j, t) \ .$$

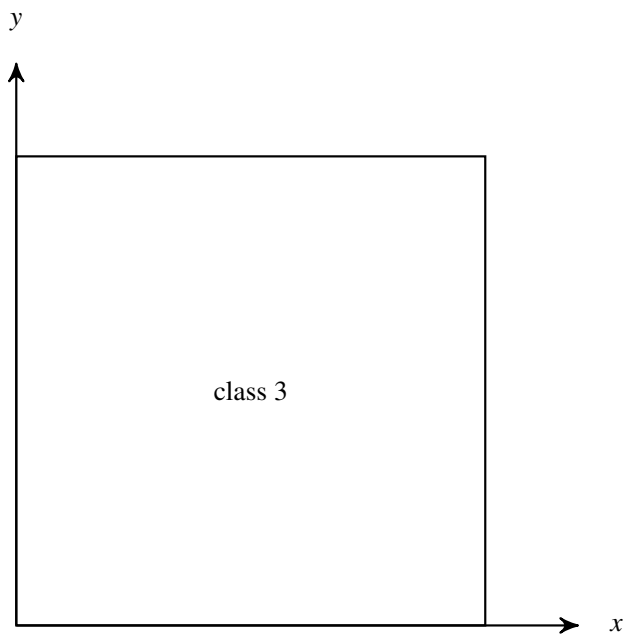
- From the list of all such points t_{mj}, pick the one with the smallest cost.
- Perform that split.

We keep doing so until the number of regions m reaches some specified, maximal value M.

Example: Data in quadratic domain, three classes, class 3 is the largest class. We specify the maximum number of regions as $M = 5$.

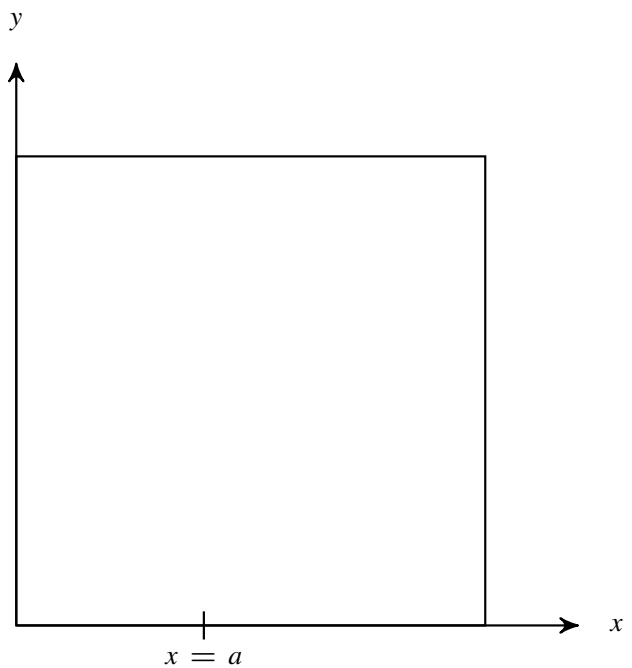


Example: Data in quadratic domain, three classes, class 3 is the largest class. We specify the maximum number of regions as $M = 5$.



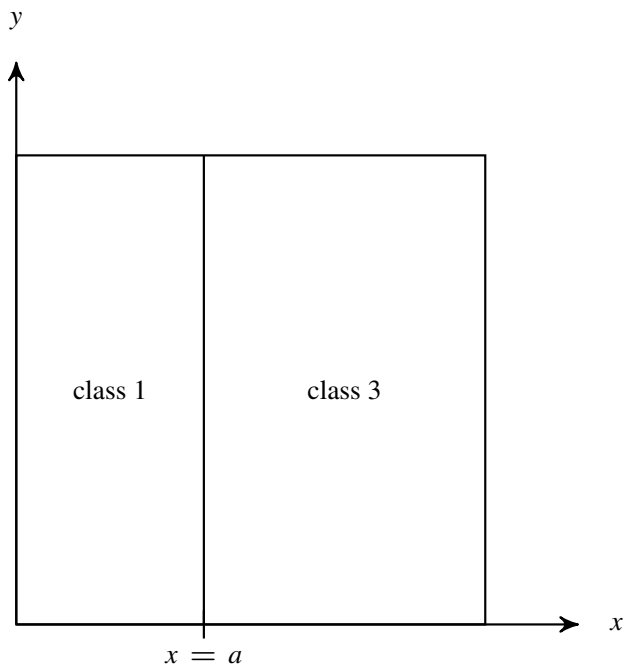
class 3

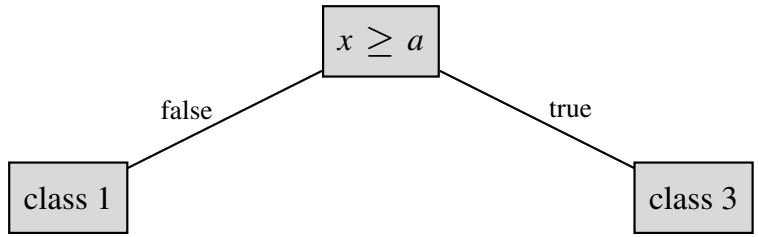
Example: Data in quadratic domain, three classes, class 3 is the largest class. We specify the maximum number of regions as $M = 5$.



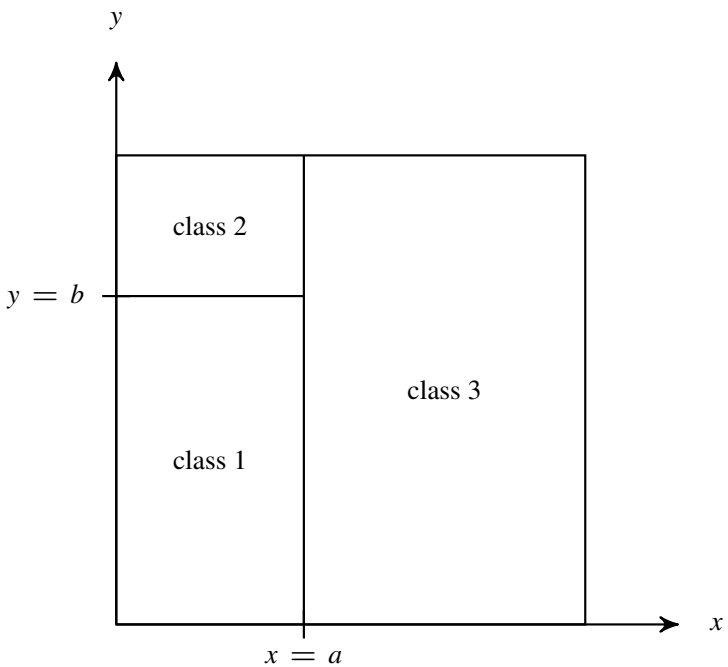
$x \geq a$

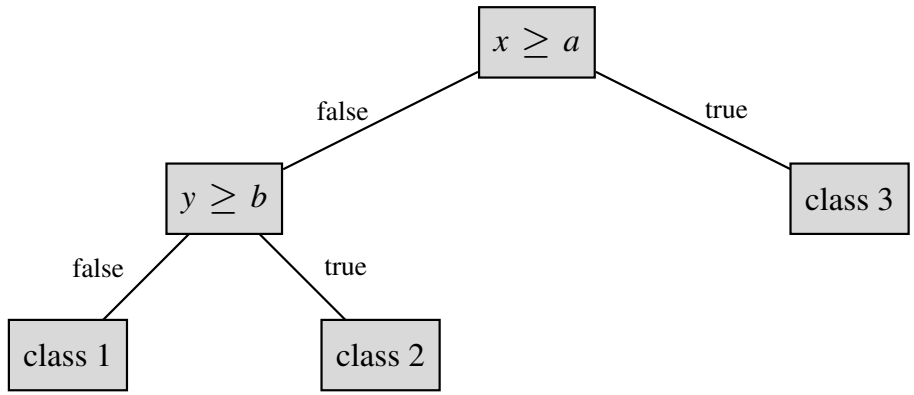
Example: Data in quadratic domain, three classes, class 3 is the largest class. We specify the maximum number of regions as $M = 5$.



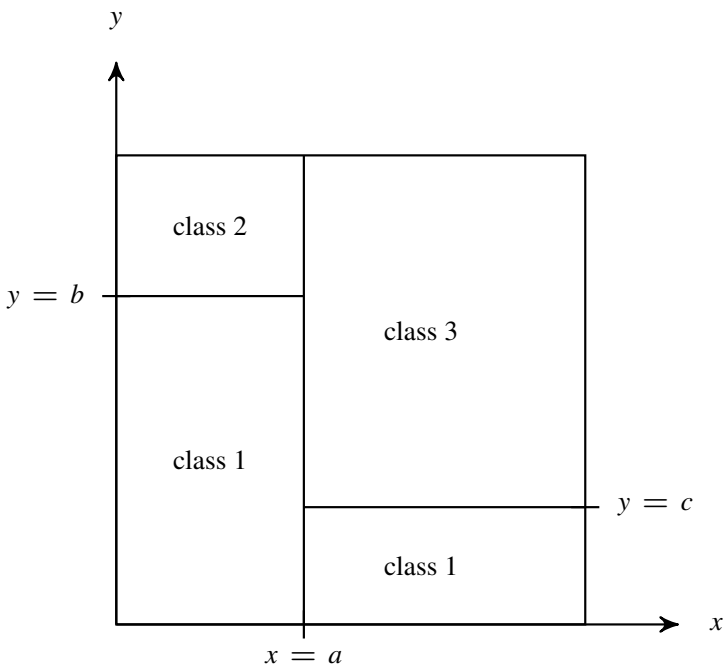


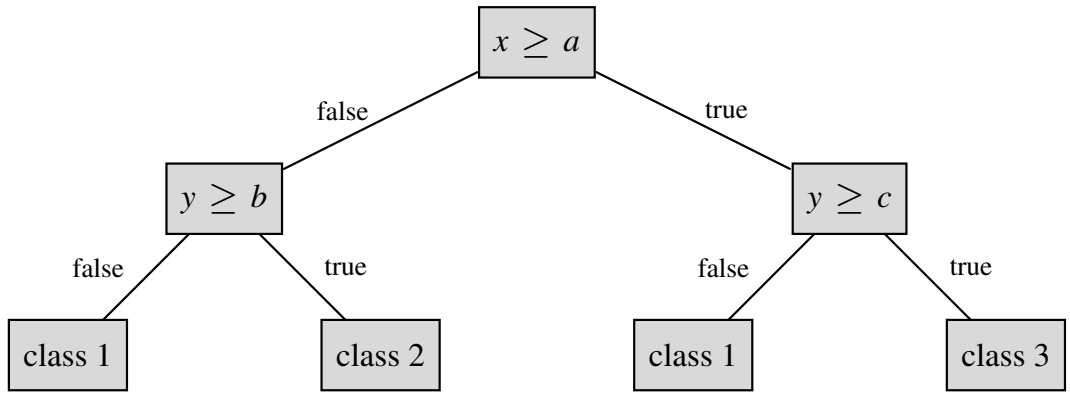
Example: Data in quadratic domain, three classes, class 3 is the largest class. We specify the maximum number of regions as $M = 5$.



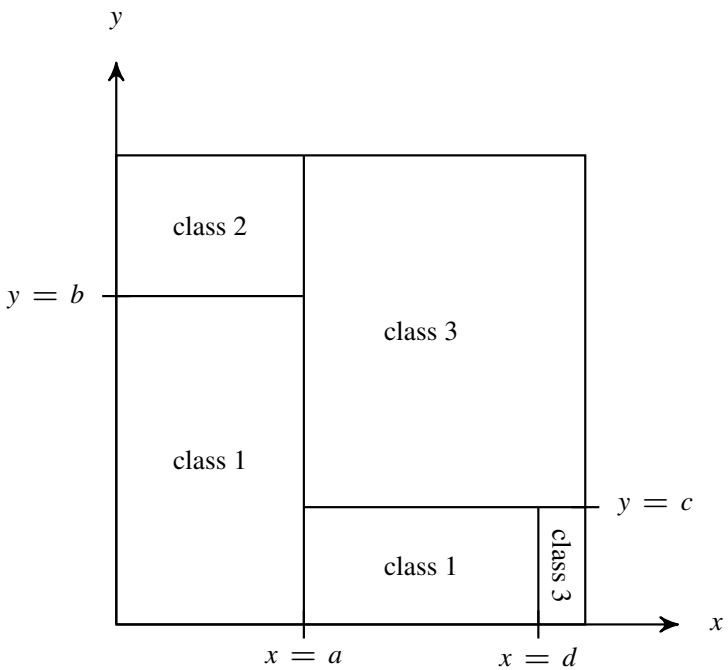


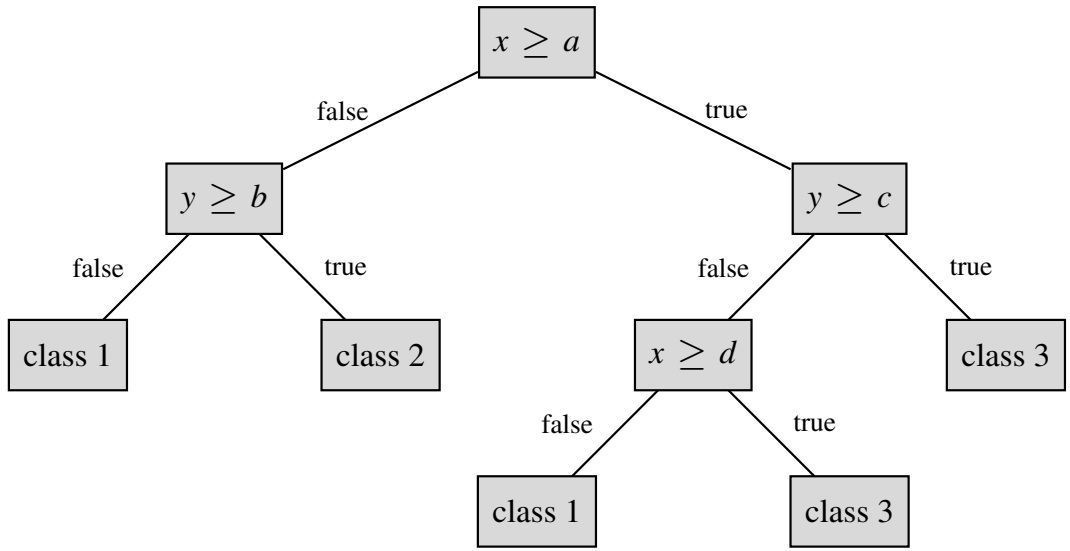
Example: Data in quadratic domain, three classes, class 3 is the largest class. We specify the maximum number of regions as $M = 5$.





Example: Data in quadratic domain, three classes, class 3 is the largest class. We specify the maximum number of regions as $M = 5$.





EXAMPLE: SPAM FILTERING

Data

- 4601 email messages
- Classes: email, spam

	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

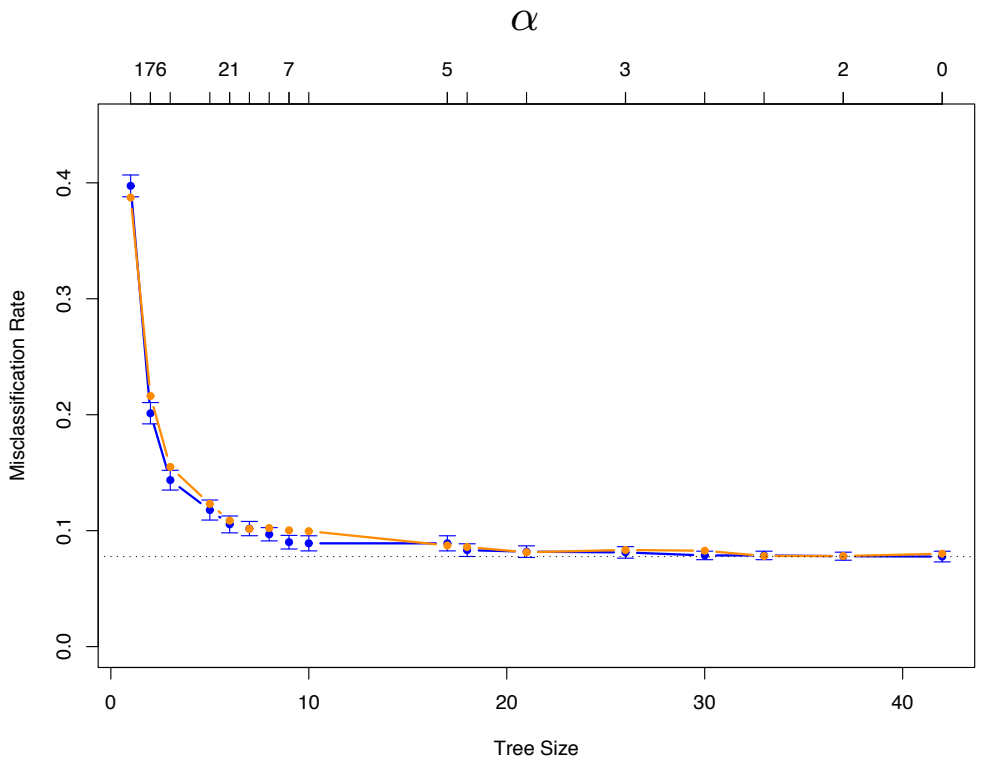
Tree classifier

- 17 nodes
- Performance:

	Predicted	
True	Email	Spam
Email	57.3%	4.0%
Spam	5.3%	33.4%

Figures: Hastie, Tibshirani, Friedman, "The elements of statistical learning"

Influence of Tree Size

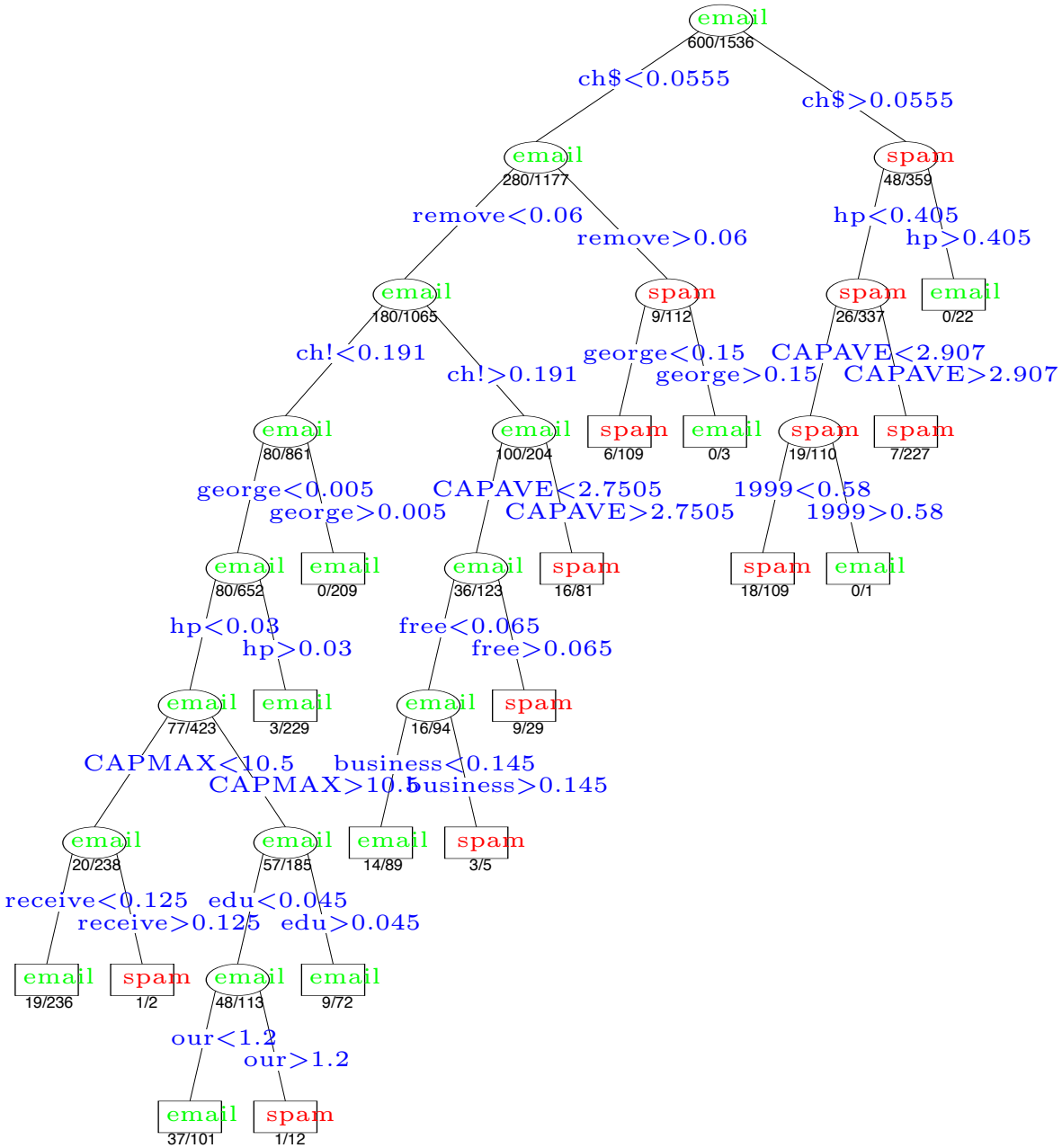


Tree Size

- Complete tree of height D defines 2^D regions.
- D too small: Insufficient accuracy. D too large: Overfitting.
- D can be determined by cross validation or more sophisticated methods ("complexity pruning" etc), which we will not discuss here.

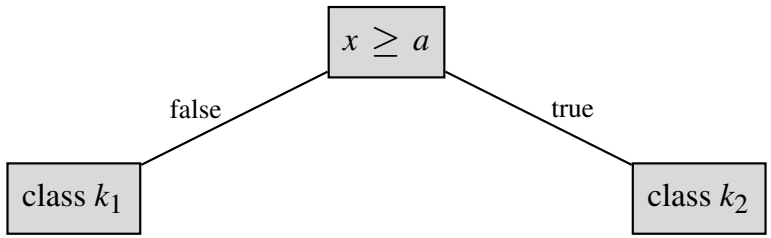
Figures: Hastie, Tibshirani, Friedman, "The elements of statistical learning"

SPAM FILTERING: TREE



Figures: Hastie, Tibshirani, Friedman, "The elements of statistical learning"

Decision Stumps

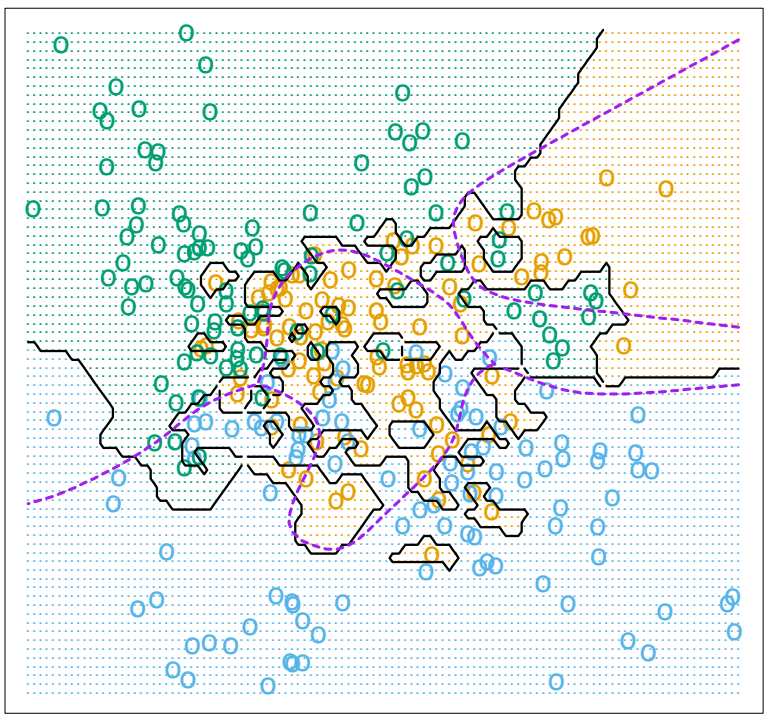


- The simplest possible tree classifier is a tree of depth 1. Such a classifier is called a **decision stump**.
- A decision stump is parameterized by a pair (j, t_j) of an axis j and a splitting point t_j.
- Splits $\mathbb{R}^d$ into two regions.
- Decision boundary is an affine hyperplane which is perpendicular to axis j and intersects the axis at t_j.
- Decision stumps are often used in so-called *ensemble methods*. These are algorithms that combine many poor classifiers into a good classifier. We will discuss ensemble methods later.

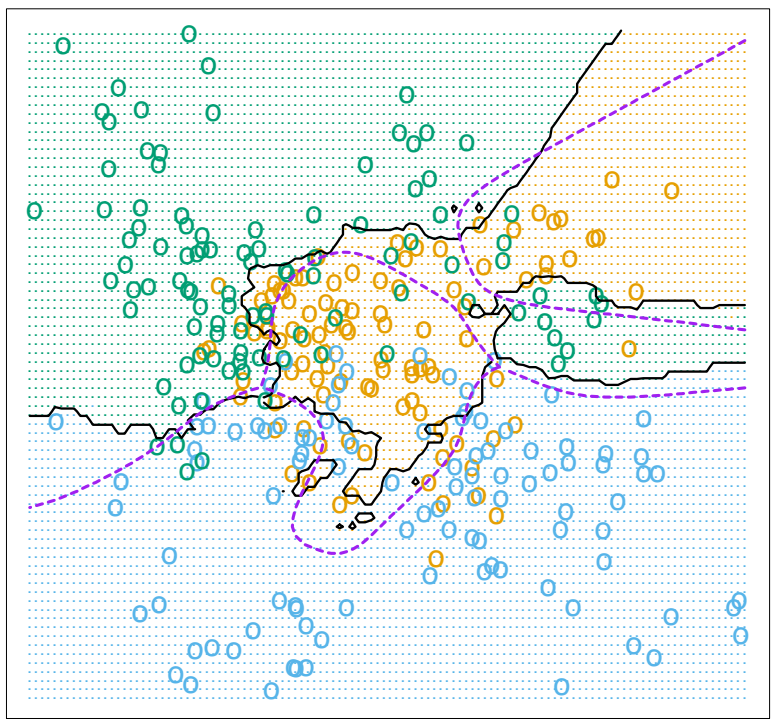
Model Selection and Cross Validation

OVERFITTING

- We had already noted that a classifier can adapt "too closely" to a training data set.
- If the classifier represents idiosyncracies of the training sample rather than the properties of the data source, it achieves small training error, but will not perform well on new data generated by the same data source.
- This phenomenon is called **overfitting**.



1-NN: Overfitting



15-NN: Better generalization

Source: Hastie/Tibshirani/Freedman, "The elements of statistical learning"

Overfitting and flexibility

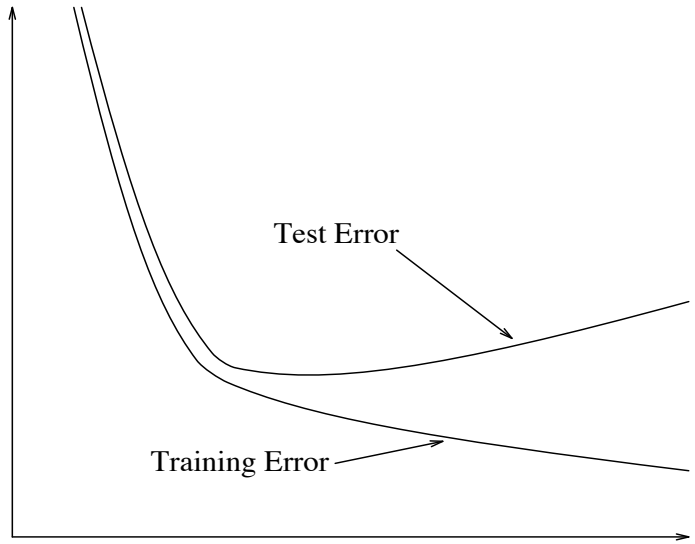
- How prone a classifier is to overfitting depends on how "flexible" it is.
- Linear classifier are not very likely to overfit.

Example: Trees

- A tree of depth 1 is a linear classifier, and will not overfit any reasonably large data set.
- A tree with many splits can subdivide the sample space into small regions.
- Suppose we train a tree with M splits on n training points. If $M \approx n$, the tree can separate almost every training point off into a separate region. That is overfitting: The tree memorizes the training data.

More complex (= flexible) models are more likely to overfit.

Conceptual illustration



Vertical: Error Horizontal: Model complexity (e.g. number of splits in tree)

- If classifier can adapt (too) well to data: Small training error, but possibly large test error.
- If classifier can hardly adapt at all: Large training and test error.
- Somewhere in between, there is a sweet spot.
- Trade-off is controlled by the parameter.

Implications for Training

- If we permit the training algorithm to make a classifier more flexible, it is likely to overfit.
- Example: If the training algorithm for a tree classifier can perform an arbitrary number of splits, it can achieve zero training error.

Avoiding overfitting

- Parameters that control flexibility (like the maximal number of splits in a tree) should be fixed during training.
- We have to develop alternative strategies to choose values for those parameters.

Types of parameters

It is customary to separate parameters into two types:

1. **Model parameters**
   The parameters that specify the solution.
   Examples:
   - Normal vector and offset of a linear classifier
   - Split points and tree structure of a tree classifier.

   (m-nearest neighbor classifiers are an exception: They have no such parameters.)
2. **Hyperparameters**
   Parameters that control the complexity of the solution.
   Examples:
   - Number of splits of tree classifier
   - number m of neighbors in m-nearest neighbor

Hyperparameters cannot be chosen by the training algorithm.

Selecting values for the parameters

- The model parameters are estimated by the training algorithm.
- Hyperparameters are often determined using data splitting methods.

Models

- Data mining and machine learning often loosely refers to a method as a *model*. It is difficult to give a definition that is both general and precise.
- A definition that often works for classification is: A **model** is the set of all possible classifiers that a given method can fit to training data. Each individual solution is often called a **hypothesis**.

Examples

- Linear classifier in $\mathbb{R}^d$: Model = all possible affine planes in $\mathbb{R}^d$, hypothesis = a specific affine plane.
- Tree classifer in $\mathbb{R}^d$: Model = all possible tree classifiers with a fixed number M of splits, hypothesis = classifier defined by one particular tree.

Models and hyperparameters

- Typically, all classifiers within a model should have the same complexity.
- For example: We think of trees with 1 split and trees with 2 splits as two distinct models.
- More generally: Different hyperparameter values define different models.

Objective

- Cross validation is a method which tries to select the best model (e.g. all tree classifiers with 3 splits) from a given family of models (e.g. all tree classifiers).
- This is done using data splitting. Cross validation is a data splitting “protocol”.
- Assumption: Quality measure is predictive performance.
- "Set of models" can simply mean "set of different parameter values".

Terminology

- The process of choosing a good model within a family of models is called **model selection**.

Cross Validation for Model Selection: Procedure

(From now on, we just write γ to denote the entire set of hyperparameters.)

Model selection

1. Randomly split data into three sets: training, validation and test data.

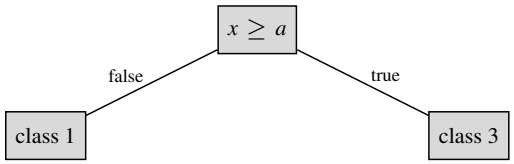
2. Train classifier on training data for different values of γ.
3. Evaluate each trained classifier on validation data (ie compute error rate).
4. Select the value of γ with lowest error rate.

Model assessment

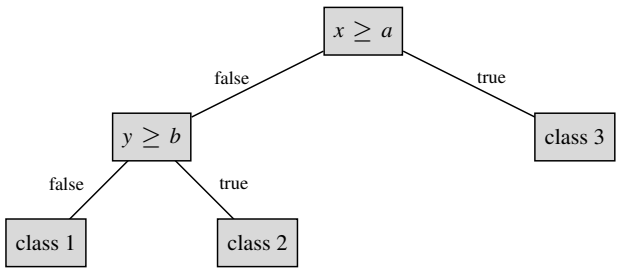
5. Finally: Estimate the error rate of the selected classifier on test data.

Basic CV for a Tree

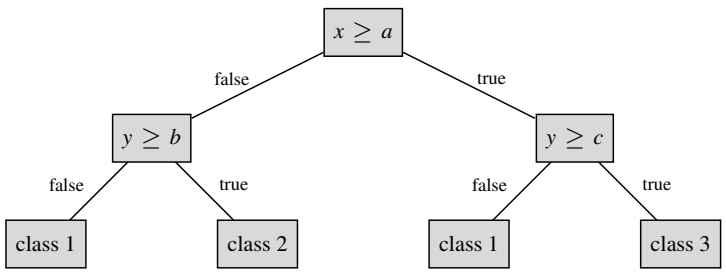
$M = 1$:



$M = 2$:



$M = 3$:



CV procedure

- Split labeled data into a training, validation and test set.
- For each M in $\{1, 2, 3\}$: Train a tree classifier with M splits on the training set.
- For each M, estimate the error rate of the trained classifier on the validation set.
- Select the value of M with the smallest error rate; say this is $M = 2$.
- Estimate the error rate for $M = 2$ on the test set.

For prediction on new data, you now use the tree classifier with $M = 2$, and report its estimated error rate as that estimated on the test set.

Meaning

- The quality measure by which we are comparing different classifiers f_γ (for different parameter values γ) is the risk

$$R\big(f_\gamma\big) = \mathbb{E}[L(y, f_\gamma(x))] \ .$$

- Since we do not know the true risk, we estimate it from data as $\hat{R}(f_\gamma)$.

Importance of model assessment step

- We always have to assume: Classifier is better adapted to *any* data used to select it than to actual data distribution.
- Model selection: Adapts classifier to *both* training and validation data.
- If we estimate error rate on any part of the training or validation data, we will in general underestimate it.

Procedure in detail

We consider possible parameter values $\gamma_1, \ldots, \gamma_m$.

1. For each value γ_j, train a classifier f_{γ_j} on the training set. (That is: γ_j is fixed, and the training algorithm outputs a fitted classifier f for this value of γ_j.)
2. Use the validation set to estimate $R(f_{\gamma_j})$ as the empirical risk

$$\hat{R}(f_\gamma) = \frac{1}{n_v} \sum_{i=1}^{n_v} L(\tilde{y}_i, f_{\gamma_j}(\tilde{\mathbf{x}}_i)) \ .$$

   n_v is the size of the validation set.
3. Select the value γ^* which achieves the smallest estimated error.
4. Re-train the classifier with parameter γ^* on all data except the test set (i.e. on training + validation data).
5. Report error estimate $\hat{R}(f_{\gamma^*})$ computed on the *test* set.

Idea

Each of the error estimates computed on validation set is computed from a single example of a trained classifier. Can we improve the estimate?

Strategy

- Set aside the test set.
- Split the remaining data into K blocks.
- Use each block in turn as validation set. Perform cross validation and average the results over all K combinations.

This method is called **K-fold cross validation**.

Example: K=5

Idea

Each of the error estimates computed on validation set is computed from a single example of a trained classifier. Can we improve the estimate?

Strategy

- Set aside the test set.
- Split the remaining data into K blocks.
- Use each block in turn as validation set. Perform cross validation and average the results over all K combinations.

This method is called **K-fold cross validation**.

Example: K=5

Step $k = 1$

Idea

Each of the error estimates computed on validation set is computed from a single example of a trained classifier. Can we improve the estimate?

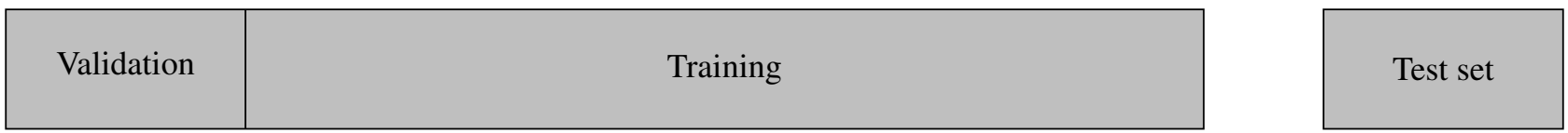
Strategy

- Set aside the test set.
- Split the remaining data into K blocks.
- Use each block in turn as validation set. Perform cross validation and average the results over all K combinations.

This method is called **K-fold cross validation**.

Example: K=5

Step $k = 2$



Idea

Each of the error estimates computed on validation set is computed from a single example of a trained classifier. Can we improve the estimate?

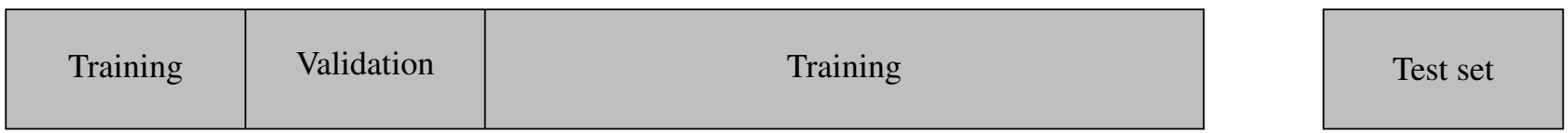
Strategy

- Set aside the test set.
- Split the remaining data into K blocks.
- Use each block in turn as validation set. Perform cross validation and average the results over all K combinations.

This method is called **K-fold cross validation**.

Example: K=5

Step $k = 3$



Idea

Each of the error estimates computed on validation set is computed from a single example of a trained classifier. Can we improve the estimate?

Strategy

- Set aside the test set.
- Split the remaining data into K blocks.
- Use each block in turn as validation set. Perform cross validation and average the results over all K combinations.

This method is called **K-fold cross validation**.

Example: K=5

Step $k = 4$

Idea

Each of the error estimates computed on validation set is computed from a single example of a trained classifier. Can we improve the estimate?

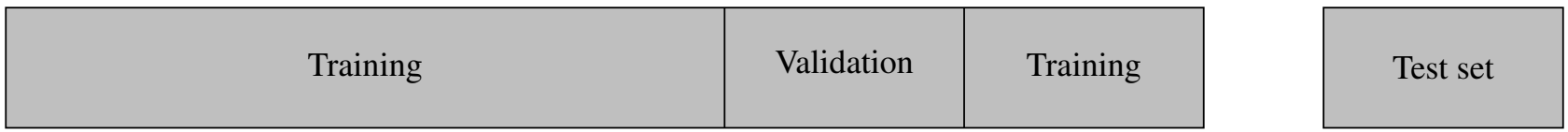
Strategy

- Set aside the test set.
- Split the remaining data into K blocks.
- Use each block in turn as validation set. Perform cross validation and average the results over all K combinations.

This method is called **K-fold cross validation**.

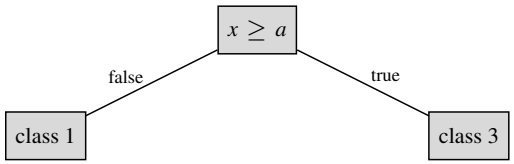
Example: K=5

Step $k = 5$

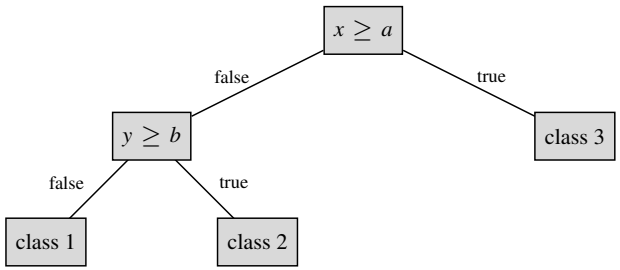


K-FOLD CV FOR A TREE

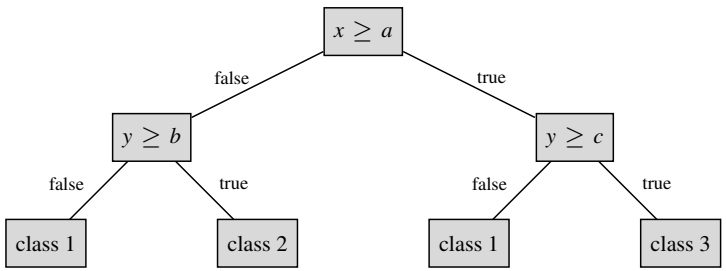
$M = 1$:



$M = 2$:



$M = 3$:

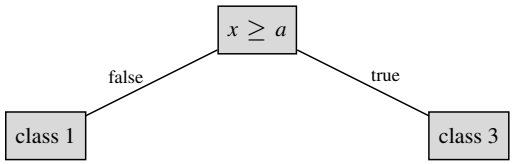


CV procedure (for $K = 5$)

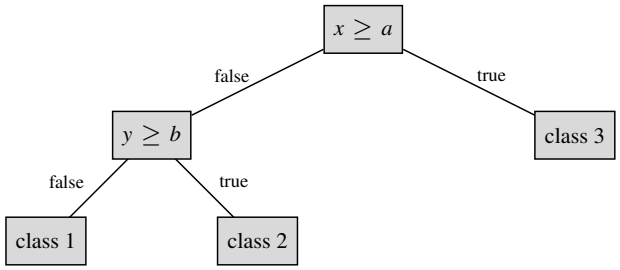
- Split off test data.
- Split remaining data into K equal parts.
- For each $k = 1, \ldots, 5$:
  1. Use kth block as validation set.
  2. For each M in $\{1, 2, 3\}$: Train a tree classifier with M splits on the remaining blocks.
  3. For each M, estimate the error rate of the trained classifier on the validation block.
- For each M, average the K error rate estimates over all values of k.
- Select the value of M with the smallest average error rate.
- Estimate the error rate for the optimal M on the test set.

K-FOLD CV FOR A TREE

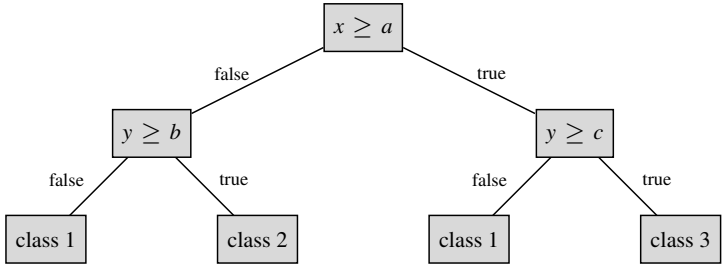
$M = 1$:

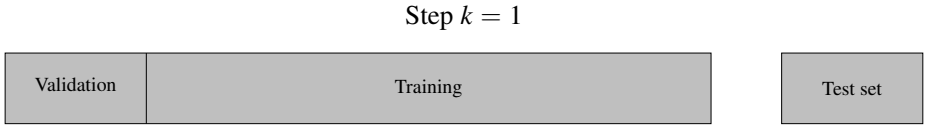


$M = 2$:



$M = 3$:



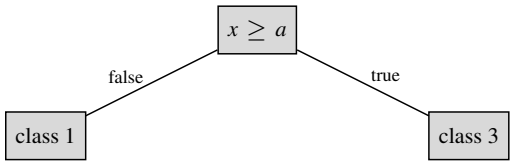


CV procedure (for $K = 5$)

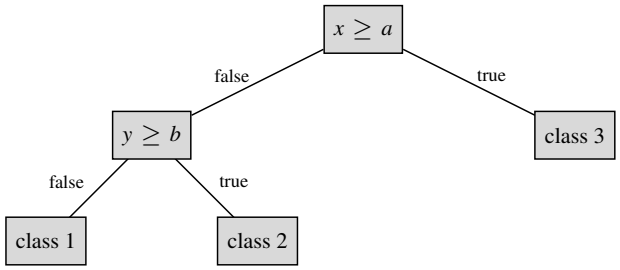
- Split off test data.
- Split remaining data into K equal parts.
- For each $k = 1, \ldots, 5$:
  1. Use kth block as validation set.
  2. For each M in $\{1, 2, 3\}$: Train a tree classifier with M splits on the remaining blocks.
  3. For each M, estimate the error rate of the trained classifier on the validation block.
- For each M, average the K error rate estimates over all values of k.
- Select the value of M with the smallest average error rate.
- Estimate the error rate for the optimal M on the test set.

K-FOLD CV FOR A TREE

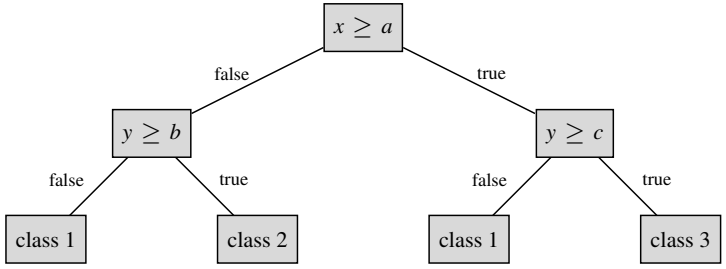
$M = 1$:



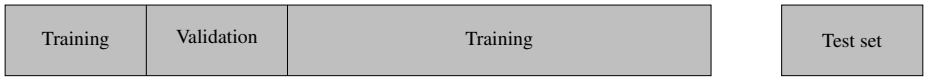
$M = 2$:



$M = 3$:



Step $k = 2$

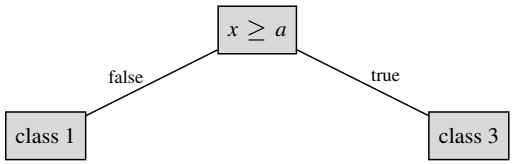


CV procedure (for $K = 5$)

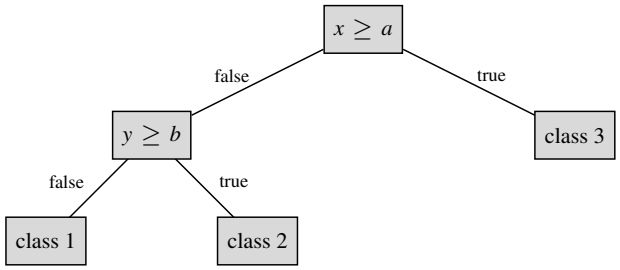
- Split off test data.
- Split remaining data into K equal parts.
- For each $k = 1, \ldots, 5$:
  1. Use kth block as validation set.
  2. For each M in $\{1, 2, 3\}$: Train a tree classifier with M splits on the remaining blocks.
  3. For each M, estimate the error rate of the trained classifier on the validation block.
- For each M, average the K error rate estimates over all values of k.
- Select the value of M with the smallest average error rate.
- Estimate the error rate for the optimal M on the test set.

K-FOLD CV FOR A TREE

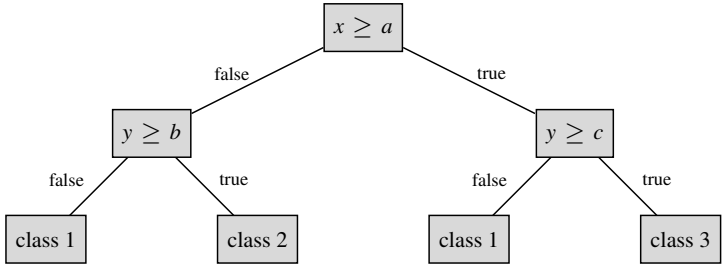
$M = 1$:

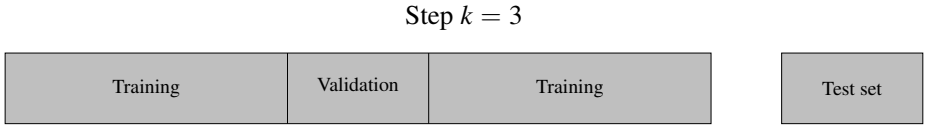


$M = 2$:



$M = 3$:



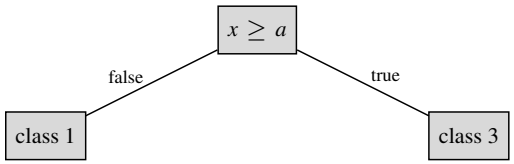


CV procedure (for $K = 5$)

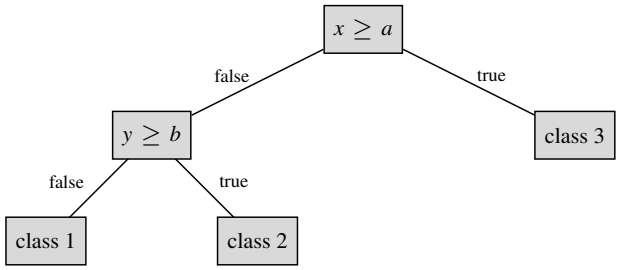
- Split off test data.
- Split remaining data into K equal parts.
- For each $k = 1, \ldots, 5$:
  1. Use kth block as validation set.
  2. For each M in $\{1, 2, 3\}$: Train a tree classifier with M splits on the remaining blocks.
  3. For each M, estimate the error rate of the trained classifier on the validation block.
- For each M, average the K error rate estimates over all values of k.
- Select the value of M with the smallest average error rate.
- Estimate the error rate for the optimal M on the test set.

K-FOLD CV FOR A TREE

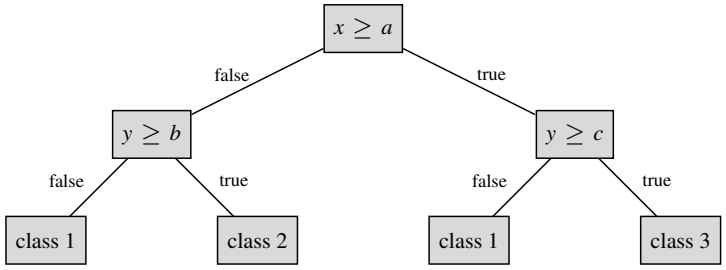
$M = 1$:



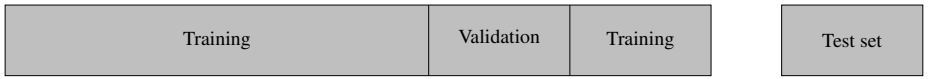
$M = 2$:



$M = 3$:



Step $k = 4$

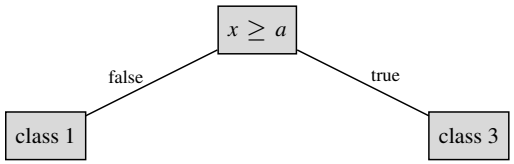


CV procedure (for $K = 5$)

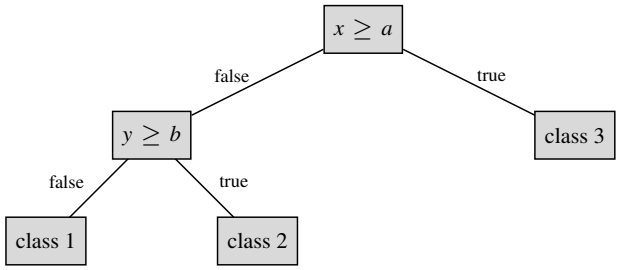
- Split off test data.
- Split remaining data into K equal parts.
- For each $k = 1, \ldots, 5$:
  1. Use kth block as validation set.
  2. For each M in $\{1, 2, 3\}$: Train a tree classifier with M splits on the remaining blocks.
  3. For each M, estimate the error rate of the trained classifier on the validation block.
- For each M, average the K error rate estimates over all values of k.
- Select the value of M with the smallest average error rate.
- Estimate the error rate for the optimal M on the test set.

K-FOLD CV FOR A TREE

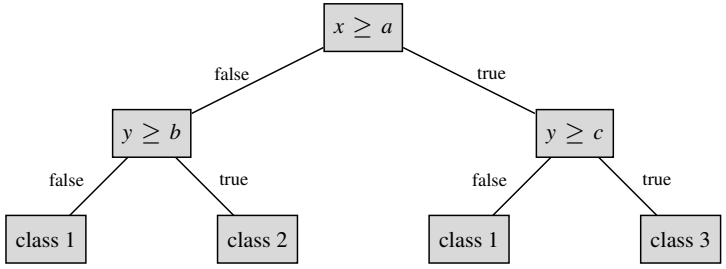
$M = 1$:



$M = 2$:



$M = 3$:



Step $k = 5$

CV procedure (for $K = 5$)

- Split off test data.
- Split remaining data into K equal parts.
- For each $k = 1, \ldots, 5$:
  1. Use kth block as validation set.
  2. For each M in $\{1, 2, 3\}$: Train a tree classifier with M splits on the remaining blocks.
  3. For each M, estimate the error rate of the trained classifier on the validation block.
- For each M, average the K error rate estimates over all values of k.
- Select the value of M with the smallest average error rate.
- Estimate the error rate for the optimal M on the test set.

Risk estimation

To estimate the risk of a classifier $f(\,.\,,\gamma_j)$:

1. Split data into K equally sized blocks.
2. Train an instance $f_{\gamma_j,k}$ of the classifier, using all blocks except block k as training data.
3. Compute the cross validation estimate

$$\hat{R}_{\text{CV}}(f_{\gamma_j}) := \frac{1}{K}\sum_{k=1}^{K}\frac{1}{|\text{block } k|}\sum_{(\tilde{\mathbf{x}},\tilde{y})\,\in\,\text{block } k} L\big(\tilde{y},f_{\gamma_j,k}(\tilde{\mathbf{x}})\big)$$

Repeat this for all parameter values $\gamma_1,\ldots,\gamma_m$.

Selecting a model

- Choose the parameter value γ^* for which estimated risk is minimal.

Model assessment

- Report risk estimate for f_{γ^*} computed on *test* data.

Extremal cases

- $K = n$, called **leave one out cross validation** (loocv)
- $K = 2$

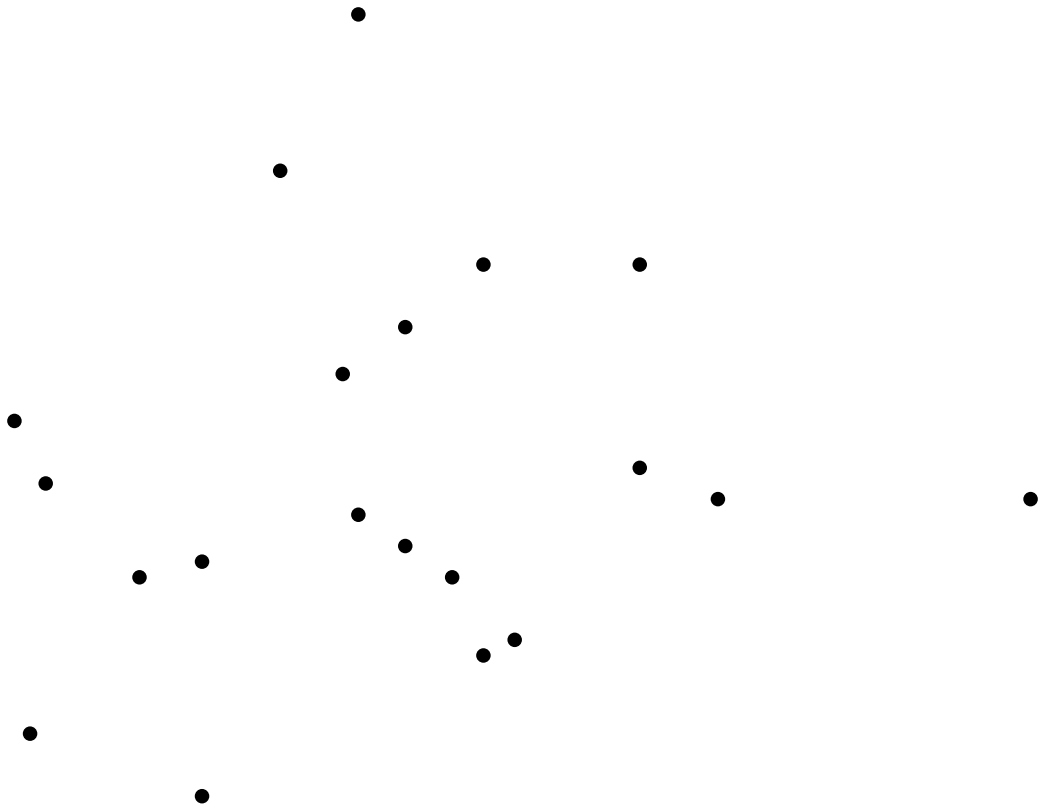
An often-cited problem with loocv is that we have to train many (= n) classifiers, but there is also a deeper problem.

Argument 1: K should be small, e.g. $K = 2$

- Unless we have a lot of data, variance between two distinct training sets may be considerable.
- **Important concept:** By removing substantial parts of the sample in turn and at random, we can simulate this variance.
- By removing a single point (loocv), we cannot make this variance visible.

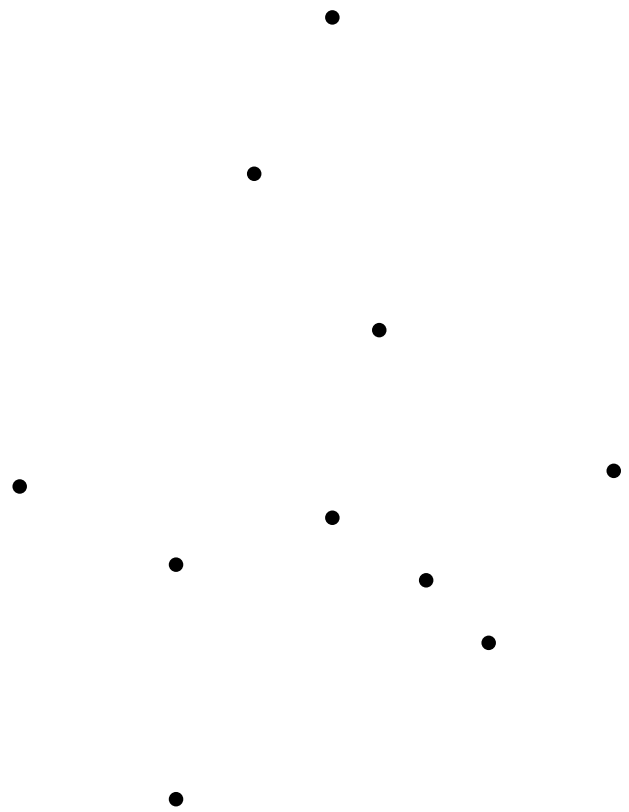
Illustration

$K = 2, n = 20$



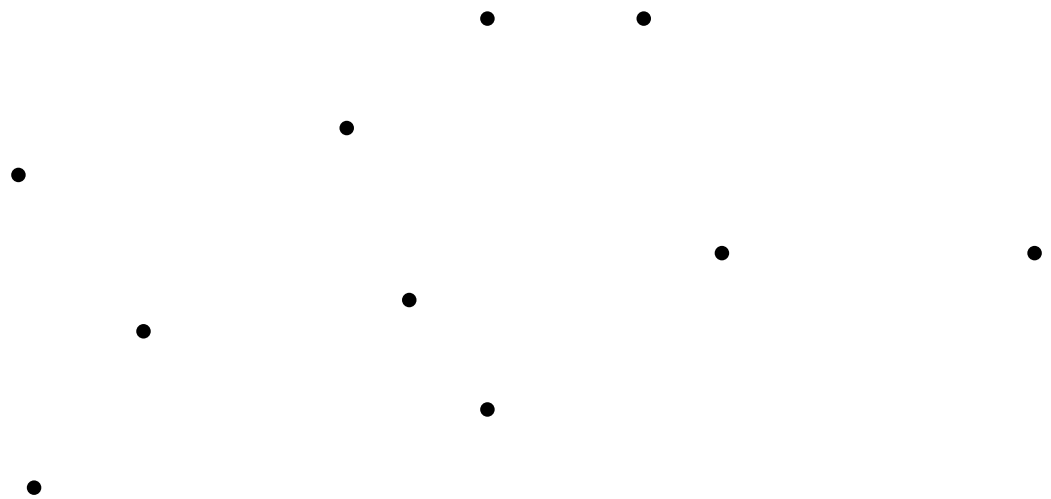
Illustration

$K = 2, n = 20$



Illustration

$K = 2, n = 20$



How to Choose K?

Argument 2: K should be large, e.g. $K = n$

- Classifiers generally perform better when trained on larger data sets.
- A small K means we substantially reduce the amount of training data used to train each f_k, so we may end up with weaker classifiers.
- This way, we will systematically overestimate the risk.

Common recommendation: $K = 5$ to $K = 10$

Intuition:

- $K = 10$ means number of samples removed from training is one order of magnitude below training sample size.
- This should not weaken the classifier considerably, but should be large enough to make measure variance effects.

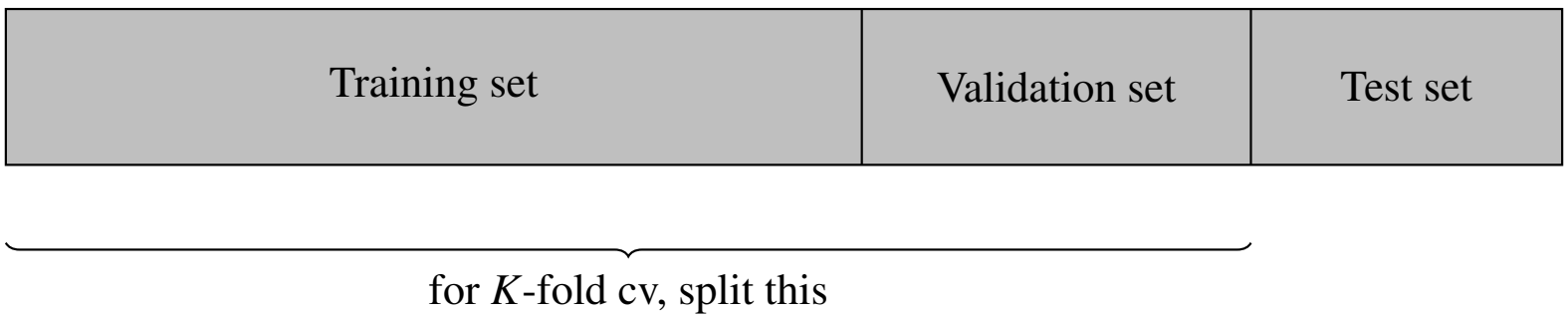
SUMMARY: CROSS VALIDATION

Purpose

Estimates the risk $R(f) = \mathbb{E}[L(y, f(x))]$ of a classifier (or regression function) from data.

Application to parameter tuning

- Compute one cross validation estimate of $R(f)$ for each parameter value.
- Note again: Cross validation procedure does not involve the test data.



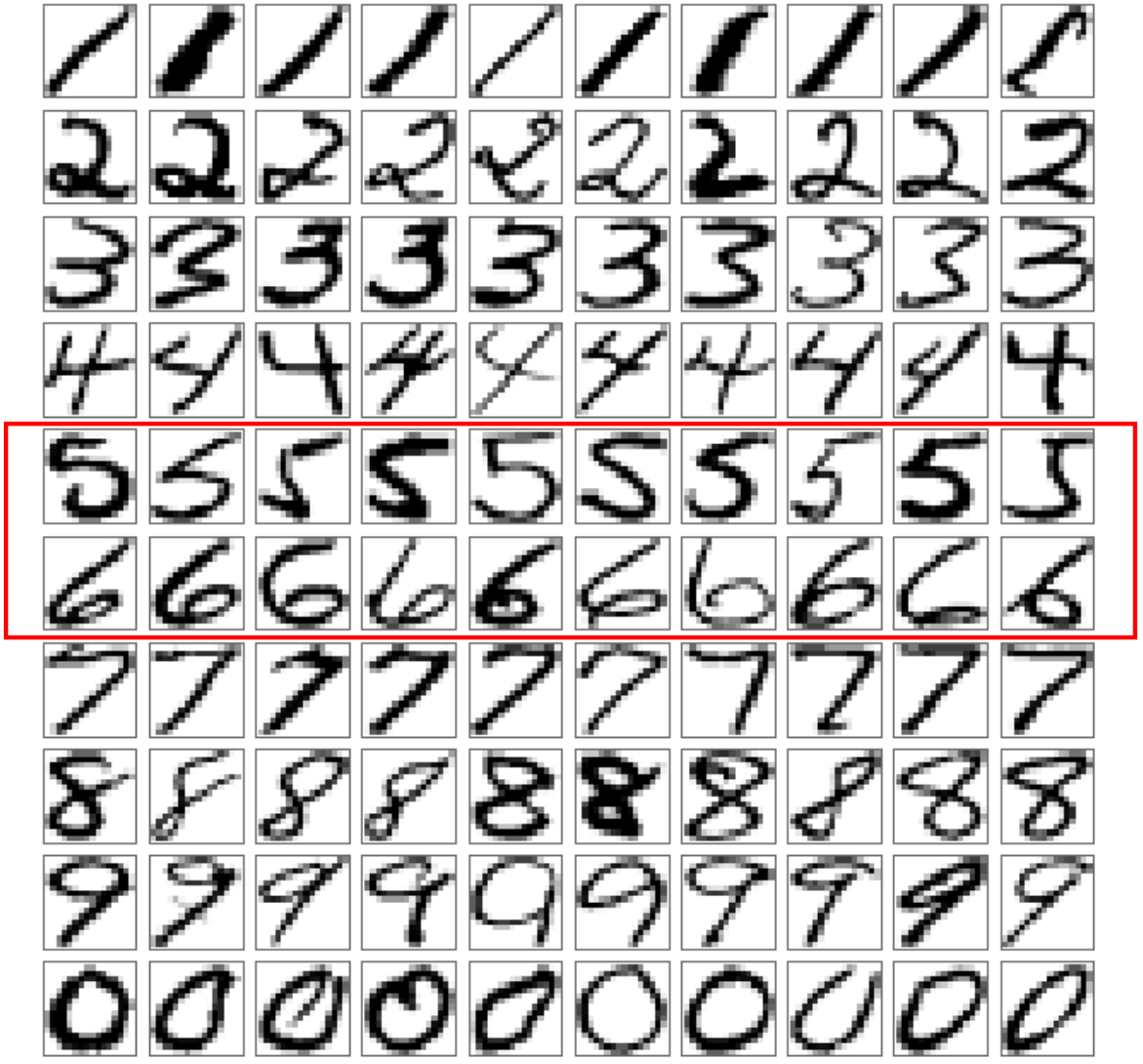
CV for a m-NN classifier

- The nearest neighbor classifier is particularly simple since it does not require a training *algorithm*.
- Since it uses training *data*, we still need to distinguish between training, validation and test data.

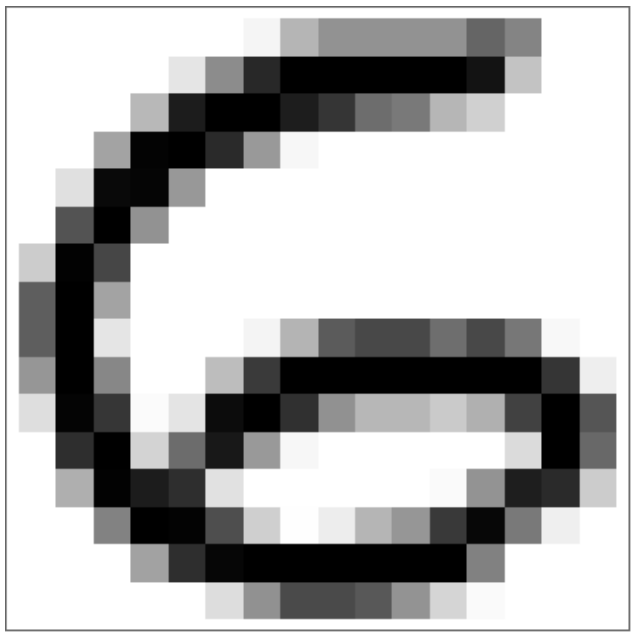
Task

- We want to classify handwritten digits (according to the value of the digit).
- We use an m-nearest neighbor classifer.
- That means we have to choose a suitable value for m.

Digit Classification: Data



Digit Classification: Data



$=$

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 10 & 74 & 109 & 109 & 109 & 109 & 154 & 123 & 0 & 0 \\
0 & 0 & 0 & 0 & 26 & 115 & 215 & 255 & 255 & 255 & 255 & 255 & 236 & 60 & 0 & 0 \\
0 & 0 & 0 & 71 & 227 & 255 & 255 & 226 & 202 & 146 & 134 & 73 & 47 & 0 & 0 & 0 \\
0 & 0 & 92 & 252 & 255 & 213 & 102 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 31 & 246 & 250 & 103 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 172 & 255 & 109 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
51 & 253 & 185 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
161 & 255 & 92 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
161 & 255 & 26 & 0 & 0 & 0 & 11 & 75 & 164 & 183 & 183 & 145 & 183 & 136 & 7 & 0 \\
105 & 255 & 120 & 0 & 0 & 66 & 197 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 202 & 17 \\
33 & 251 & 201 & 4 & 27 & 243 & 255 & 207 & 110 & 72 & 72 & 53 & 79 & 190 & 255 & 170 \\
0 & 209 & 255 & 44 & 147 & 231 & 102 & 8 & 0 & 0 & 0 & 0 & 0 & 36 & 255 & 151 \\
0 & 80 & 253 & 227 & 209 & 30 & 0 & 0 & 0 & 0 & 0 & 5 & 108 & 225 & 213 & 51 \\
0 & 0 & 125 & 255 & 252 & 177 & 48 & 1 & 18 & 74 & 105 & 198 & 248 & 134 & 16 & 0 \\
0 & 0 & 0 & 93 & 209 & 249 & 255 & 255 & 255 & 255 & 255 & 255 & 126 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 34 & 110 & 181 & 181 & 167 & 108 & 42 & 5 & 0 & 0 & 0
\end{pmatrix}$$

- Grayscale values $0, \ldots, 255$ (where 0 means white and 255 black).
- Each matrix is "rolled off" into a vector. These vectors are collected in a matrix.
- Each image is of size $16 \times 16 = 256$ pixels, so we obtain vectors in $\mathbb{R}^{256}$.

Basic cross validation for m-NN

- Choose candidate values for m, say $m = 1, 3, 5$.
- Split data into training, validation and test set.
- For each $m \in \{1, 3, 5\}$, implement the m-NN classifier f_m. Each of these uses the training data set to classify.
- Compute the misclassification rate for each m on the validation set.
- Choose the m with the smallest misclassification rate.
- Compute the misclassifation rate on the test set for the optimal m.

Are we allowed to use the validation set for prediction?

Measuring Classifier Performance

Error types in a two-class problem

- **False positives** (type I error): True label is -1, predicted label is +1.
- **False negative** (type II error): True label is +1, predicted label is -1.

We write TP = # true positives, FP = # false positives, TN = # true negatives, FN = # false negatives

Error rate

$$\text{ER} = \frac{\text{\# wrong predictions}}{\text{\# observations}} = \frac{\text{FP} + \text{FN}}{\text{FP} + \text{FN} + \text{TP} + \text{TN}}$$

Does not distinguish errors between classes.

Relevance

Distinction between error types is crucial e.g. if:

- Classes differ significantly in size
- One type of error has worse consequences than other

MATRIX REPRESENTATION

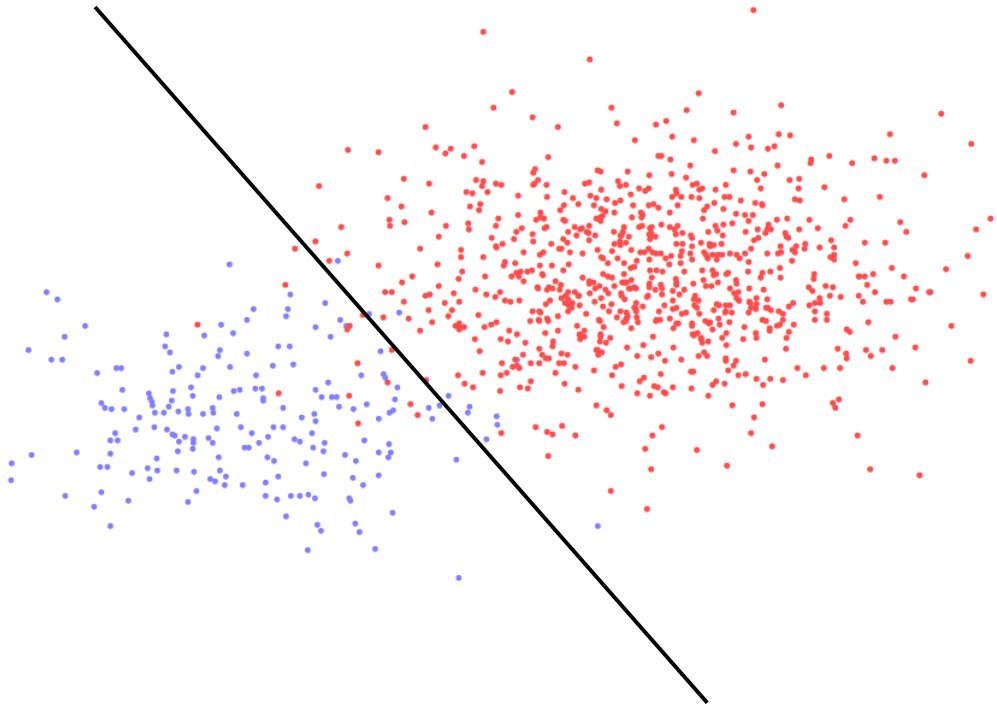
The different types of errors can be summarized in a matrix as

	positive label	negative label
predicted positive	TP/n	FP/n
predicted negative	FN/n	TN/n

where n is the number of observations.

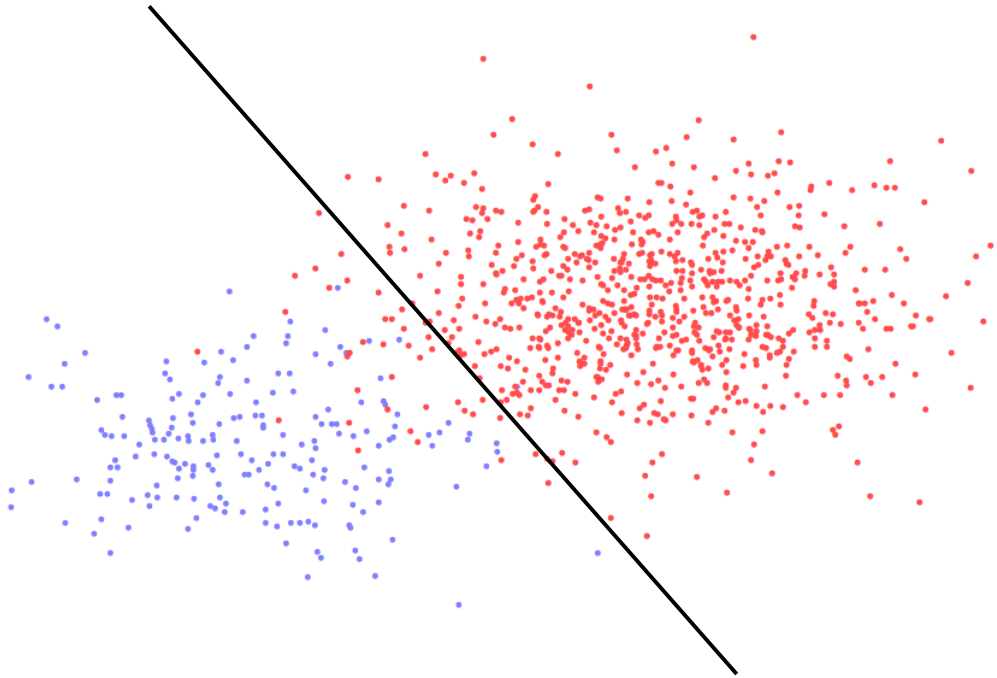
This is called a **confusion matrix** or **contingency table**.

Dependence on Parameters



- Suppose a classifier is determined by some parameter θ.
- As we change θ, the number of false positives and false negatives changes.
- We hence have parameter-dependent quantities $\mathrm{TP}(\theta)$, $\mathrm{TN}(\theta)$, etc.

Dependence on Parameters



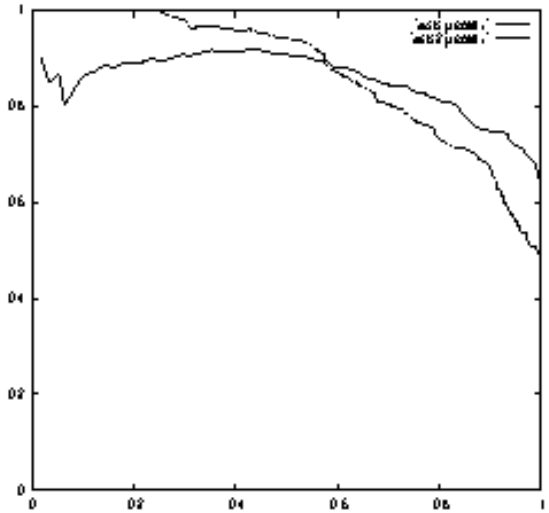
- Suppose a classifier is determined by some parameter θ.
- As we change θ, the number of false positives and false negatives changes.
- We hence have parameter-dependent quantities $\mathrm{TP}(\theta)$, $\mathrm{TN}(\theta)$, etc.

Precision and Recall

One summary measure of classifier performance are precision and recall:

$$\textbf{Precision}(\theta) := \frac{\text{TP}(\theta)}{\text{TP}(\theta) + \text{FP}(\theta)} \qquad \textbf{Recall}(\theta) := \frac{\text{TP}(\theta)}{\text{TP}(\theta) + \text{FN}(\theta)}$$

A **precision/recall plot** eveluates precision and recall on validation/test data for a range of different values of θ, and plots precision (vertical axis) against recall (horizontal axis):

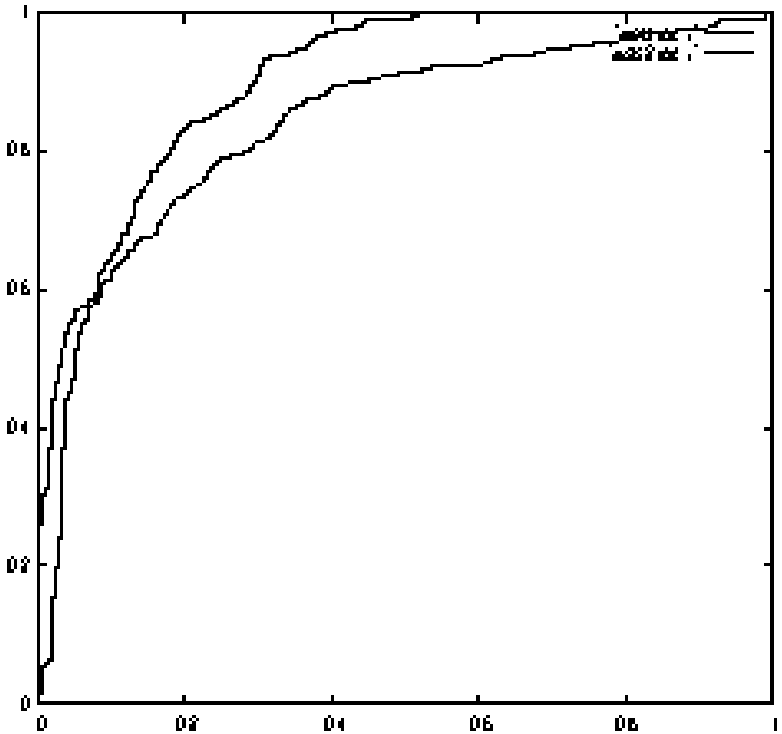


- Each point in the plot represents a classifier, for one value of θ.
- Ideally, both precision and recall are high, so "good values" are in the upper right corner.

ROC Diagrams

A plot of the *true positive rate* (TPR) versus the *false positive rate* (FPR) is called a **receiver operating characteristic** (ROC) curve:

$$\text{TPR} = \frac{\text{TP}}{\text{\# Positives}} \qquad \text{FPR} = \frac{\text{FP}}{\text{\# Negatives}}$$



- "Good" region: Upper left corner. (P/R: Upper *right* corner.)
- Classifier below diagonal (lower left to upper right): Worse than random decision.

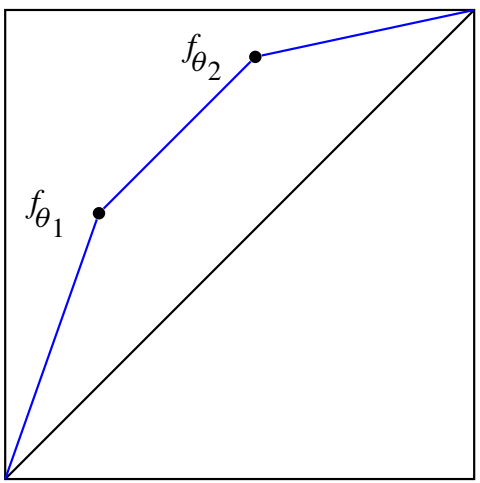
Linear interpolation of classifiers

- Given: Classifiers $f_{\theta_1}, f_{\theta_2}$, interpolation parameter $\lambda \in [0, 1]$.
- Define new classifier f_λ as: Randomly choose output of f_{θ_1} with probability λ, output of f_{θ_2} with probability $1 - \lambda$.

Error rates under interpolation

$$\text{TPR}(f_\lambda) = \lambda \text{TPR}(f_{\theta_1}) + (1 - \lambda)\text{TPR}(f_{\theta_2})$$

The same holds for FPR, ER (but *not* for Precision and Recall).



- ROC plot: Every point represents a classifier performance.
- Consequence: A classifier with performance represented by a point on a straight line between f_{θ_1} and f_{θ_2} in the plot can be constructed by linear interpolation.
- The perfomance values constructable from existing classifiers in this way are called *achievable*.

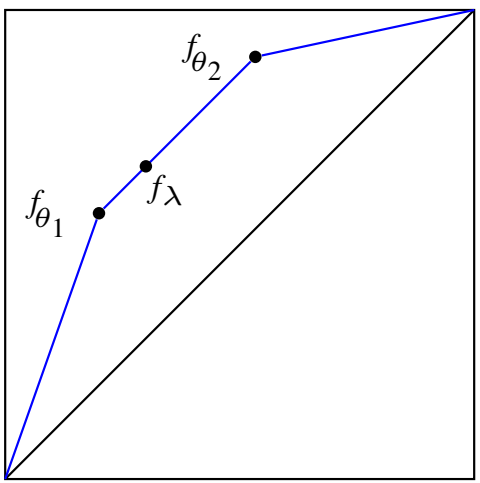
Linear interpolation of classifiers

- Given: Classifiers $f_{\theta_1}, f_{\theta_2}$, interpolation parameter $\lambda \in [0, 1]$.
- Define new classifier f_λ as: Randomly choose output of f_{θ_1} with probability λ, output of f_{θ_2} with probability $1 - \lambda$.

Error rates under interpolation

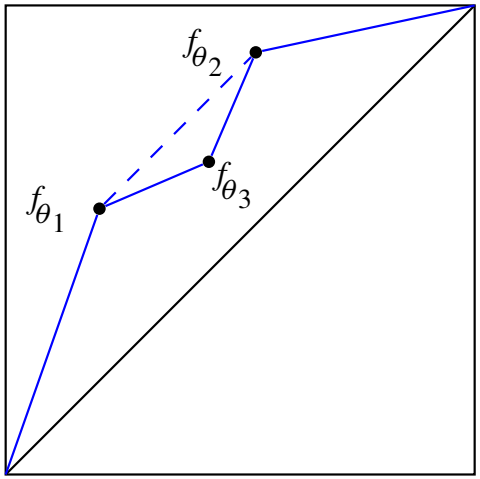
$$\text{TPR}(f_\lambda) = \lambda \text{TPR}(f_{\theta_1}) + (1 - \lambda)\text{TPR}(f_{\theta_2})$$

The same holds for FPR, ER (but *not* for Precision and Recall).



- ROC plot: Every point represents a classifier performance.
- Consequence: A classifier with performance represented by a point on a straight line between f_{θ_1} and f_{θ_2} in the plot can be constructed by linear interpolation.
- The perfomance values constructable from existing classifiers in this way are called *achievable*.

ROC Interpolation: Convex Hull



- Suppose classifiers $f_{\theta_1}, f_{\theta_2}, f_{\theta_3}$ are given:
- If the objective is to optimize ROC performance, f_{θ_3} is worthless.
- We can always obtain a better classifiers by interpolating f_{θ_1} and f_{θ_2}.

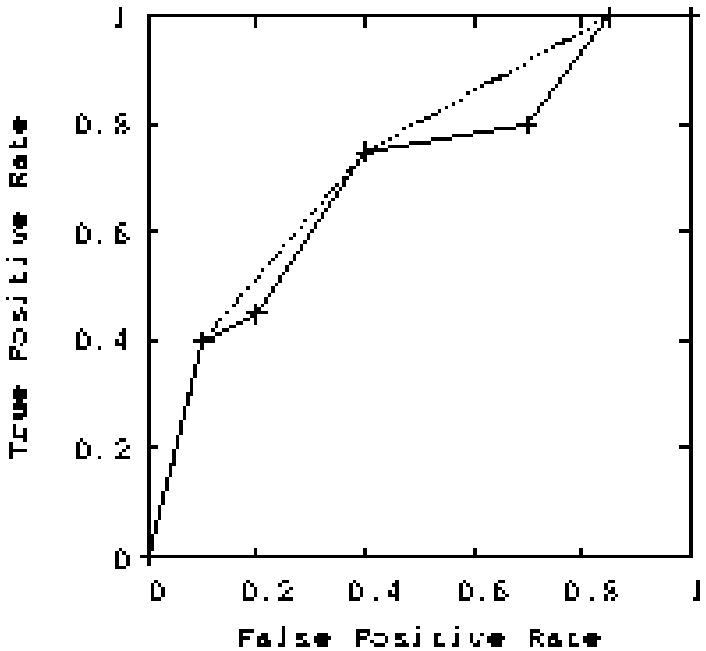
In general

- Recall the interpolation formula $\lambda \text{TPR}(f_{\theta_1}) + (1-\lambda)\text{TPR}(f_{\theta_2})$ is a convex combination.
- If $\{f_{\theta_1}, \ldots, f_{\theta_k}\}$ are given: Any convex combination of these is achievable.

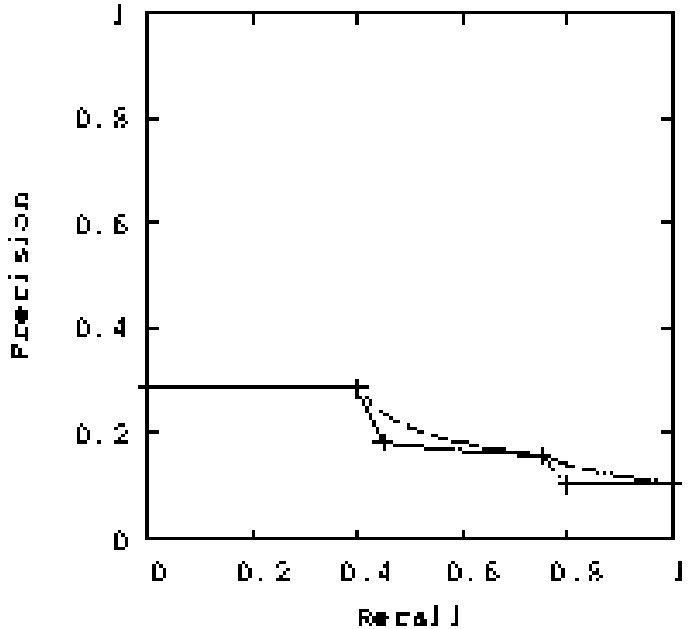
For given classifiers $\{f_{\theta_1}, \ldots, f_{\theta_k}\}$, the convex hull of these classifiers in the ROC plot is achievable.

ROC vs Precision/Recall

In Precision/Recall graphs, linear interpolation of classifiers does *not* correspond to linear interpolation of points in the plot.



ROC convex hull



Translation to P/R curve

Disadvantage of ROC

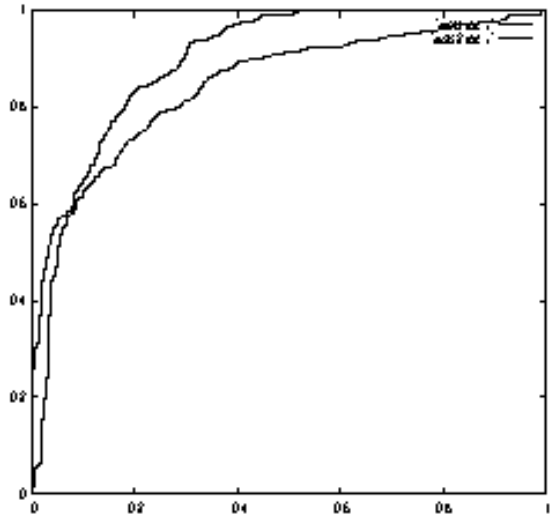
- If the TNR is high, any system can easily achive good FPR or ER by biasing towards the negative class.
- High TNR problems are typically those where one tries to pick out a few interesting points against a large background class (e.g. face detection).

Example

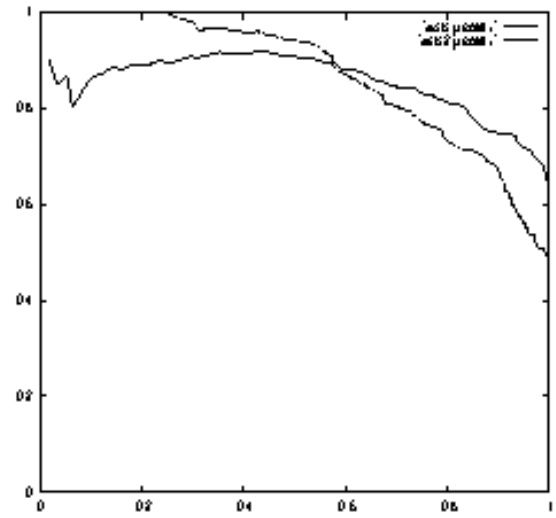
- Two classes are given. Increase the size of the negative class by a factor 10.
- The TP value of a given classifier and # Positives in training data do not depend on the negative class, so the TPR does not change.
- Since FP increases roughly by a factor ten, the FPR does not change either:

$$\mathrm{FPR}_{\mathrm{new}} \approx \frac{10 \cdot \mathrm{FP}_{\mathrm{old}}}{10 \cdot \#\,\mathrm{Negatives}_{\mathrm{old}}} = \mathrm{FPR}_{\mathrm{old}}$$

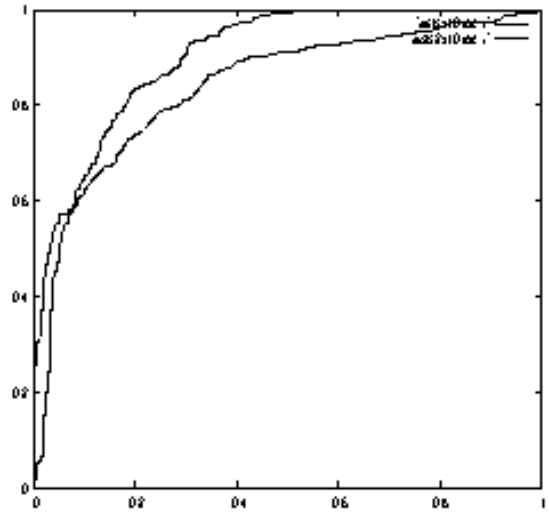
- Consequence: The ROC curve does not change, up to small fluctuations.



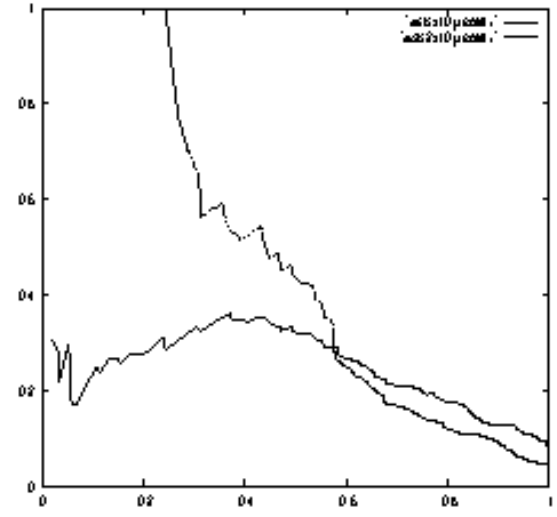
ROC (original classes)



P/R (original classes)



ROC (negative class $\times$ 10)



P/R (negative class $\times$ 10)

Parametrization by a threshold τ

- Many classifiers we have seen can be written as comparing a function g to a threshold τ.
- The classification result $f(\mathbf{x})$ is then computed as

$$f(\mathbf{x}) = \begin{cases} +1 & g(\mathbf{x}) \geq \tau \\ -1 & g(\mathbf{x}) < \tau \end{cases}$$

For example

f	$g(\mathbf{x})$	τ
linear classifier	$\langle \mathbf{v}, \mathbf{x} \rangle - c$	$\tau = 0$
logistic regression	$\sigma(\langle \mathbf{v}, \mathbf{x} \rangle - c)$	$\tau = \frac{1}{2}$
one gaussian density p per class	$p_{+1}(\mathbf{x}) - p_{-1}(\mathbf{x})$	$\tau = 0$

Varying τ

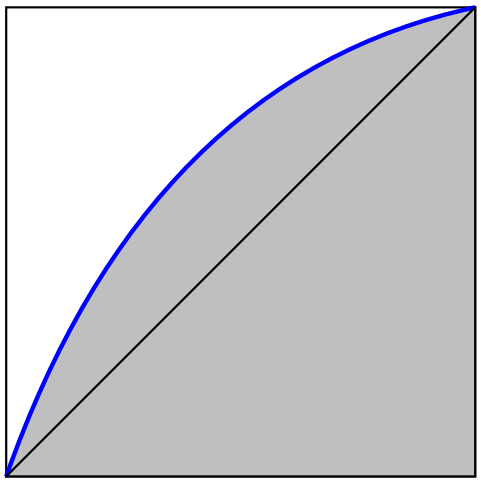
- We can denote the classifier f above as f_τ for a given value of τ, and vary that value.
- As τ changes, the values of TP, FN, etc change.
- For a larger value of τ, fewer points are classified as positive, so we expect fewer false positives and more false negatives.
- If we regard τ as the parameter θ above, we can draw a ROC curve or Precision/Recall diagram for f, where each point correspond to a value of τ.

If you see a ROC or P/R curve reported for a single classifier, this is usually what it means.

Definition

The **Area Under the Curve** (AUC) the area under an ROC curve. Note this is a value between 0 and 1.

Illustration

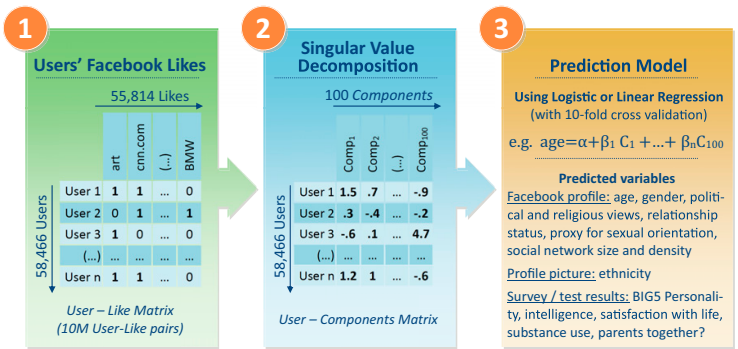


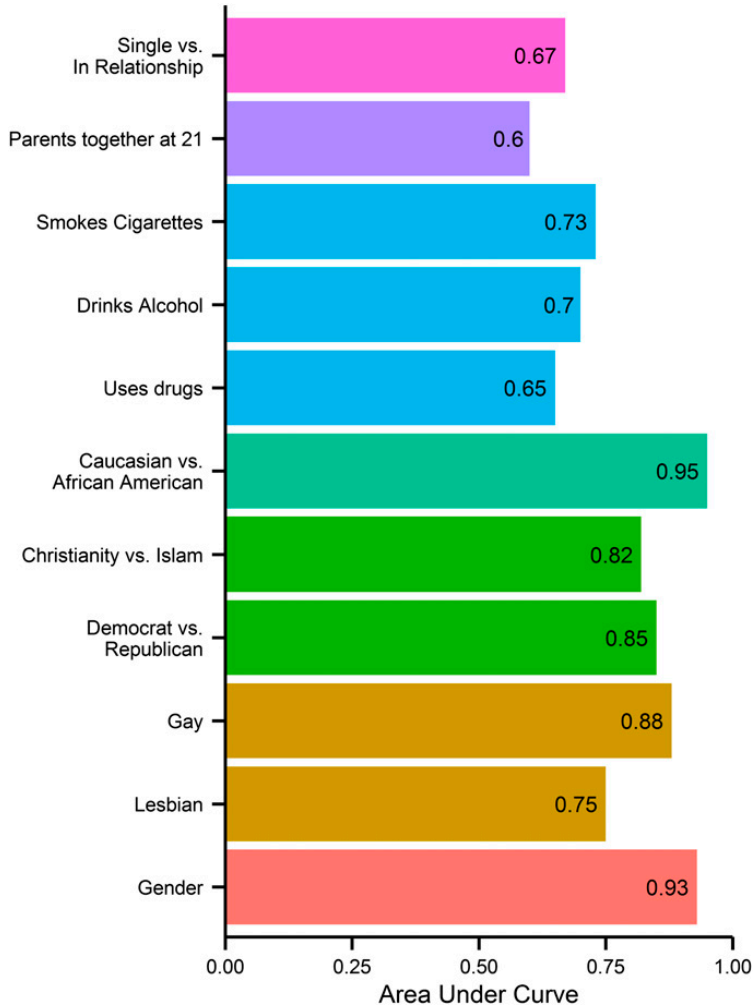
- The blue curve is an ROC curve.
- The AUC value is the size of the area shaded in gray.
- AUC is a summary statistic that summarizes a ROC diagram in a single number.

AUC of a classifier

When AUC is reported for a single classifier, it typically refers to the AUC defined by the ROC diagram obtained by varying a threshold τ as above.

EXAMPLE

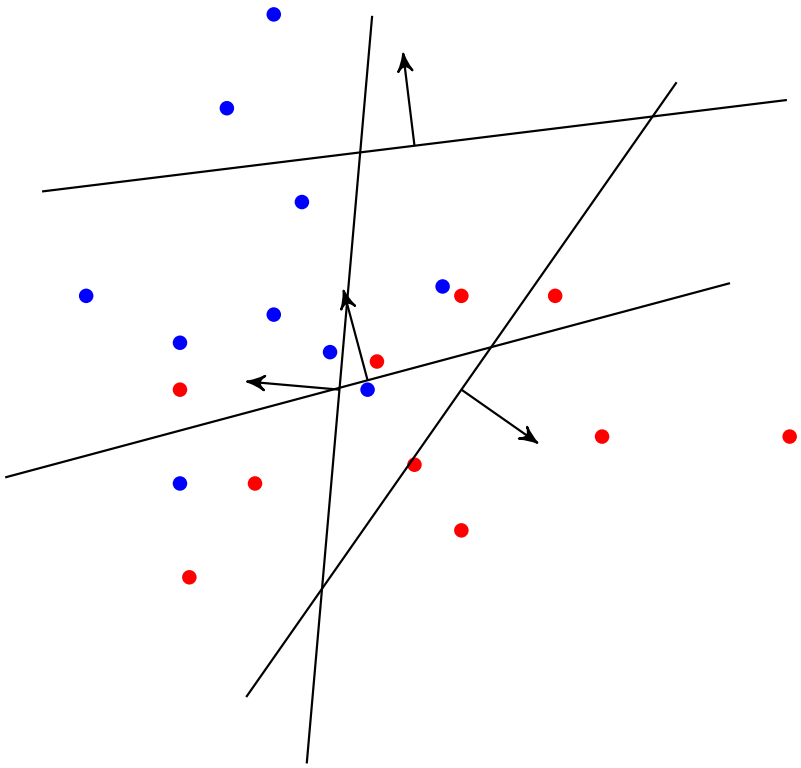




From: Kosinski, Stillwell, Graepel: PNAS, Vol 110, No 15, 2013.

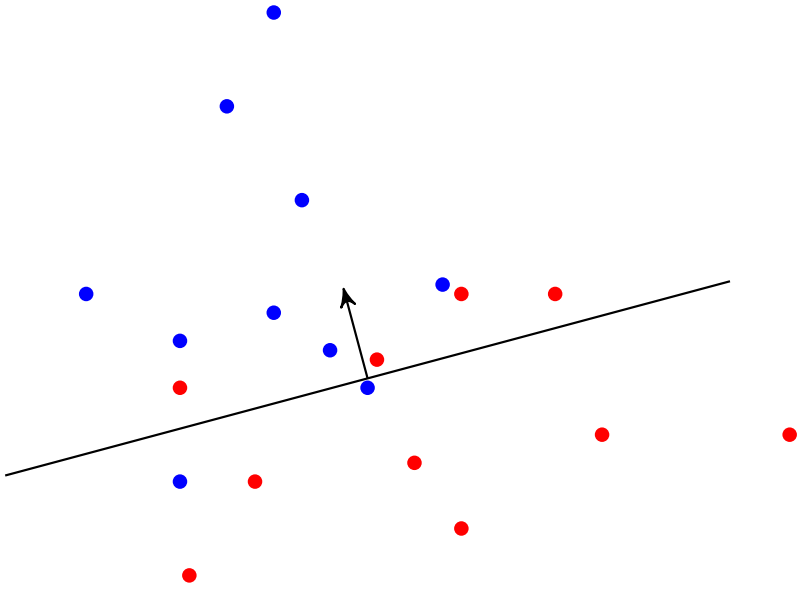
BOOSTING

Suppose we are given a data source with two classes, and manage to generate a *random* hyperplane classifier with *expected* error of 0.5 (i.e. 50%).

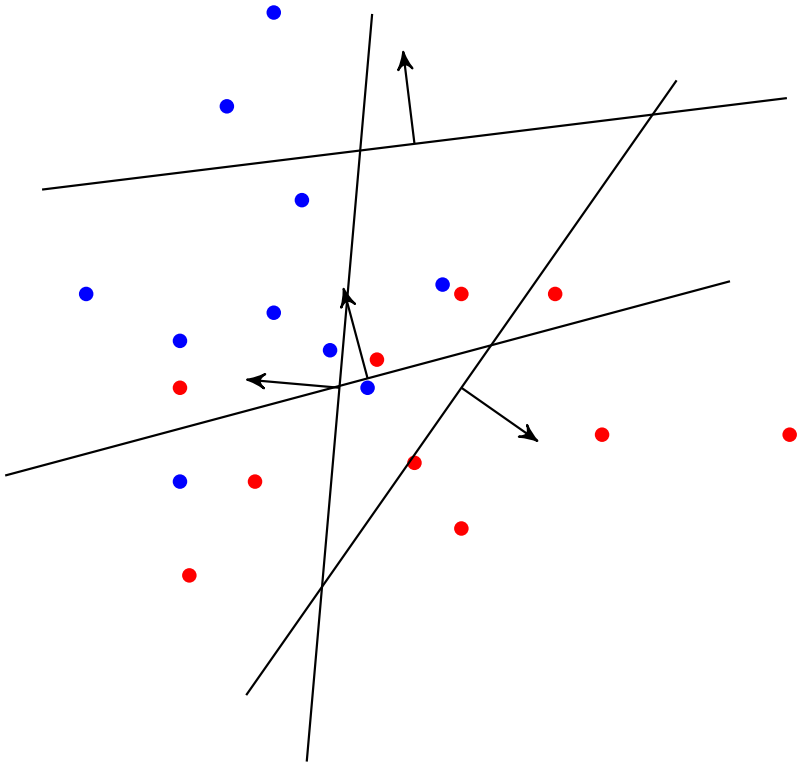


(Informally, think of this as not knowing the data source and generating a "uniformly distributed classifier".)

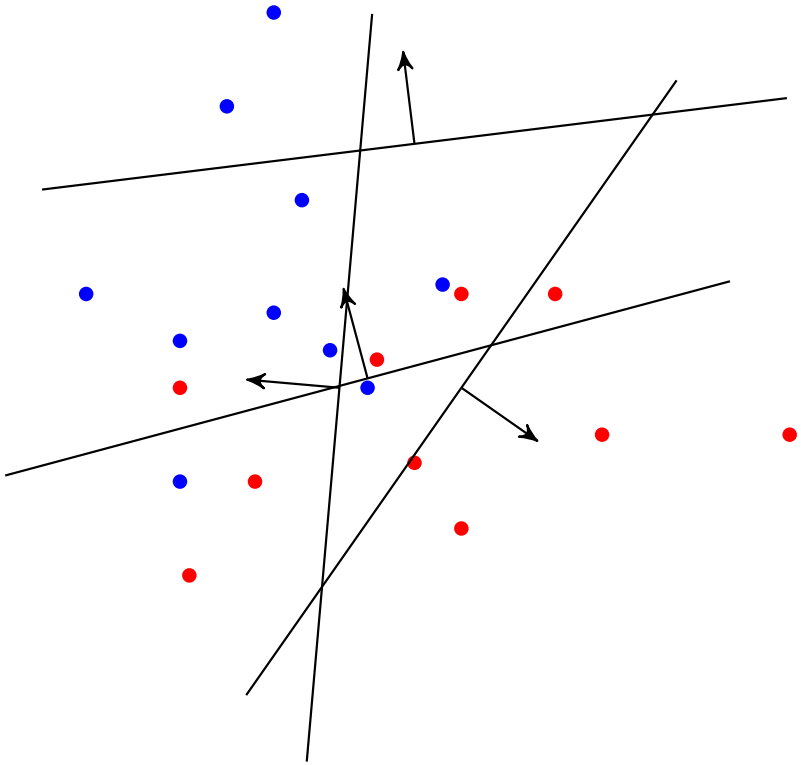
A *randomly* chosen hyperplane classifier has an *expected* error of 0.5 (i.e. 50%).



A *randomly* chosen hyperplane classifier has an *expected* error of 0.5 (i.e. 50%).



A *randomly* chosen hyperplane classifier has an *expected* error of 0.5 (i.e. 50%).



- Many random hyperplanes combined by majority vote: Still 0.5.
- A single classifier slightly better than random: $0.5 + \varepsilon$.
- What if we use m such classifiers and take a majority vote?

Decision by majority vote

- m individuals (or classifiers) take a vote. m is an odd number.
- They decide between two choices; one is correct, one is wrong.
- After everyone has voted, a decision is made by simple majority.

Note: For two-class classifiers $f_1, \ldots, f_m$ (with output ± 1):

$$\text{majority vote} = \operatorname{sgn}\Big(\sum_{j=1}^{m} f_j\Big)$$

Assumptions

Before we discuss ensembles, we try to convince ourselves that voting can be beneficial. We make some simplifying assumptions:

- Each individual makes the right choice with probability $p \in [0, 1]$.
- The votes are *independent*, i.e. stochastically independent when regarded as random outcomes.

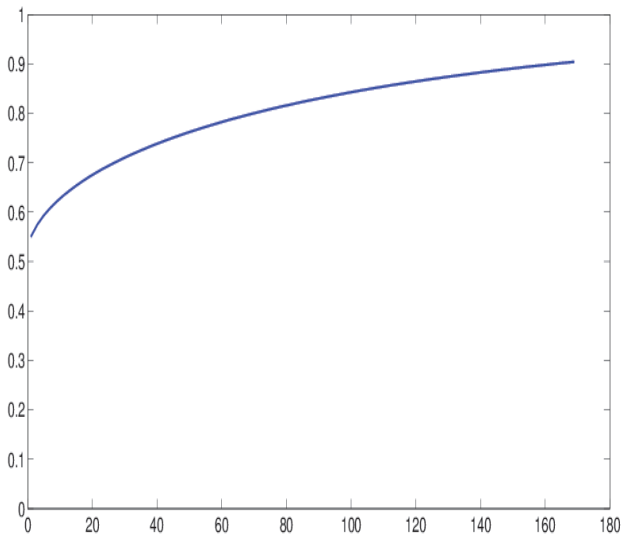
Condorcet's rule

If the individual votes are independent, the answer is

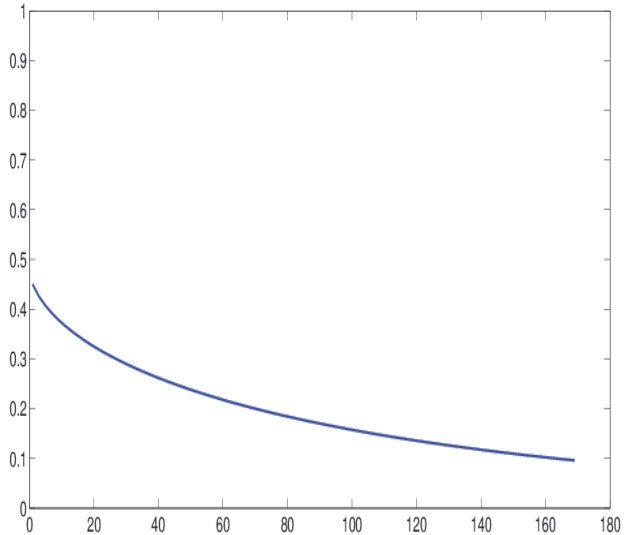
$$\Pr\{\text{ majority makes correct decision }\} = \sum_{j=\frac{m+1}{2}}^{m} \frac{m!}{j!(m-j)!} p^j (1-p)^{m-j}$$

This formula is known as **Condorcet's jury theorem**.

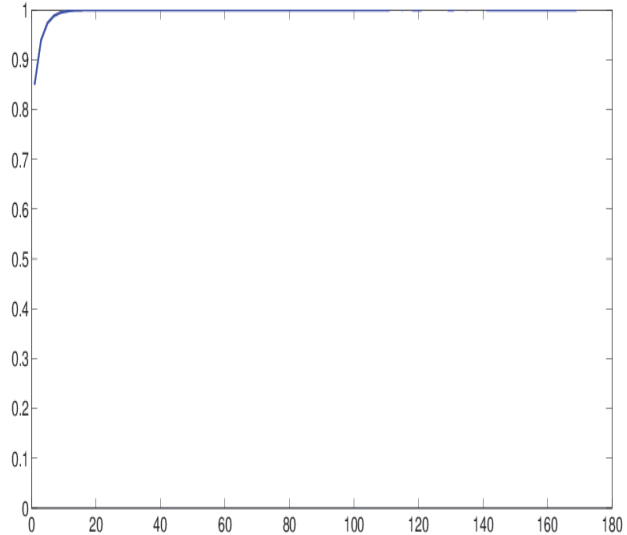
Probability as function of the number of votes



$p = 0.55$



$p = 0.45$



$p = 0.85$

Terminology

- An **ensemble method** makes a prediction by combining the predictions of many classifiers into a single vote.
- The individual classifiers are usually required to perform only slightly better than random. For two classes, this means slightly more than 50% of the data are classified correctly. Such a classifier is called a **weak learner**.

Strategy

- We have seen above that if the weak learners are random and independent, the prediction accuracy of the majority vote will increase with the number of weak learners.
- Since the weak learners all have to be trained on the training data, producing random, independent weak learners is difficult.
- Different ensemble methods (e.g. Boosting, Bagging, etc) use different strategies to train and combine weak learners that behave relatively independently.

Methods We Will Discuss

Boosting

- After training each weak learner, data is modified using weights.
- Deterministic algorithm.

Bagging

- Each weak learner is trained on a random subset of the data.

Random forests

- Bagging with tree classifiers as weak learners.
- Uses an additional step to remove dimensions in $\mathbb{R}^d$ that carry little information.

Boosting

- Arguably the most popular (and historically the first) ensemble method.
- Weak learners can be trees (decision stumps are popular), Perceptrons, etc.
- Requirement: It must be possible to train the weak learner on a *weighted* training set.

Overview

- Boosting adds weak learners one at a time.
- A weight value is assigned to each training point.
- At each step, data points which are currently classified correctly are weighted down (i.e. the weight is smaller the more of the weak learners already trained classify the point correctly).
- The next weak learner is trained on the *weighted* data set: In the training step, the error contributions of misclassified points are multiplied by the weights of the points.
- Roughly speaking, each weak learner tries to get those points right which are currently not classified correctly.

Example: Decision stump

A decision stump classifier for two classes is defined by

$$f(\,\mathbf{x}\,|\,j,t\,) := \begin{cases} +1 & x^{(j)} > t \\ -1 & \text{otherwise} \end{cases}$$

where $j \in \{1,\ldots,d\}$ indexes an axis in $\mathbb{R}^d$.

Weighted data

- Training data $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \ldots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$.
- With each data point $\tilde{\mathbf{x}}_i$ we associate a weight $w_i \geq 0$.

Training on weighted data

Minimize the *weighted* misclassifcation error:

$$(j^*, t^*) := \arg\min_{j,t} \frac{\sum_{i=1}^n w_i \mathbb{I}\{\tilde{y}_i \neq f(\tilde{\mathbf{x}}_i | j,t)\}}{\sum_{i=1}^n w_i}$$

Input

- Training data $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \ldots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$
- Algorithm parameter: Number M of weak learners

Training algorithm

1. Initialize the observation weights $w_i = \frac{1}{n}$ for $i = 1, 2, \ldots, n$.
2. For $m = 1$ to M:
   - 2.1 Fit a classifier $g_m(x)$ to the training data using weights w_i.
   - 2.2 Compute
   $$\text{err}_m := \frac{\sum_{i=1}^{n} w_i \mathbb{I}\{y_i \neq g_m(x_i)\}}{\sum_i w_i}$$
   - 2.3 Compute $\alpha_m = \log(\frac{1-\text{err}_m}{\text{err}_m})$
   - 2.4 Set $w_i \leftarrow w_i \cdot \exp(\alpha_m \cdot \mathbb{I}(y_i \neq g_m(x_i)))$ for $i = 1, 2, \ldots, n$.
3. Output
$$f(x) := \text{sign}\left(\sum_{m=1}^{M} \alpha_m g_m(x)\right)$$

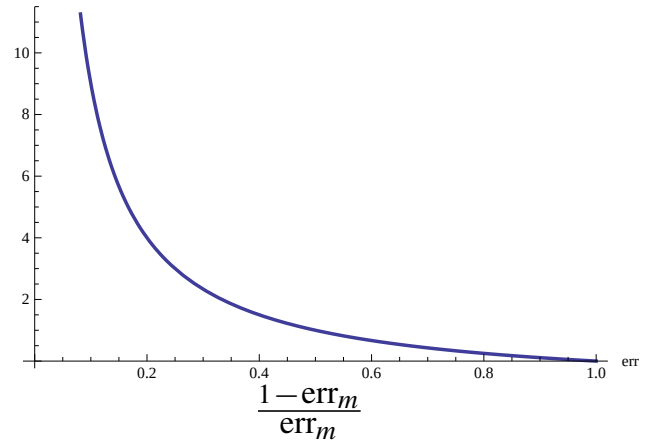
Weight updates

$$\alpha_m = \log\left(\frac{1 - \text{err}_m}{\text{err}_m}\right)$$

$$w_i^{(\text{m})} = w_i^{(\text{m-1})} \cdot \exp(\alpha_m \cdot \mathbb{I}(y_i \neq g_m(x_i)))$$

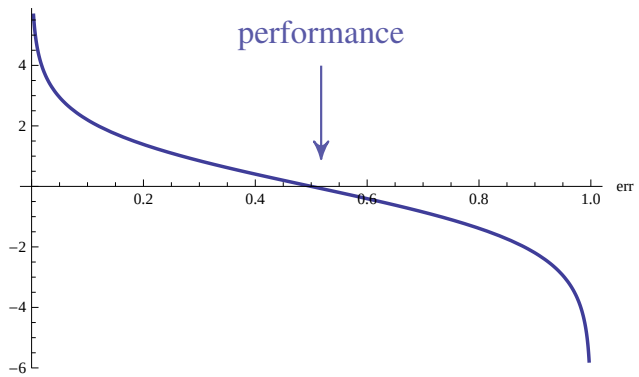
Hence:

$$w_i^{(\text{m})} = \begin{cases} w_i^{(\text{m-1})} & \text{if } g_m \text{ classifies } x_i \text{ correctly} \\ w_i^{(\text{m-1})} \cdot \frac{1-\text{err}_m}{\text{err}_m} & \text{if } g_m \text{ misclassifies } x_i \end{cases}$$



$$\frac{1-\text{err}_m}{\text{err}_m}$$

0.5 = random performance

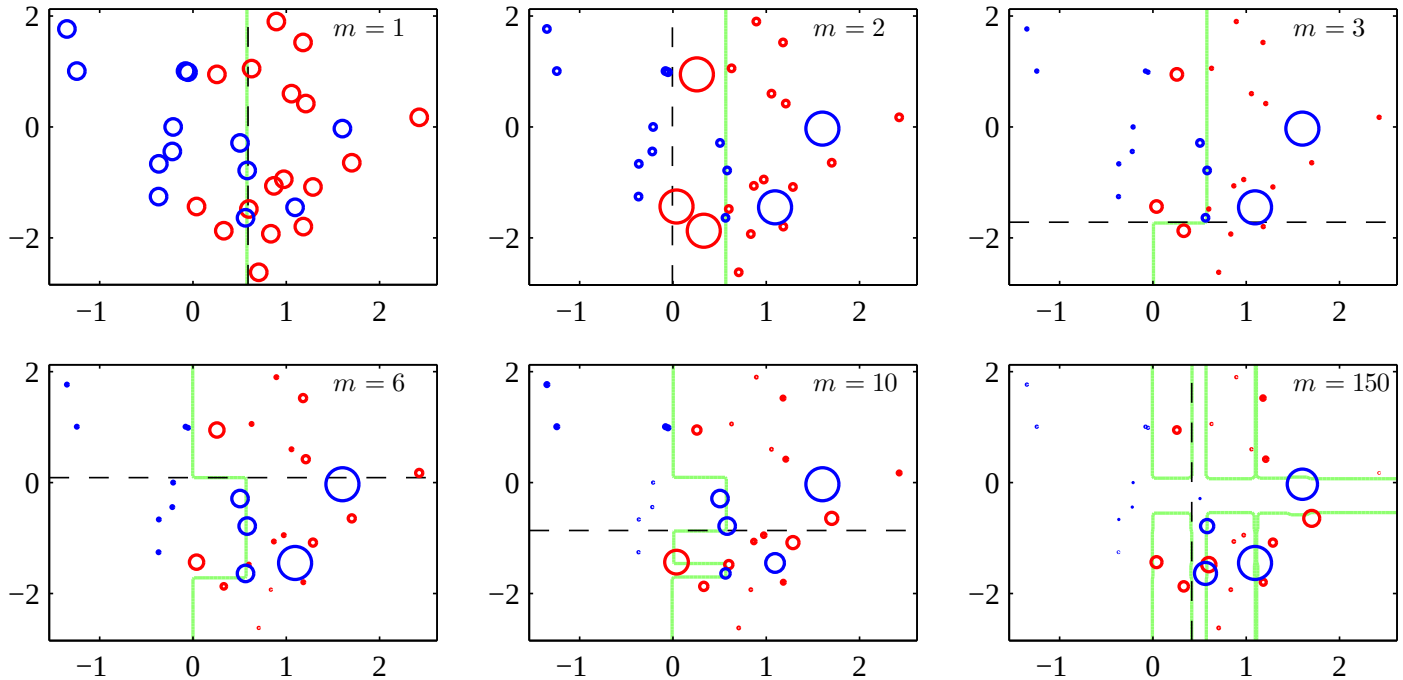


$$\alpha_m = \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$$

Weighted classifier

$$f(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m g_m(x)\right)$$

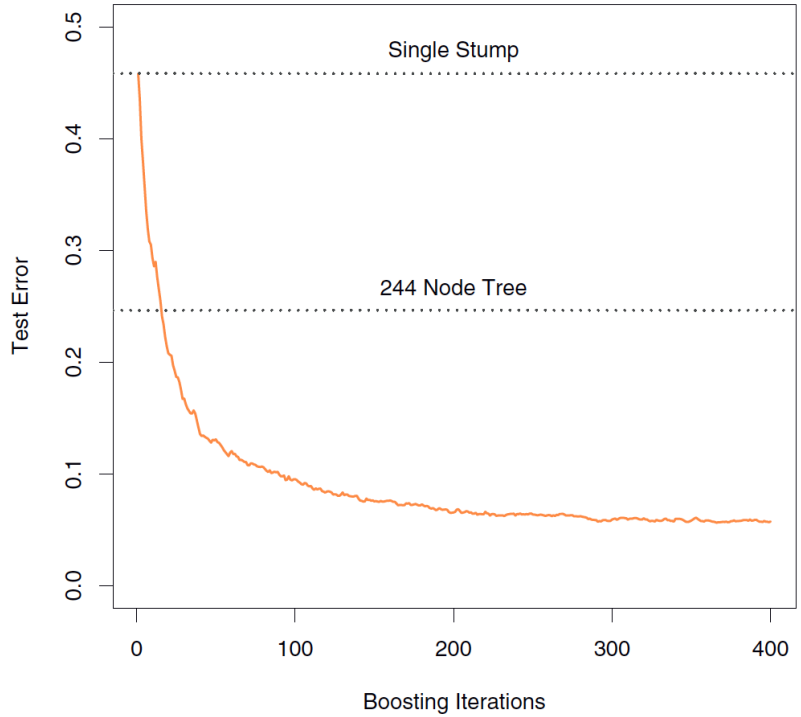
ILLUSTRATION



Circle = data points, circle size = weight.

Dashed line: Current weak learner. Green line: Aggregate decision boundary.

AdaBoost test error (simulated data)



- Weak learners used are decision stumps.
- Combining many trees of depth 1 yields much better results than a single large tree.

Properties

- AdaBoost is one of most widely used classifiers in applications.
- Decision boundary is non-linear.
- Can handle multiple classes if weak learner can do so.

Test vs training error

- Most training algorithms (e.g. Perceptron) terminate when training error reaches minimum.
- AdaBoost weights keep changing even if training error is minimal.
- Interestingly, the *test error* typically keeps decreasing even *after* training error has stabilized at minimal value.
- It can be shown that this behavior can be interpreted in terms of a margin:
  - Adding additional classifiers slowly pushes overall f towards a maximum-margin solution.
  - May not improve training error, but improves generalization properties.
- This does *not* imply that boosting magically outperforms SVMs, only that minimal training error does not imply an optimal solution.

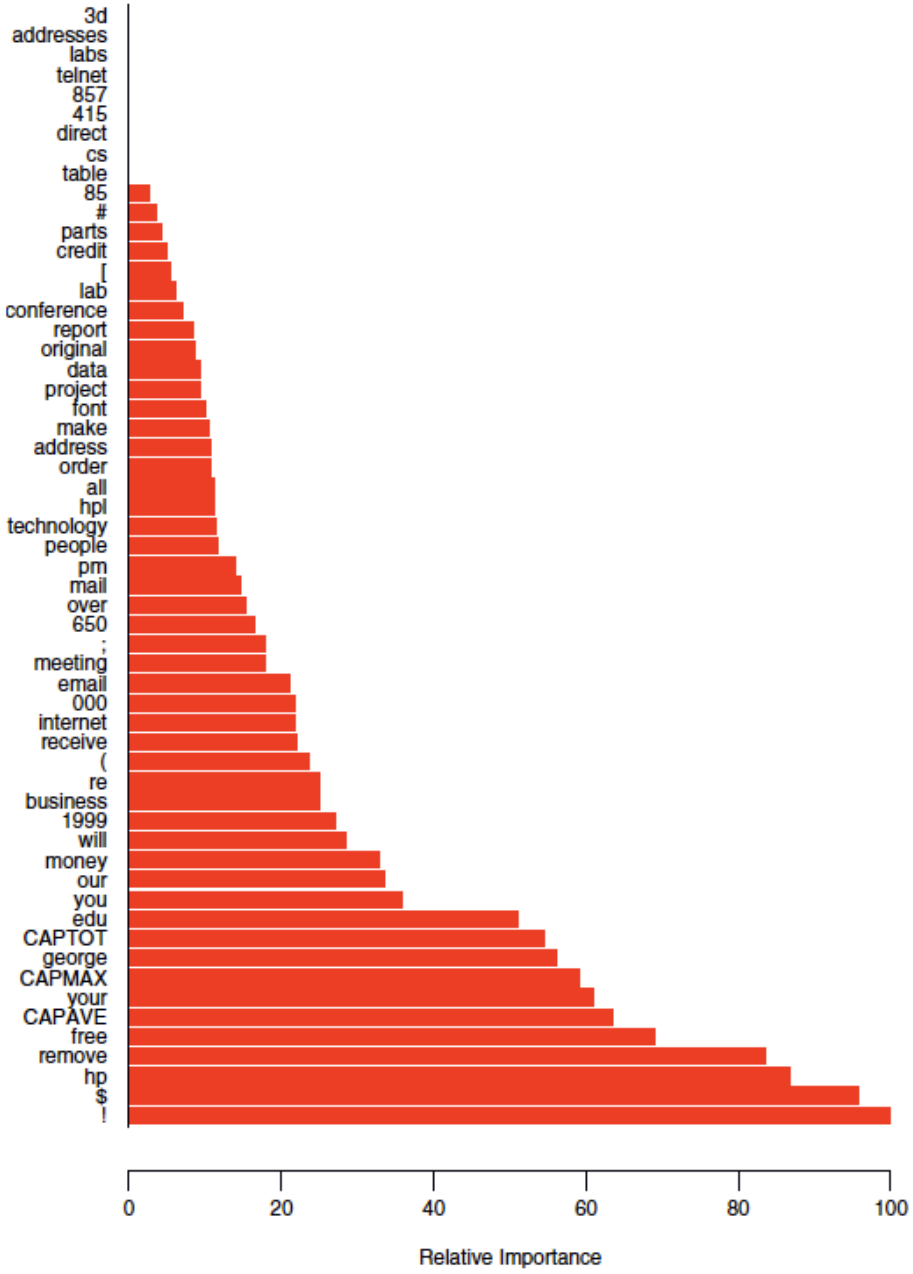
AdaBoost with Decision Stumps

- Once AdaBoost has trained a classifier, the weights α_m tell us which of the weak learners are important (i.e. classify large subsets of the data well).
- If we use Decision Stumps as weak learners, each f_m corresponds to one axis.
- From the weights α, we can read off which axis are important to separate the classes.

Terminology

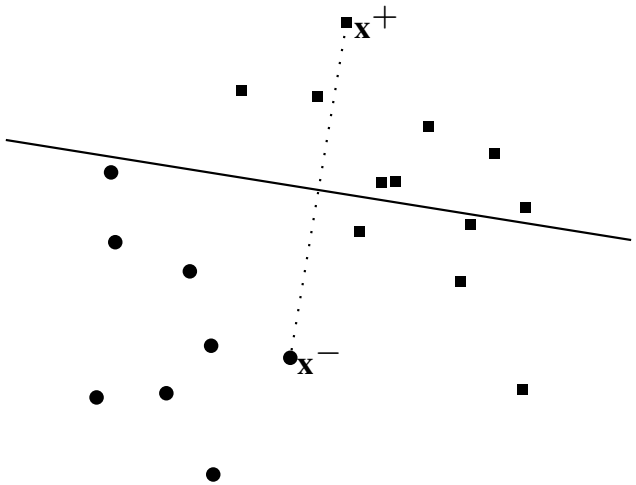
The dimensions of $\mathbb{R}^d$ (= the measurements) are often called the **features** of the data. The process of selecting features which contain important information for the problem is called **feature selection**. Thus, AdaBoost with Decision Stumps can be used to perform feature selection.

Spam Data



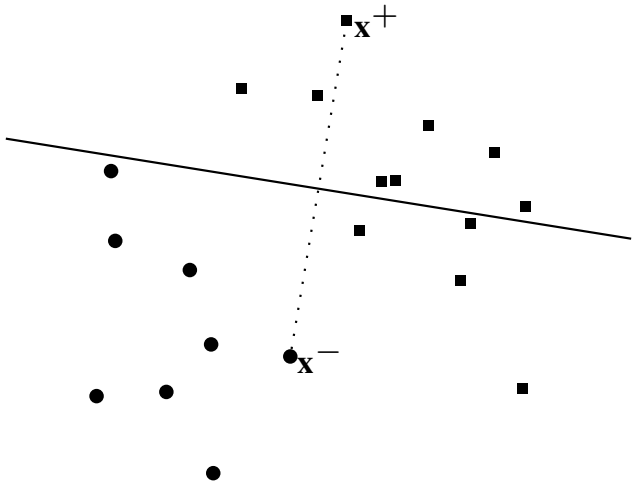
- Tree classifier: 9.3% overall error rate
- Boosting with decision stumps: 4.5%
- Figure shows feature selection results of Boosting.

HOMEWORK: A PRIMITIVE ENSEMBLE



Idea

- Try to implement the "randomly throwing out hyperplanes" idea directly.
- Strategy: Build a "weak lerner" by selecting two points at random and let them determine a hyperplane.



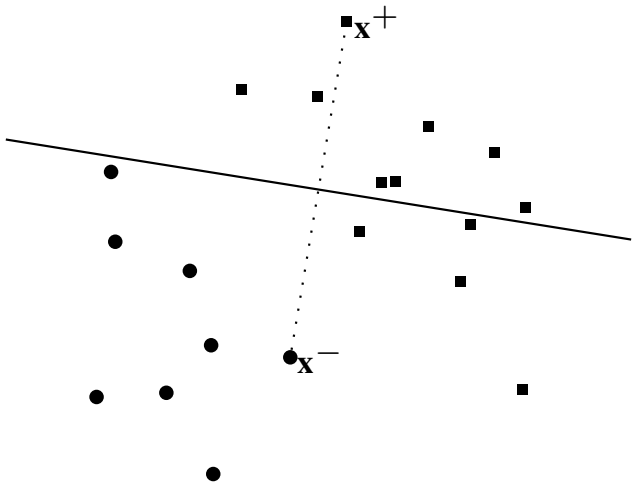
Weak classifier

- Choose two training data points $\mathbf{x}^-$ and $\mathbf{x}^+$, one in each class.
- Place an affine plane "in the middle" between the two:

$$\mathbf{w} := \frac{\mathbf{x}^+ - \mathbf{x}^-}{\|\mathbf{x}^+ - \mathbf{x}^-\|} \quad \text{and} \quad c := \left\langle \mathbf{w}, \mathbf{x}^- + \frac{1}{2}(\mathbf{x}^+ - \mathbf{x}^-) \right\rangle$$

- Choose the orientation with smaller training error: Define weak classifier as

$$f(\,.\,) = \operatorname{sgn}(\langle\,.\,, \mathbf{v}\rangle - c) \quad \text{where either } \mathbf{v} := \mathbf{w} \text{ or } \mathbf{v} := -\mathbf{w}\,.$$



Ensemble training

- Split the available data into to equally sized parts (training and test).
- Select m pairs of points $(\mathbf{x}_1^-, \mathbf{x}_1^+), \ldots, (\mathbf{x}_m^-, \mathbf{x}_m^+)$ uniformly (with replacement).
- For each such pair $(\mathbf{x}_i^-, \mathbf{x}_i^+)$, compute the classifer f_i given by $(\mathbf{v}_i, c_i)$ as described above.
- The overall classifier g_m is defined as the majority vote

$$g_m(\mathbf{x}) = \operatorname{sgn}\Big(\sum_{j=1}^{m} f_i(\mathbf{x})\Big) = \operatorname{sgn}\Big(\sum_{j=1}^{m} \operatorname{sgn}(\langle \mathbf{v}_i, \mathbf{x}\rangle - c_i)\Big)$$

Application: Face Detection

Searching for faces in images

Two problems:

- **Face detection** Find locations of all faces in image. Two classes.
- **Face recognition** Identify a person depicted in an image by recognizing the face. One class per person to be identified + background class (all other people).

Face detection can be regarded as a solved problem. Face recognition is not solved.

Face detection as a classification problem

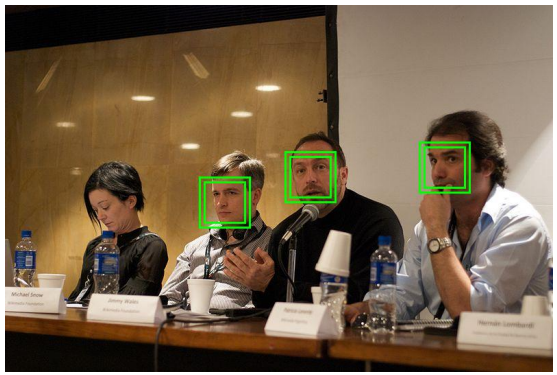
- Divide image into patches.
- Classify each patch as "face" or "not face"

Reference: Viola & Jones, “Robust real-time face detection”, Int. Journal of Computer Vision, 2004.

Unbalanced Classes

- Our assumption so far was that both classes are roughly of the same size.
- Some problems: One class is much larger.
- Example: Face detection.

  - Image subdivided into small quadratic patches.
  - Even in pictures with several people, only small fraction of patches usually represent faces.



Standard classifier training

Suppose positive class is very small.

- Training algorithm can achieve good error rate by classifiying *all* data as negative.
- The error rate will be precisely the proportion of points in positive class.

Image source: Wikipedia

Addressing class imbalance

- We have to change cost function: False negatives (= classify face as background) are expensive.
- Consequence: Training algorithm will focus on keeping proportion of false negatives small.
- Problem: Will result in many false positives (= background classified as face).

Cascade approach

- Use many classifiers linked in a chain structure ("cascade").
- Each classifier eliminates part of the negative class.
- With each step down the cascade, class sizes become more even.

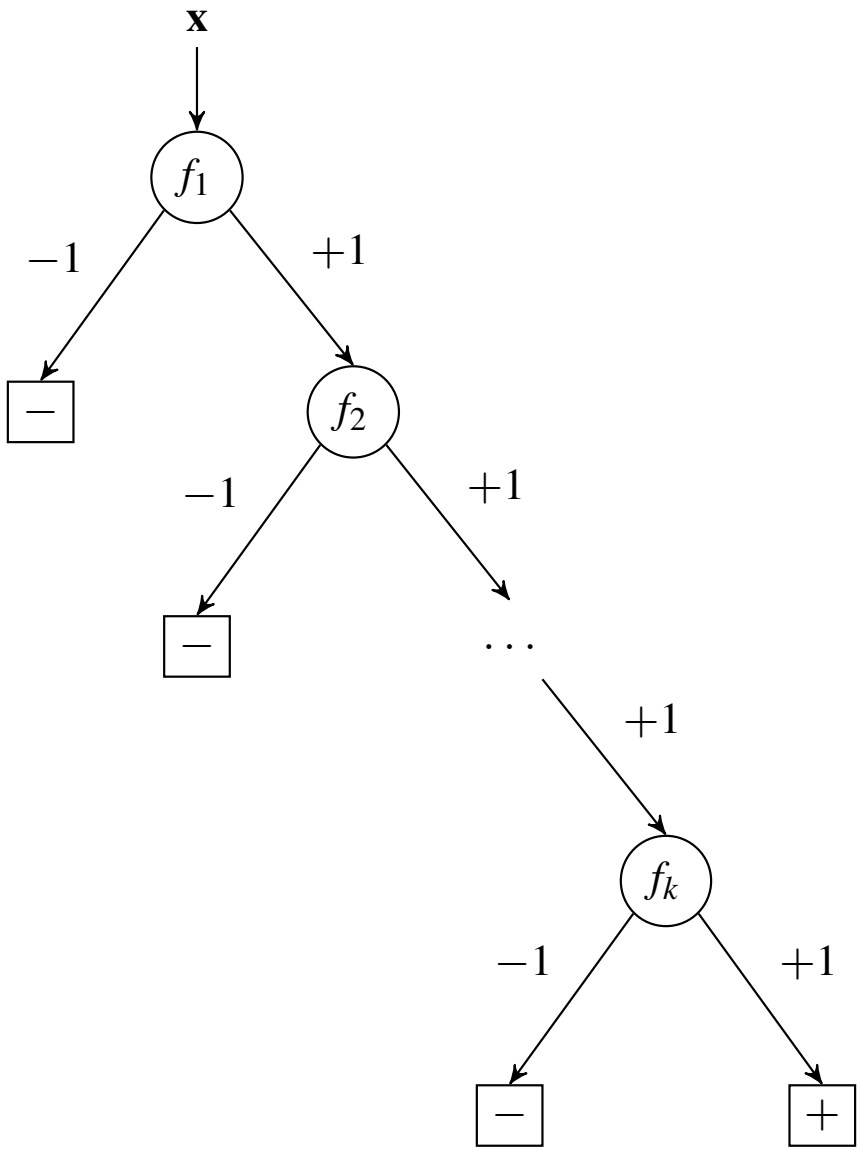
Training a cascade

Use imbalanced loss, with very low false negative rate for each f_j.

1. Train classifier f_1 on entire training data set.
2. Remove all $\tilde{\mathbf{x}}_i$ in negative class which f_1 classifies correctly from training set.
3. On smaller training set, train f_2.
4. ...
5. On remaining data at final stage, train f_k.

Classifying with a cascade

- If any f_j classifies $\mathbf{x}$ as negative, $f(\mathbf{x}) = -1$.
- Only if all f_j classify $\mathbf{x}$ as positive, $f(\mathbf{x}) = +1$.



We have to consider two rates

$$\text{false positive rate} \qquad \text{FPR}(f_j) = \frac{\#\text{negative points classified as "+1"}}{\#\text{negative training points at stage } j}$$

$$\text{recall (detection rate)} \qquad \text{Recall}(f_j) = \frac{\#\text{correctly classified positive points}}{\#\text{positive training points at stage } j}$$

We want to achieve a low value of $\text{FPR}(f)$ and a high value of $\text{Recall}(f)$.

Class imbalance

In face detection example:

- Number of faces classified as background is (size of face class) $\times$ $(1 - \text{Recall}(f))$
- We would like to see a decently high detection rate, say 90%
- Number of background patches classified as faces is
  (size of background class) $\times$ $(\text{FPR}(f))$
- Since background class is huge, $\text{FPR}(f)$ has to be *very* small to yield roughly the same amount of errors in both classes.

Cascade recall

The rates of the overall cascade classifier f are

$$\text{FPR}(f) = \prod_{j=1}^{k} \text{FPR}(f_j) \qquad \text{Recall}(f) = \prod_{j=1}^{k} \text{Recall}(f_j)$$

- Suppose we use a 10-stage cascade ($k = 10$)
- Each $\text{Recall}(f_j)$ is 99% and we permit $\text{FPR}(f_j)$ of 30%.
- We obtain $\text{Recall}(f) = 0.99^{10} \approx 0.90$ and $\text{FPR}(f) = 0.3^{10} \approx 6 \times 10^{-6}$

Objectives

- Classification step should be computationally efficient.
- Expensive training affordable.

Strategy

- Extract very large set of measurements (features), i.e. d in $\mathbb{R}^d$ large.
- Use Boosting with decision stumps.
- From Boosting weights, select small number of important features.
- Class imbalance: Use Cascade.

Classification step

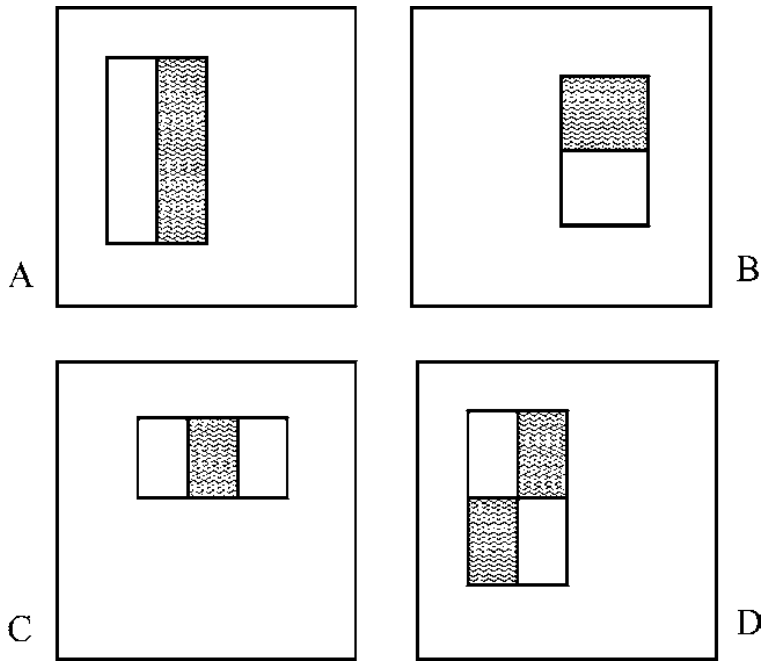
Compute only the selected features from input image.

Extraction method

1. Enumerate possible windows (different shapes and locations) by $j = 1, \ldots, d$.
2. For training image i and each window j, compute

   x_{ij} := average of pixel values in gray block(s)
   − average of pixel values in white block(s)

3. Collect values for all j in a vector $\mathbf{x}_i := (x_{i1}, \ldots, x_{id}) \in \mathbb{R}^d$.

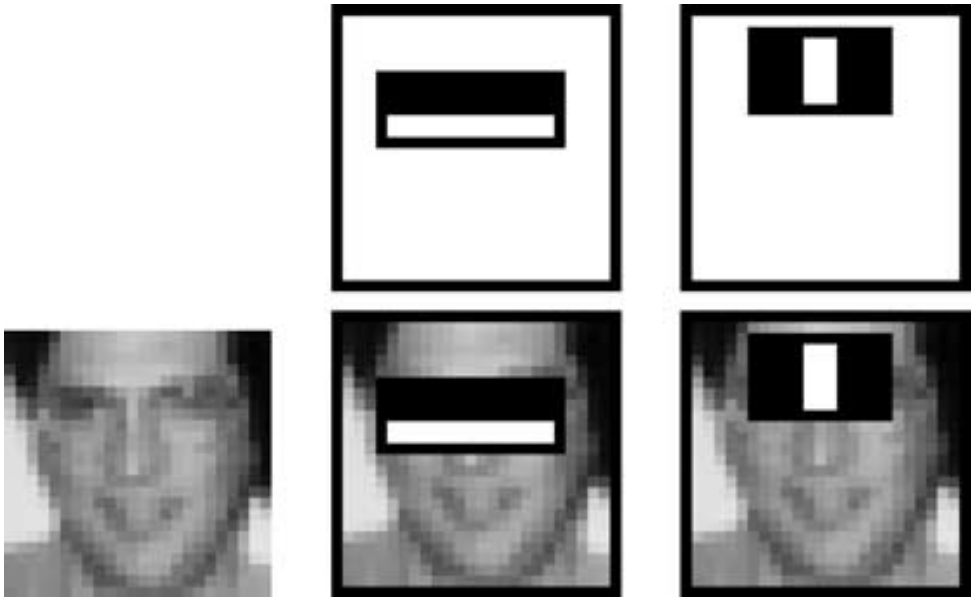


The dimension is huge

- One entry for (almost) every possible location of a rectangle in image.
- Start with small rectangles and increase edge length repeatedly by 1.5.
- In Viola-Jones paper: Images are 384×288 pixels, $d \approx 160000$.

Selected Features

First two selected features



200 features are selected in total.

Training procedure

1. User selects acceptable rates (FPR and Recall) for each level of the cascade.
2. At each level of the cascade:
    - Train a boosting classifier.
    - Gradually increase the number of selected features until required rates are achieved.

Use of training data

Each training step uses:

- All positive examples (= faces).
- Negative examples (= non-faces) misclassified at previous cascade layer.

Example Results

RESULTS

Table 3. Detection rates for various numbers of false positives on the MIT + CMU test set containing 130 images and 507 faces.

	False detections							
Detector	10	31	50	65	78	95	167	422
Viola-Jones	76.1%	88.4%	91.4%	92.0%	92.1%	92.9%	93.9%	94.1%
Viola-Jones (voting)	81.1%	89.7%	92.1%	93.1%	93.1%	93.2%	93.7%	–
Rowley-Baluja-Kanade	83.2%	86.0%	–	–	–	89.2%	90.1%	89.9%
Schneiderman-Kanade	–	–	–	94.4%	–	–	–	–
Roth-Yang-Ahuja	–	–	–	–	(94.8%)	–	–	–

BAGGING AND RANDOM FORESTS

We briefly review a technique called *bootstrap* on which bagging and random forests are based.

Bootstrap

Bootstrap (or **resampling**) is a technique for improving the quality of estimators.

Resampling = sampling from the empirical distribution

Application to ensemble methods

- We will use resampling to generate weak learners for classification.
- We discuss two classifiers which use resampling: Bagging and random forests.
- Before we do so, we consider the traditional application of Bootstrap, namely improving estimators.

Given

- A sample $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n$.
- An estimator $\hat{S}$ for a statistic S.

Bootstrap algorithm

1. Generate B **bootstrap samples** $\mathcal{B}_1, \ldots, \mathcal{B}_B$. Each bootstrap sample is obtained by sampling n times with replacement from the sample data. (Note: Data points can appear multiple times in any $\mathcal{B}_b$.)
2. Evaluate the estimator on each bootstrap sample:

   $$\hat{S}_b := \hat{S}(\mathcal{B}_b)$$

   (That is: We estimate S pretending that $\mathcal{B}_b$ is the data.)
3. Compute the **bootstrap estimate** of S by averaging over all bootstrap samples:

   $$\hat{S}_{\text{BS}} := \frac{1}{B} \sum_{b=1}^{B} \hat{S}_b$$

Recall: Plug-in estimators for mean and variance

$$\hat{\mu} := \frac{1}{n}\sum_{i=1}^{n} \tilde{\mathbf{x}}_i \qquad \hat{\sigma}^2 := \frac{1}{n}\sum_{i=1}^{n} (\tilde{\mathbf{x}}_i - \hat{\mu})^2$$

Bootstrap Variance Estimate

1. For $b = 1, \ldots, B$, generate a boostrap sample $\mathcal{B}_b$. In detail:
   For $i = 1, \ldots, n$:
   - Sample an index $j \in \{1, \ldots, n\}$.
   - Set $\tilde{\mathbf{x}}_i^{(b)} := \tilde{\mathbf{x}}_j$ and add it to $\mathcal{B}_b$.
2. For each b, compute mean and variance estimates:

$$\hat{\mu}_b := \frac{1}{n}\sum_{i=1}^{n} \tilde{\mathbf{x}}_i^{(b)} \qquad \hat{\sigma}_b^2 := \frac{1}{n}\sum_{i=1}^{n} (\tilde{\mathbf{x}}_i^{(b)} - \hat{\mu}_b)^2$$

3. Compute the bootstrap estimate:

$$\hat{\sigma}^2_{\text{BS}} := \frac{1}{B}\sum_{b=1}^{B} \hat{\sigma}_b^2$$

Not examinable.

How Often Do We See Each Sample?

Sample $\{\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n\}$, bootstrap resamples $\mathcal{B}_1, \ldots, \mathcal{B}_B$.

In how many sets does a given $\mathbf{x}_i$ occur?

Probability for $\mathbf{x}_i$ *not* to occur in n draws:

$$\Pr\{\tilde{\mathbf{x}}_i \notin \mathcal{B}_b\} = (1 - \frac{1}{n})^n$$

For large n:

$$\lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^n = \frac{1}{e} \approx 0.3679$$

- Asymptotically, any $\tilde{\mathbf{x}}_i$ will appear in $\sim 63\%$ of the bootstrap resamples.
- Multiple occurrences possible.

How often is $\tilde{\mathbf{x}}_i$ expected to occur?

The *expected* number of occurences of each $\tilde{\mathbf{x}}_i$ is B.

Bootstrap estimate averages over reshuffled samples.

Not examinable.

Estimate variance of estimators

- Since estimator $\hat{S}$ depends on (random) data, it is a random variable.
- The more this variable scatters, the less we can trust our estimate.
- If scatter is high, we can expect the values $\hat{S}_b$ to scatter as well.
- In previous example, this means: Estimating the variance of the variance estimator.

Variance reduction

- Averaging over the individual bootstrap samples can reduce the variance in $\hat{S}$.
- In other words: $\hat{S}_{\text{BS}}$ typically has lower variance than $\hat{S}$.
- This is the property we will use for classicifation in the following.

As alternative to cross validation

To estimate prediction error of classifier:

- For each b, train on $\mathcal{B}_b$, estimate risk on points not in $\mathcal{B}_b$.
- Average risk estimates over bootstrap samples.

Not examinable.

Idea

- Recall Boosting: Weak learners are deterministic, but selected to exhibit high variance.
- Strategy now: Randomly distort data set by resampling.
- Train weak learners on resampled training sets.
- Resulting algorithm: **Bagging** (= **B**ootstrap **agg**regation)

REPRESENTATION OF CLASS LABELS

For Bagging with K classes, we represent class labels as vectors:

$$\mathbf{x}_i \text{ in class } k \qquad \text{as} \qquad y_i = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \longleftarrow k\text{th entry}$$

This way, we can average together multiple class labels:

$$\frac{1}{n}(y_1 + \ldots + y_n) = \begin{pmatrix} p_1 \\ \vdots \\ p_k \\ \vdots \\ p_K \end{pmatrix}$$

We can interpret p_k as the probability that one of the n points is in class k.

Bagging: Algorithm

Training

For $b = 1, \ldots, B$:

1. Draw a bootstrap sample $\mathcal{B}_b$ of size n from training data.
2. Train a classifier f_b on $\mathcal{B}_b$.

Classification

- Compute

$$f_{\text{avg}}(\mathbf{x}) := \frac{1}{B} \sum_{b=1}^{B} f_b(\mathbf{x})$$

  This is a vector of the form $f_{\text{avg}}(\mathbf{x}) = (p_1(\mathbf{x}), \ldots, p_k(\mathbf{x}))$.

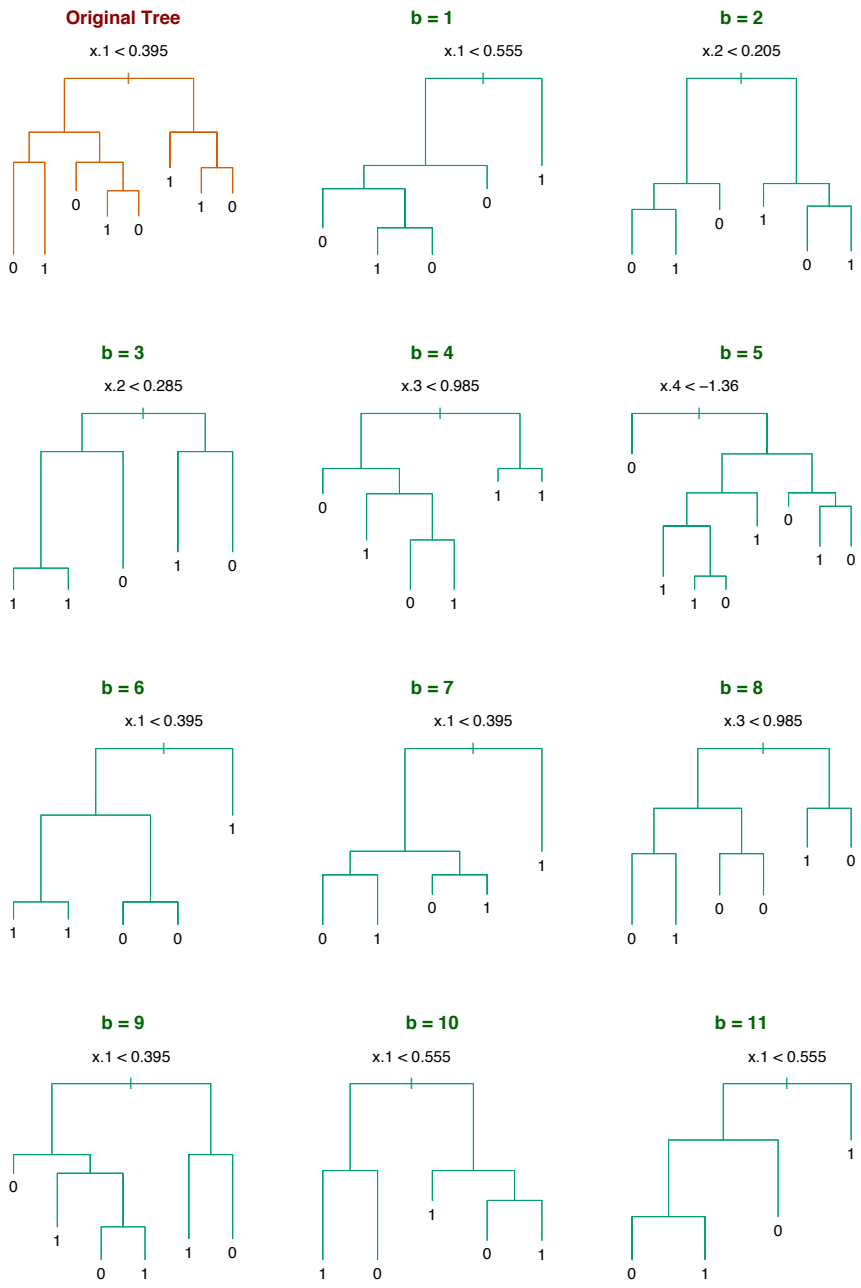
- The Bagging classifier is given by

$$f_{\text{Bagging}}(\mathbf{x}) := \arg\max_k \{p_1(\mathbf{x}), \ldots, p_k(\mathbf{x})\} \ ,$$

  i.e. we predict the class label which most weak learners have voted for.

EXAMPLE: BAGGING TREES

- Two classes, each with Gaussian distribution in $\mathbb{R}^5$.
- Note the variance between bootstrapped trees.



Bagging vs. Boosting

- Bagging works particularly well for trees, since trees have high variance.
- Boosting typically outperforms bagging with trees.
- The main culprit is usually dependence: Boosting is better at reducing correlation between the trees than bagging is.

Random Forests

Modification of bagging with trees designed to further reduce correlation.

- Tree training optimizes each split over all dimensions.
- Random forests choose a different subset of dimensions *at each split*.
- Optimal split is chosen within the subset.
- The subset is chosen at random out of all dimensions $\{1, \ldots, d\}$.

Training

Input parameter: m (positive integer with $m < d$)

For $b = 1, \ldots, B$:

1. Draw a bootstrap sample $\mathcal{B}_b$ of size n from training data.
2. Train a tree classifier f_b on $\mathcal{B}_b$, where each split is computed as follows:
   - Select m axes in $\mathbb{R}_d$ at random.
   - Find the best split (j^*, t^*) on this subset of dimensions.
   - Split current node along axis j^* at t^*.

Classification

Exactly as for bagging: Classify by majority vote among the B trees. More precisely:

- Compute $f_{\text{avg}}(\mathbf{x}) := (p_1(\mathbf{x}), \ldots, p_k(\mathbf{x})) := \frac{1}{B} \sum_{b=1}^{B} f_b(\mathbf{x})$
- The Random Forest classification rule is

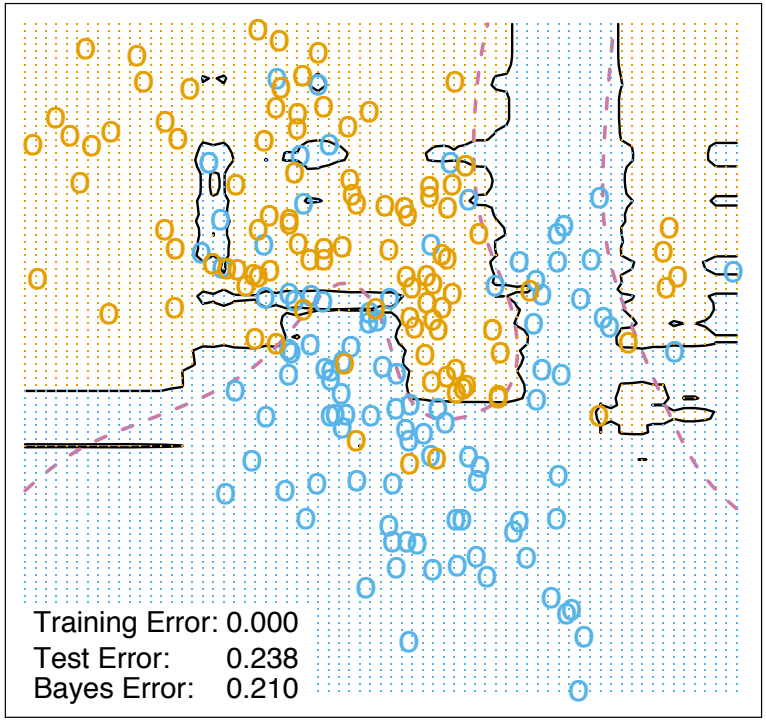
$$f_{\text{Bagging}}(\mathbf{x}) := \arg\max_k \{p_1(\mathbf{x}), \ldots, p_k(\mathbf{x})\}$$

Remarks

- Recommended value for m is $m = \lfloor \sqrt{d} \rfloor$ or smaller.
- RF typically achieve similar results as boosting. Implemented in most packages, often as standard classifier.

Example: Synthetic Data

- This is the RF classification boundary on the synthetic data we have already seen a few times.
- Note the bias towards axis-parallel alignment.



Application: Cancer Diagnosis

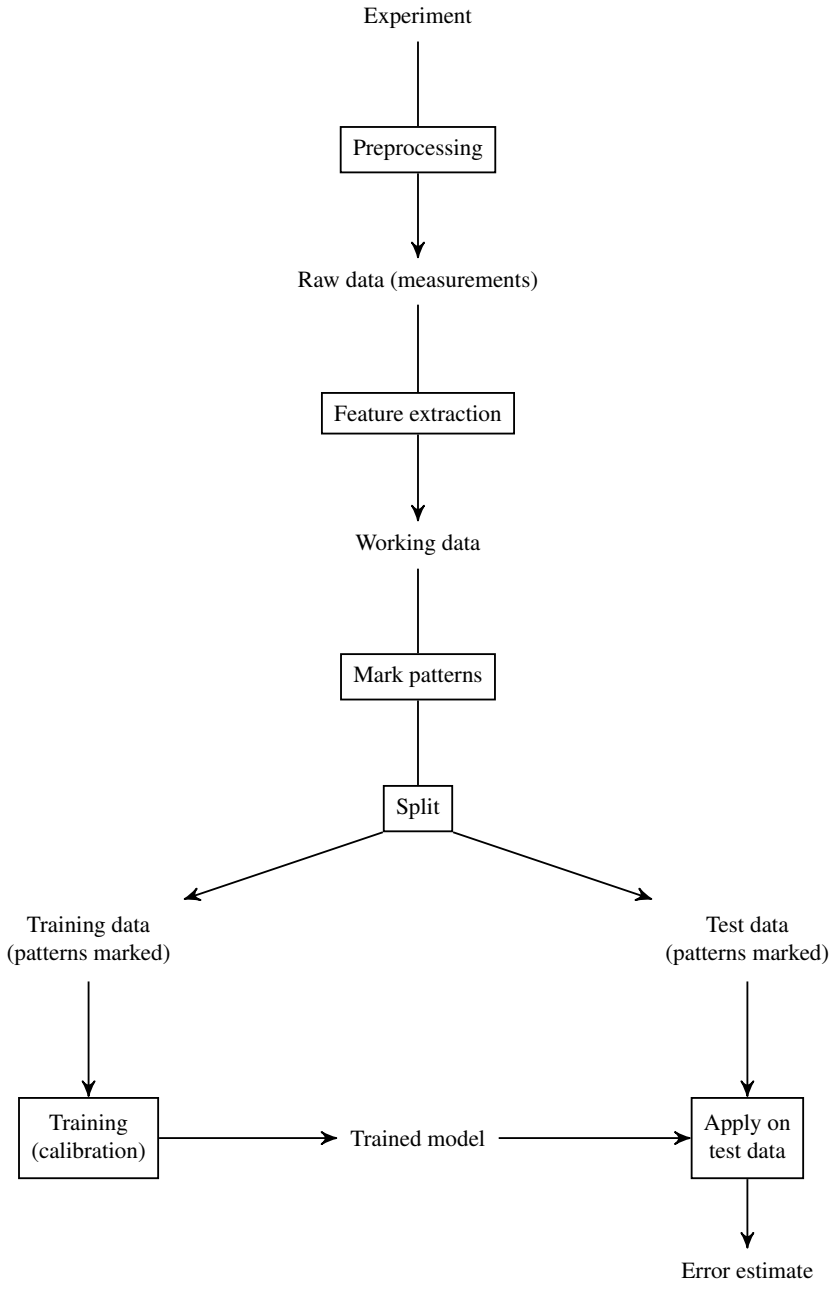
Kidney cancer diagnosis: Clinical procedure

- Take tissue sample from patient's kidney
- Preprocess sample and photograph under microscope
- A pathologist looks at the image and diagnosis patient on scale from healthy to advanced stage cancer

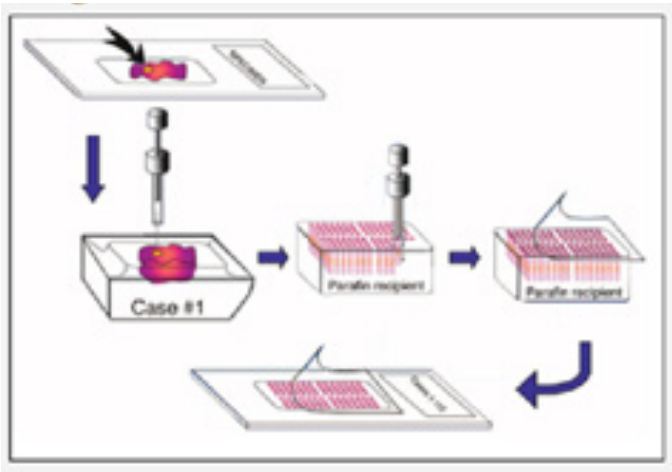
Task

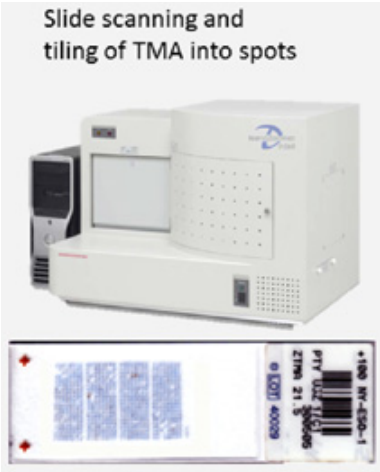
- Empirically, the results vary significantly between pathologists
- The objective is to build a classifier that produces a diagnosis using the same scale as the pathologist, hopefully with more stable results.

Data Analysis Pipeline

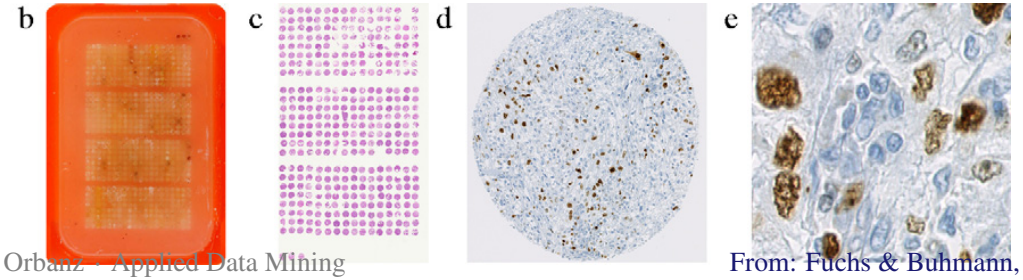


PREPROCESSING



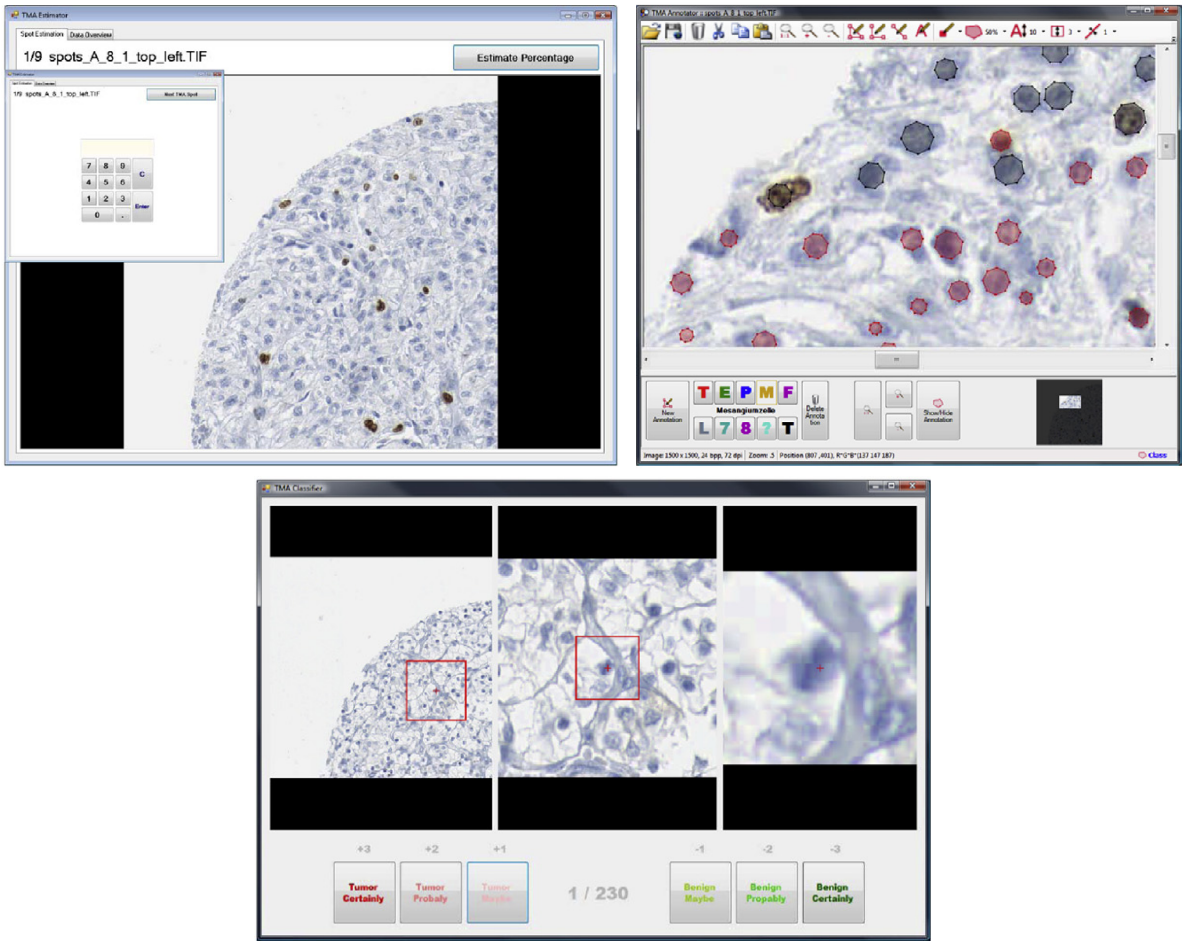


1. Tissue sample is “stained” with marking fluid
2. Thin slice is cut and placed on microscope slide
3. Sample is photographed under microscope
4. Tumor cells absorb more marker fluid and tend to be darker



From: Fuchs & Buhmann, Computerized Medical Imaging and Graphics Vol. 35, 2011.

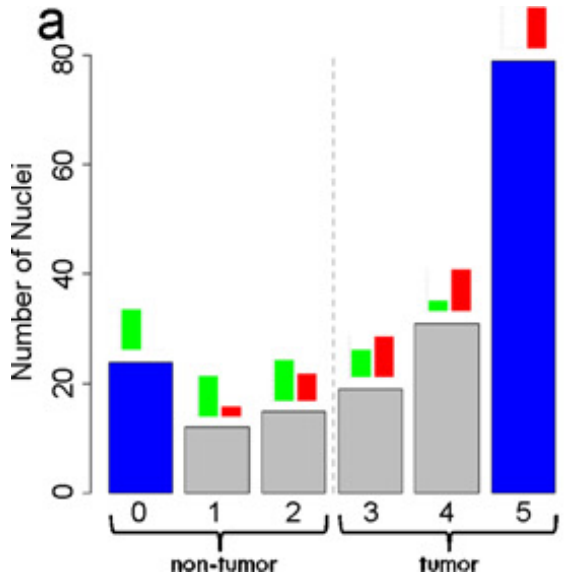
LABELING



A pathologist uses a software with a graphical user interface to (1) mark the locations of nuclei and (2) label nuclei as healthy/cancerous.

From: Fuchs & Buhmann, Computerized Medical Imaging and Graphics Vol. 35, 2011.

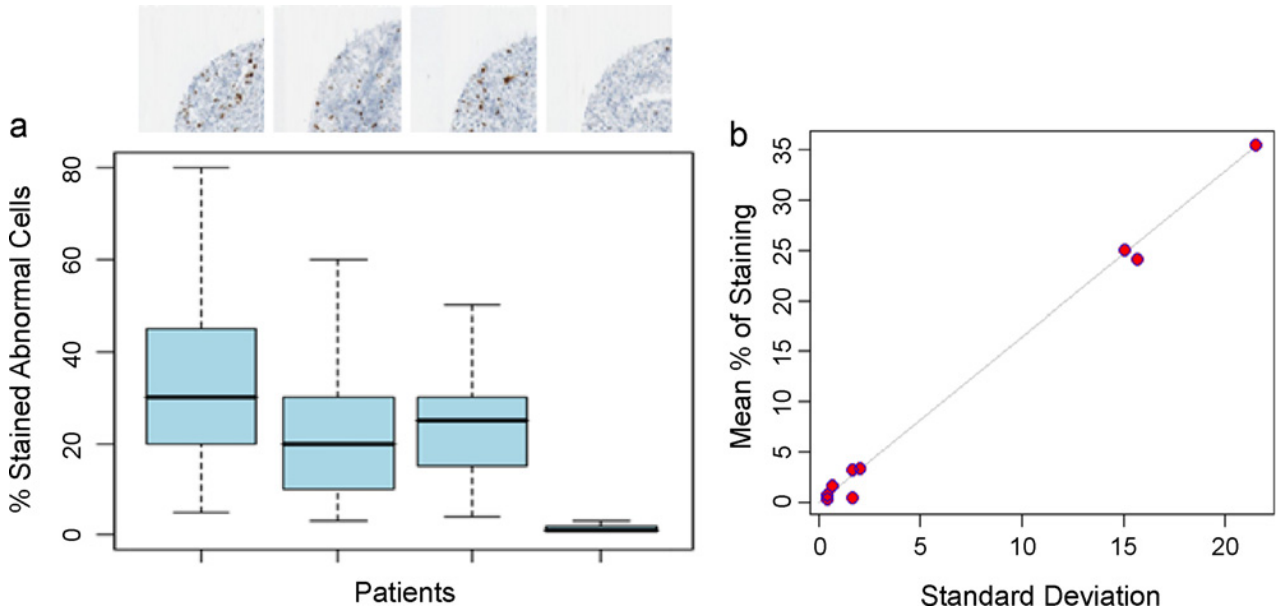
Comparison Between Experts (1)



- Five experts label the same set of nuclei (180 in total)
- For each data point (nucleus), count the number of votes $(0, \ldots, 5)$ in favor of "tumor"
- The diagram above is a histogram of the vote counts for the 180 data points
- All five experts agree if the count is 0 (all say healthy) or 5 (all say tumor)
- (The small red/green bars are the vote proportions, so they encode the same information as the numbers at the bottom.)

From: Fuchs & Buhmann, Computerized Medical Imaging and Graphics Vol. 35, 2011.

Comparison Between Experts (2)



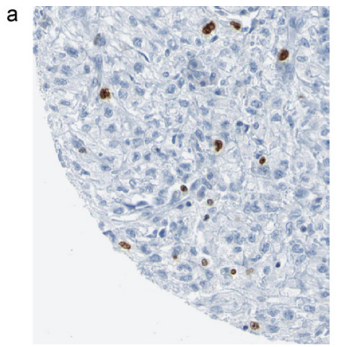
Results for 14 pathologists labeling four patients.

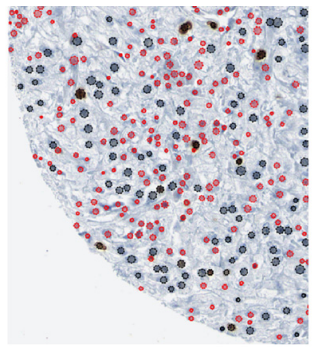
- Each box in the boxplot represents one patient.
- Box plot: The line in the middle of each box is the median. The upper and lower box boundary are the third and first quartile, respectively. The horizontal bars at either end of the dashed vertical line represent one standard deviation around the mean.
- In three of the four cases, disagreement between experts is substantial.
- Plot on the right: The standard deviation increases linearly with the overall number of stained nuclei. (Roughly, the more cancer cells there are, the more volatile the diagnosis becomes.)

From: Fuchs & Buhmann, Computerized Medical Imaging and Graphics Vol. 35, 2011.

EXAMPLE

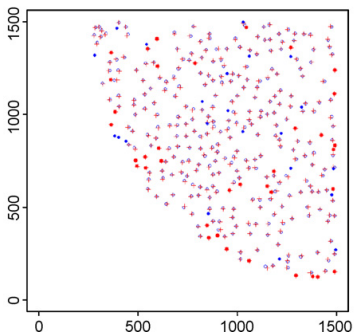
input image

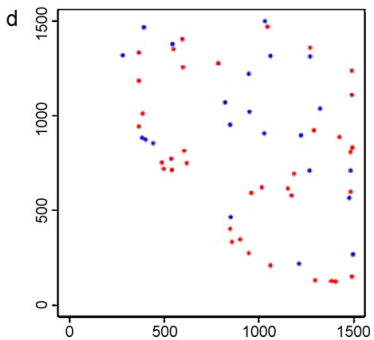




locations marked by one expert

classification by one expert





disagreement between two experts

From: Fuchs & Buhmann, Computerized Medical Imaging and Graphics Vol. 35, 2011.

Data

"Relative images" of individual cell nuclei, i.e. the difference between a nucleus image and a background patch from the same tissue sample image. That makes results less sensitive to variations between tissue samples.

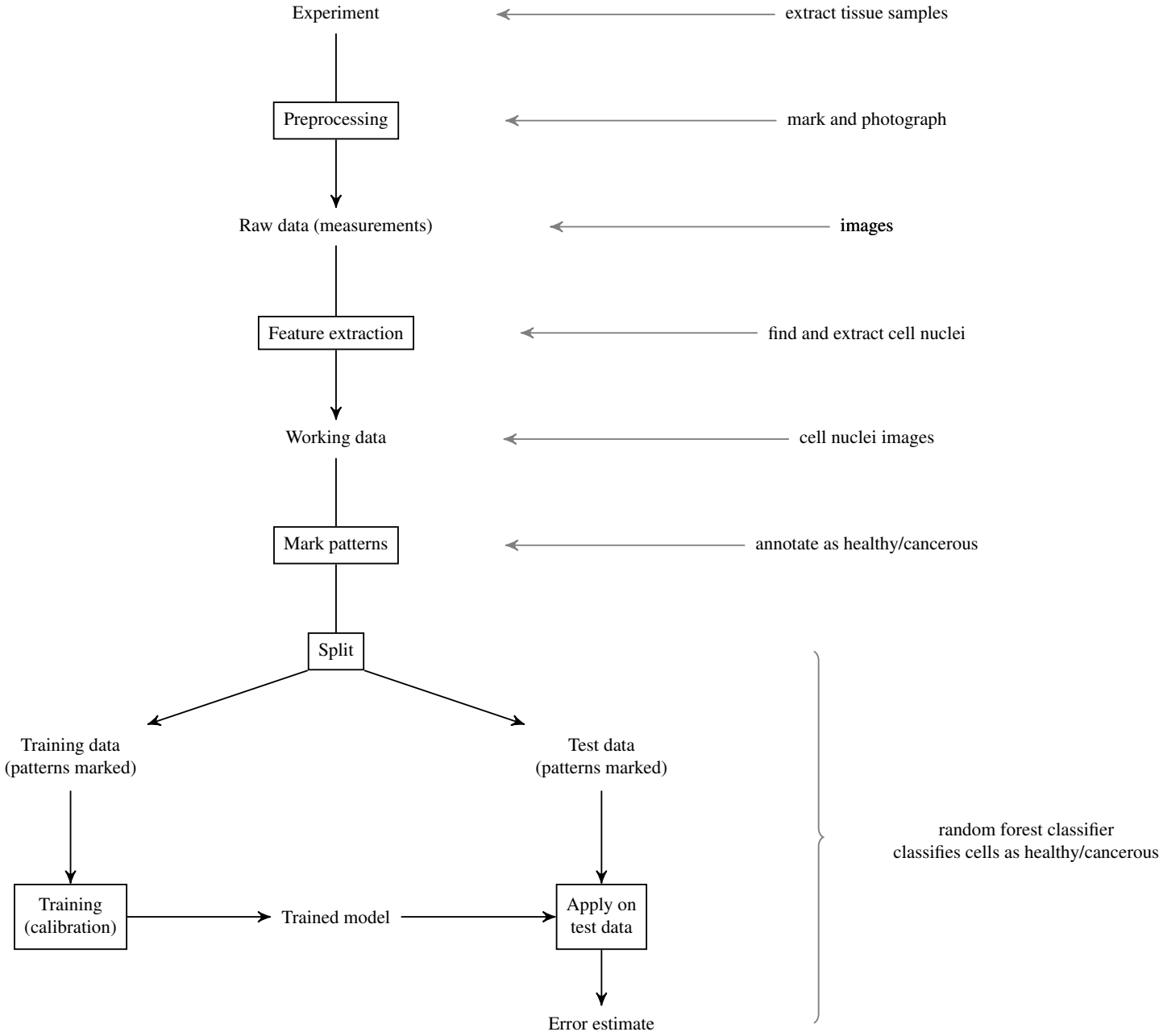
Training

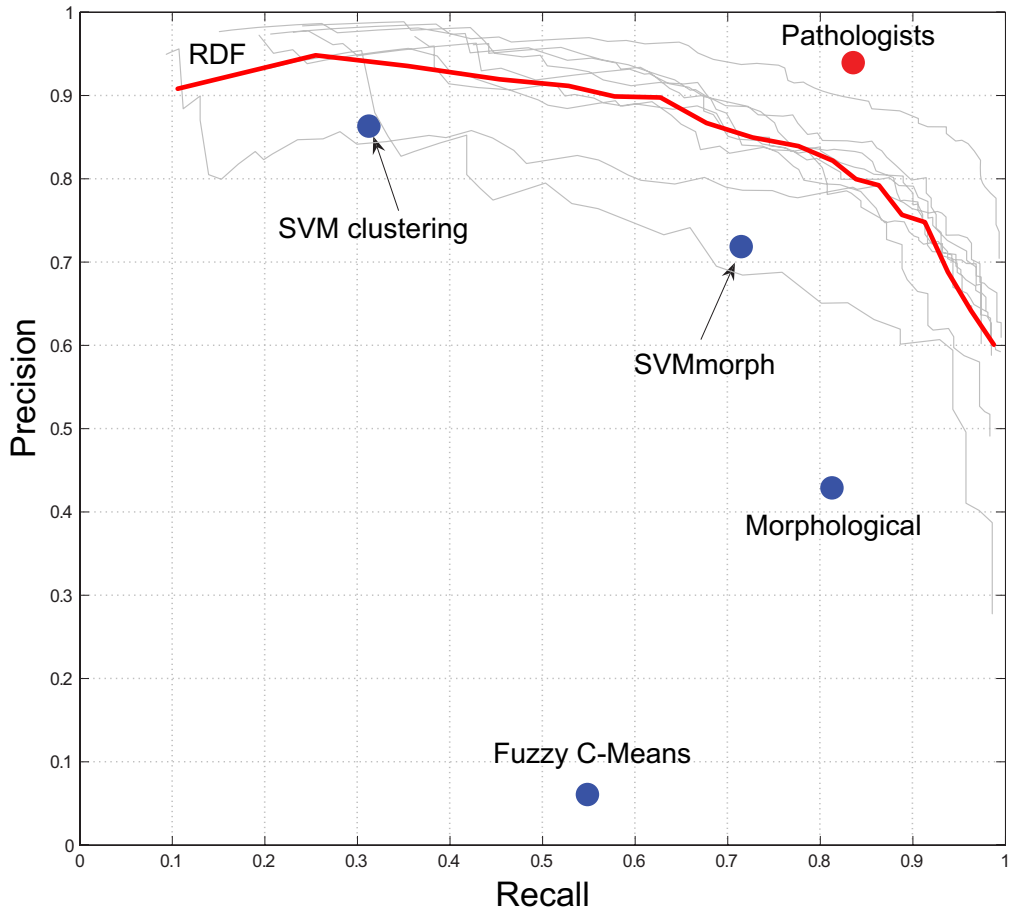
1. Nuclei are hand-labeled by pathologists.
2. A random forest classifier is trained on the relative images (as training data points) and the labels (as training labels).

Diagnosis of a new tissue sample image

1. Input: Entire tissue sample image.
2. Find nuclei using an image segmentation algorithm.
3. Extract subimages of these nuclei and apply random forest classifier.
4. Diagnose according to ratio of healthy to cancerous cells.

Data Analysis Pipeline





Precision/Recall plot for the random forest method ("RDF") compared to other classifiers. The "true label" for each data point is a randomly selected pathologist. The performance of pathologists (red dot) is the average of the aggregate result for all remaining pathologists.

REMARKS

This application illustrates a number of challenges encountered in applications:

- Generating label information is work-intensive.
- The comparison experiments show that the training/test labels themselves have limited reliability.
- These methods are now several years old. Neural networks developed in the last few years might be able to improve the feature extraction step. (More on neural networks and feature extraction later.)

NEURAL NETWORKS

The Most Important Bit

A neural network represents a function $f : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$.

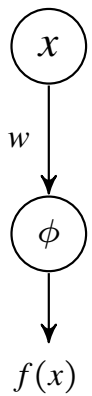
Units

The basic building block is a **node** or **unit**:

- The unit has incoming and outgoing arrows. We think of each arrow as "transmitting" a signal.
- The signal is always a scalar.
- A unit represents a function ϕ.

We read the diagram as: A scalar value (say x) is transmitted to the unit, the function ϕ is applied, and the result $\phi(x)$ is transmitted from the unit along the outgoing arrow.

Weights

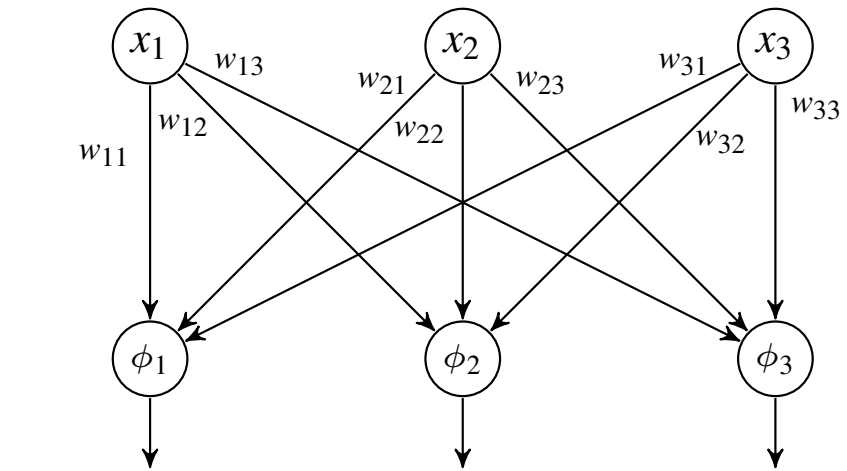


- If we want to "input" a scalar x, we represent it as a unit, too.
- We can think of this as the unit representing the constant function $g(x) = x$.
- Additionally, each arrow is usually inscribed with a (scalar) weight w.
- As the signal x passes along the edge, it is multiplied by the edge weight w.

The diagram above represents the function $f(x) := \phi(wx)$.

READING NEURAL NETWORKS

$$f : \mathbb{R}^3 \to \mathbb{R}^3 \quad \text{with input} \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_2 \end{pmatrix}$$



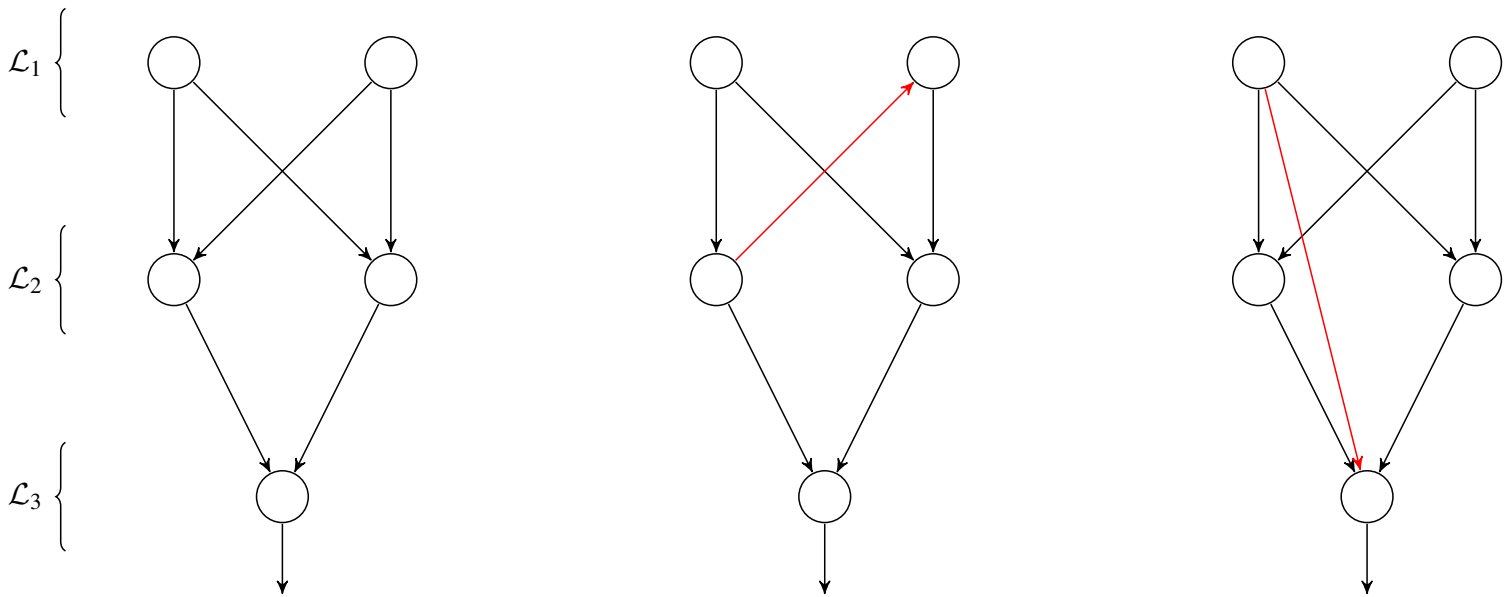
$$f_1(\mathbf{x}) = \phi_1(\langle \mathbf{w}_1, \mathbf{x} \rangle) \quad f_2(\mathbf{x}) = \phi_2(\langle \mathbf{w}_2, \mathbf{x} \rangle) \quad f_3(\mathbf{x}) = \phi_3(\langle \mathbf{w}_3, \mathbf{x} \rangle)$$

$$f(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ f_3(\mathbf{x}) \end{pmatrix} \quad \text{with} \quad f_i(\mathbf{x}) = \phi_i\left(\sum_{j=1}^{3} w_{ij} x_j\right)$$

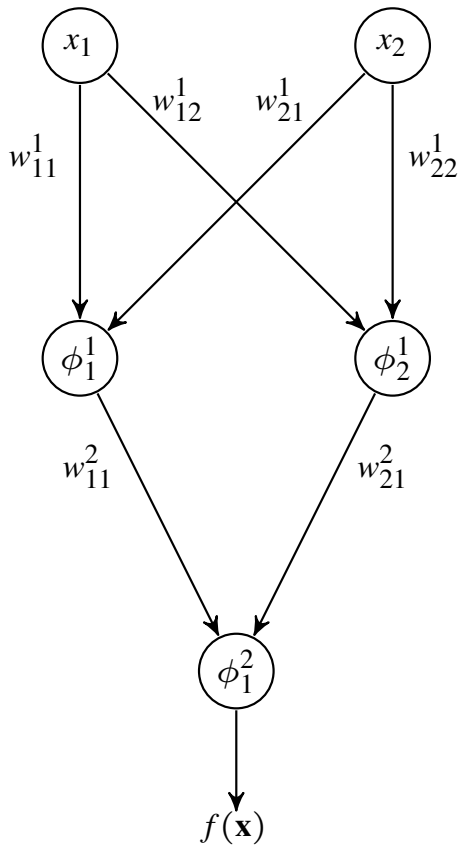
Feed-Forward Networks

A **feed-forward network** is a neural network whose units can be arranged into groups $\mathcal{L}_1, \ldots, \mathcal{L}_K$ so that connections (arrows) only pass from units in group $\mathcal{L}_k$ to units in group $\mathcal{L}_{k+1}$. The groups are called **layers**. In a feed-forward network:

- There are no connections within a layer.
- There are no backwards connections.
- There are no connections that skip layers, e.g. from $\mathcal{L}_k$ to units in group $\mathcal{L}_{k+2}$.



feed-forward | not feed-forward | not feed-forward



- This network computes the function

$$f(x_1, x_2) = \phi_1^2\Big(w_{11}^2\phi_1^1\big(w_{11}^1 x_1 + w_{21}^1 x_2\big) + w_{12}^2\phi_2^1\big(w_{21}^1 x_1 + w_{22}^1 x_2\big)\Big)$$

- Clearly, writing out f gets complicated fairly quickly as the network grows.

First shorthand: Scalar products

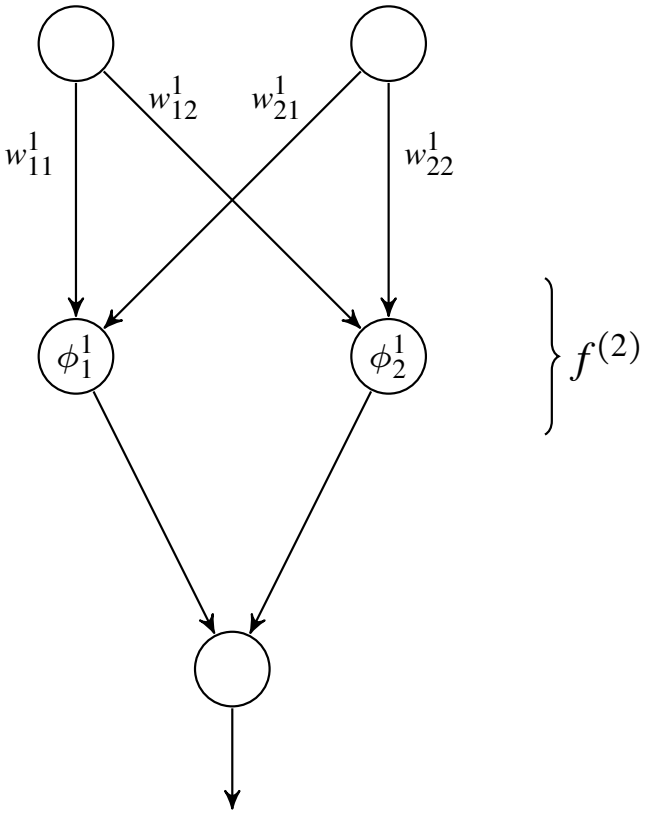
- Collect all weights coming into a unit into a vector, e.g.

$$\mathbf{w}_1^2 := (w_{11}^2, w_{21}^2)$$

- Same for inputs: $\mathbf{x} = (x_1, x_2)$
- The function then becomes

$$f(\mathbf{x}) = \phi_1^2\left(\left\langle \mathbf{w}_1^2, \begin{pmatrix}\phi_1^1(\langle \mathbf{w}_1^1, \mathbf{x}\rangle)\\ \phi_2^1(\langle \mathbf{w}_2^1, \mathbf{x}\rangle)\end{pmatrix}\right\rangle\right)$$

LAYERS



- Each layer represents a function, which takes the output values of the previous layers as its arguments.
- Suppose the output values of the two nodes at the top are y_1, y_2.
- Then the second layer defines the (two-dimensional) function

$$f^{(2)}(\mathbf{y}) = \begin{pmatrix} \phi_1^1(\langle \mathbf{w}_1^1, \mathbf{y} \rangle) \\ \phi_2^1(\langle \mathbf{w}_2^1, \mathbf{y} \rangle) \end{pmatrix}$$

Basic composition

Suppose f and g are two function $\mathbb{R} \to \mathbb{R}$. Their **composition** $g \circ f$ is the function

$$g \circ f(x) := g(f(x)) \ .$$

For example:

$$f(x) = x + 1 \qquad g(y) = y^2 \qquad g \circ f(x) = (x+1)^2$$

We could combine the same functions the other way around:

$$f \circ g(x) = x^2 + 1$$

In multiple dimensions

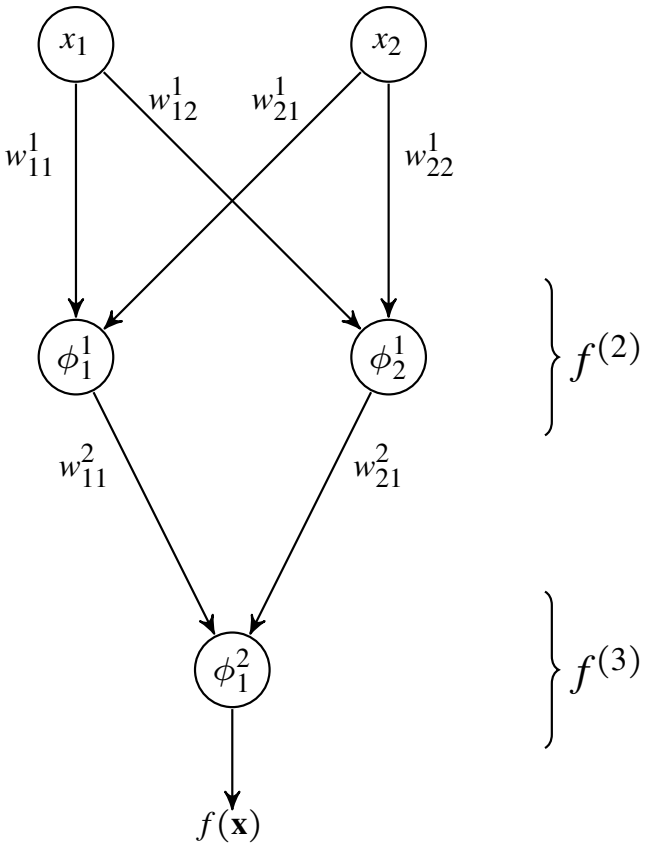
Suppose $f : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ and $g : \mathbb{R}^{d_2} \to \mathbb{R}^{d_3}$. Then

$$g \circ f(\mathbf{x}) = g(f(\mathbf{x})) \qquad \text{is a function } \mathbb{R}^{d_1} \to \mathbb{R}^{d_3} \ .$$

For example:

$$f(\mathbf{x}) = \langle \mathbf{x}, \mathbf{v} \rangle - c \qquad g(y) = \operatorname{sgn}(y) \qquad g \circ f(\mathbf{x}) = \operatorname{sgn}(\langle \mathbf{x}, \mathbf{v} \rangle - c)$$

Layers and Composition



- As above, we write

$$f^{(2)}(\,\bullet\,) = \begin{pmatrix} \phi^1_1(\langle \mathbf{w}^1_1, \bullet \rangle) \\ \phi^1_2(\langle \mathbf{w}^1_2, \bullet \rangle) \end{pmatrix}$$

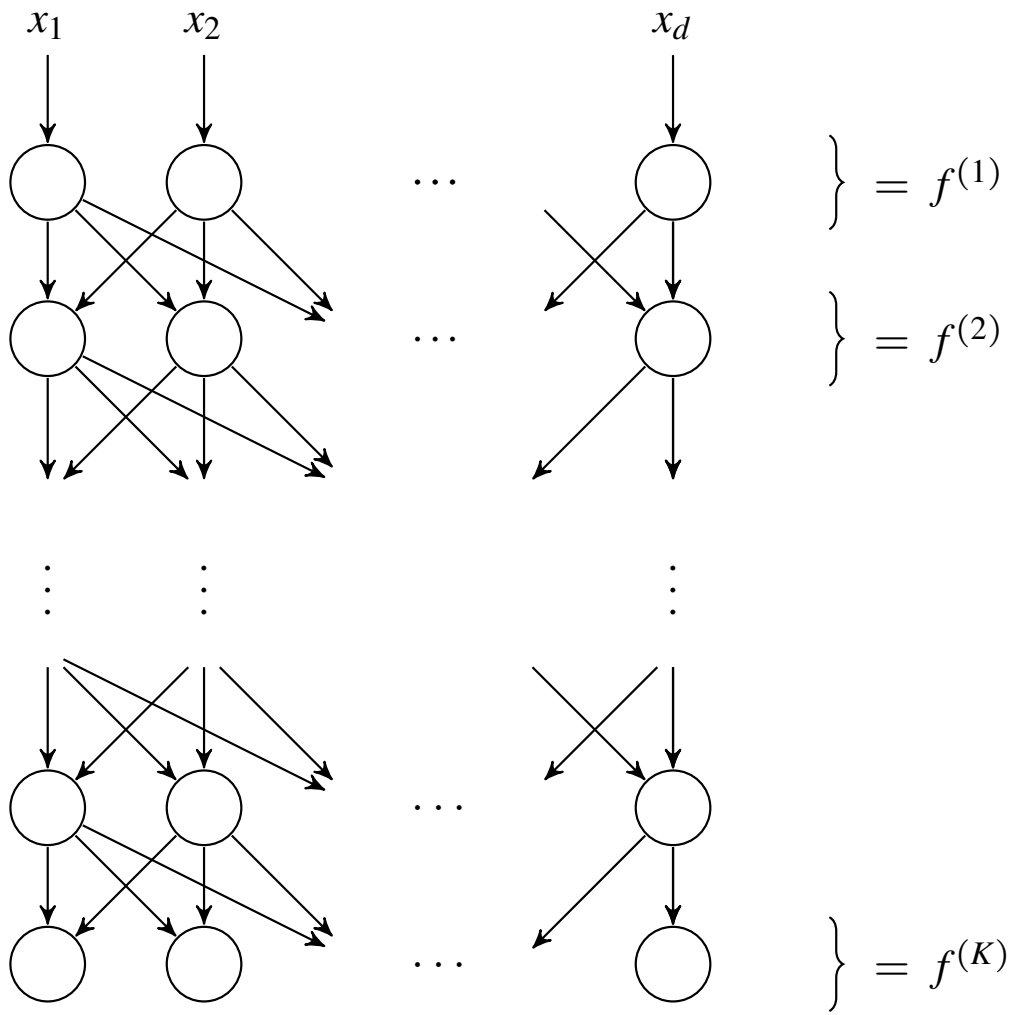
- The function for the third layer is similarly

$$f^{(3)}(\,\bullet\,) = \phi^2_1(\left\langle \mathbf{w}^2_1, \bullet \right\rangle)$$

- The entire network represents the function

$$f(\mathbf{x}) = f^{(3)}(f^{(2)}(\mathbf{x})) = f^{(3)} \circ f^{(2)}(\mathbf{x})$$

A feed-forward network represents a function as a composition of several functions, each given by one layer.



$$f(\mathbf{x}) \quad = \quad f^{(K)}(\cdots f^{(2)}(f^{(1)}(\mathbf{x}))) \quad = \quad f^{(K)} \circ \ldots \circ f^{(1)}(\mathbf{x})$$

General feed-forward networks

A feed-forward network with K layers represents a function

$$f(\mathbf{x}) \quad = \quad f^{(K)} \circ \ldots \circ f^{(1)}(\mathbf{x})$$

Each layer represents a function $f^{(k)}$. These functions are of the form:

$$f^{(k)}(\,\bullet\,) = \begin{pmatrix} \phi_1^{(k)}(\langle \mathbf{w}_1^{(k)}, \bullet \rangle) \\ \vdots \\ \phi_d^{(k)}(\langle \mathbf{w}_d^{(k)}, \bullet \rangle) \end{pmatrix} \qquad \text{typically:} \qquad \phi^{(k)}(x) = \begin{cases} \sigma(x) & \text{(sigmoid)} \\ \mathbb{I}\{\pm x > \tau\} & \text{(threshold)} \\ c & \text{(constant)} \\ x & \text{(linear)} \\ \max\{0, x\} & \text{(rectified linear)} \end{cases}$$

Dimensions

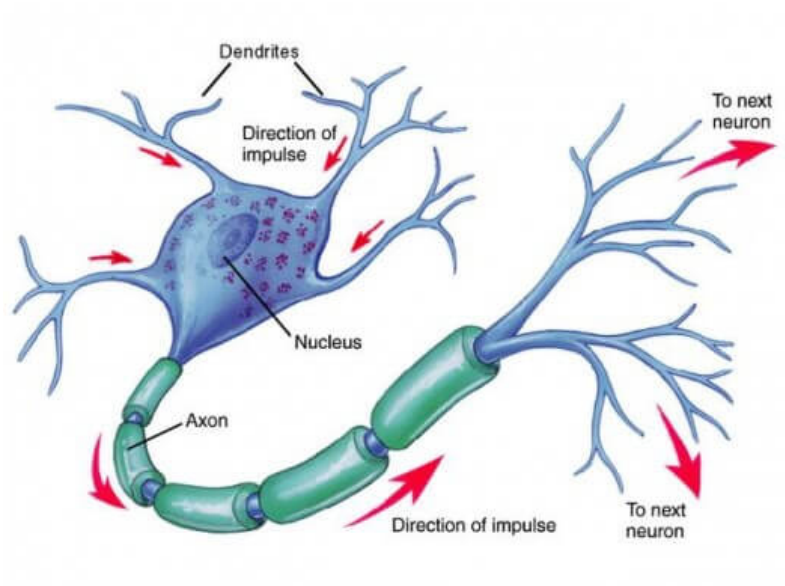
- Each function $f^{(k)}$ is of the form

$$f^{(k)} : \mathbb{R}^{d_k} \to \mathbb{R}^{d_{k+1}}$$

- d_k is the number of nodes in the kth layer. It is also called the *width* of the layer.
- We mostly assume for simplicity: $d_1 = \ldots = d_K =: d$.

Origin of the Name

If you look up the term “neuron” online, you will find illustrations like this:



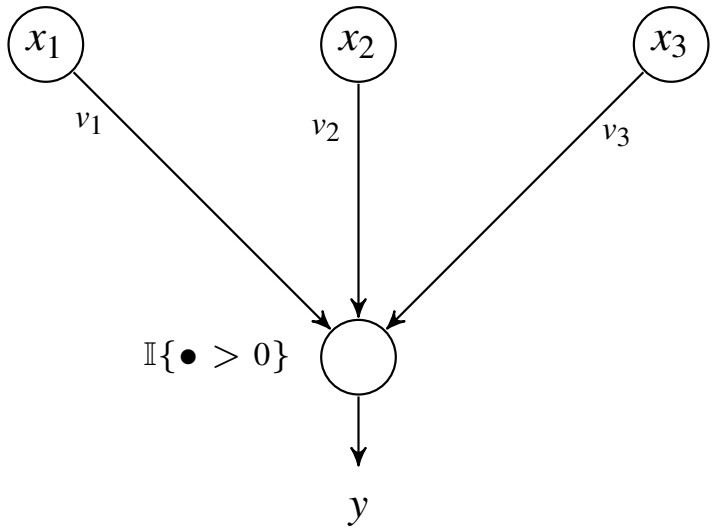
This one comes from a web site called easyscienceforkids.com, which means it is likely to be scientifically more accurate than typical references to “neuron” and “neural” in machine learning.

Roughly, a neuron is a brain cell that:

- Collects electrical signals (typically from other neurons)
- Processes them
- Generates an output signal

What happens inside a neuron is an intensely studied problem in neuroscience.

A neuron is modeled as a "thresholding device" that combines input signals:



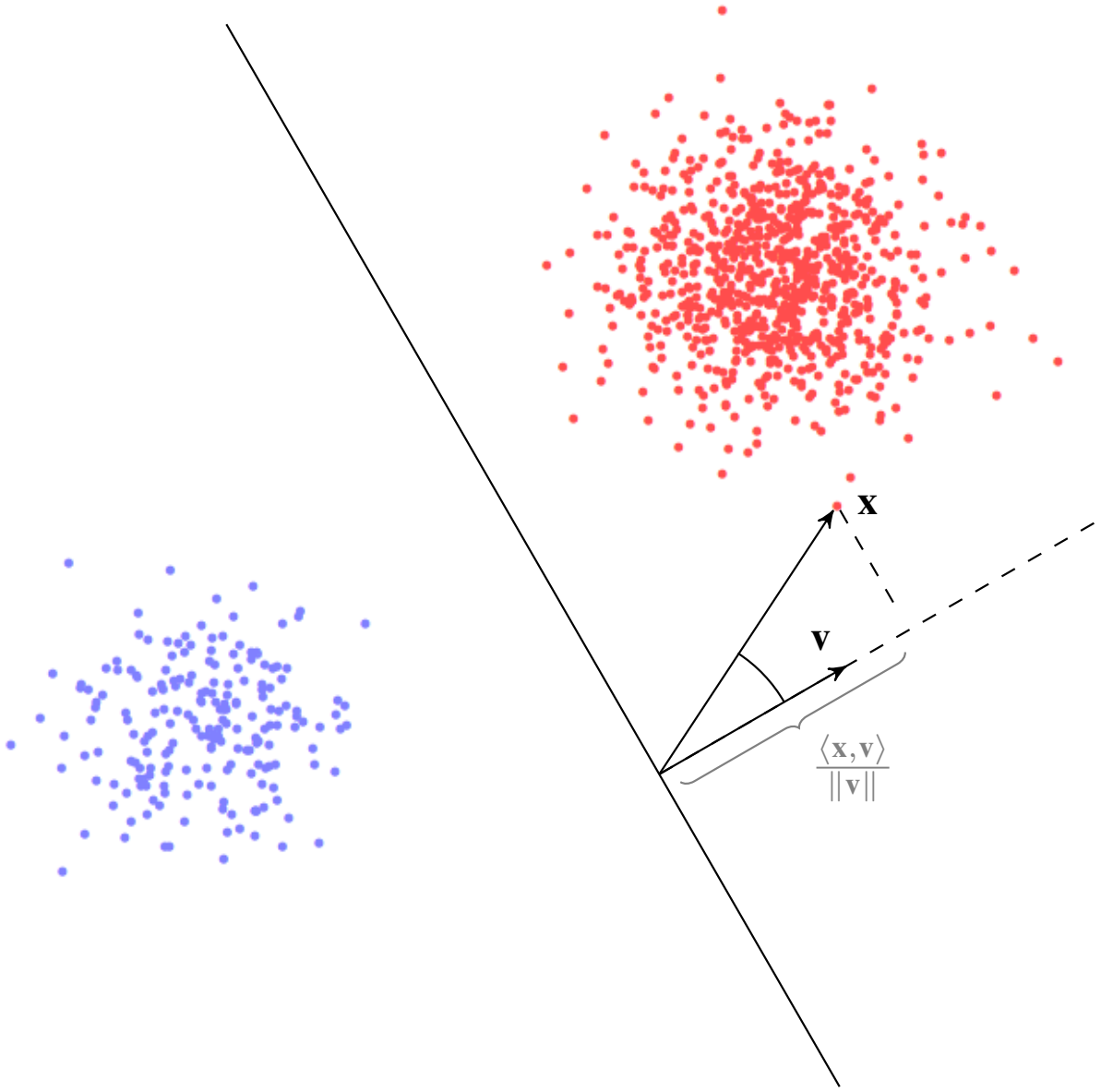
McCulloch-Pitts neuron model (1943)

- Collect the input signals x_1, x_2, x_3 into a vector $\mathbf{x} = (x_1, x_2, x_3) \in \mathbb{R}^3$
- Choose fixed vector $\mathbf{v} \in \mathbb{R}^3$ and constant $c \in \mathbb{R}$.
- Compute:

$$y = \mathbb{I}\{\langle \mathbf{v}, \mathbf{x} \rangle > 0\} \qquad \text{for some } c \in \mathbb{R} \ .$$

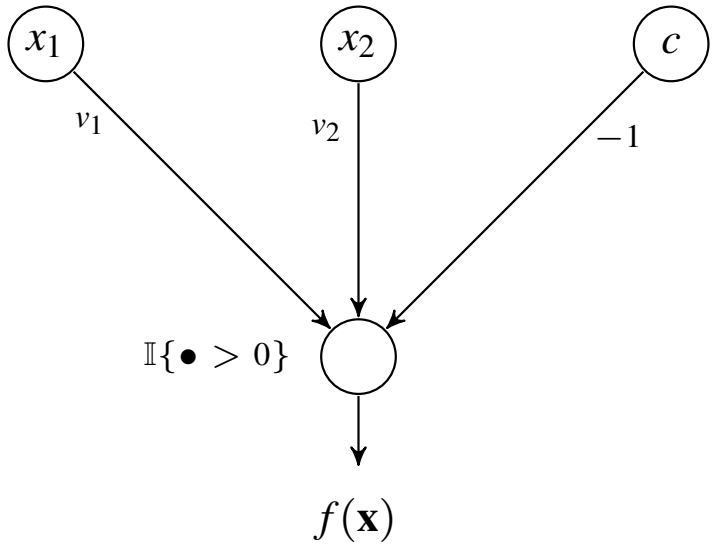
- In hindsight, this is a neural network with two layers, and function $\phi(\,\bullet\,) = \mathbb{I}\{\langle \mathbf{v}, \mathbf{x} \rangle > 0\}$ at the bottom unit.

Recall: Linear Classification



$$f(\mathbf{x}) = \text{sgn}(\langle \mathbf{v}, \mathbf{x} \rangle - c)$$

Linear Classifier in $\mathbb{R}^2$ as Two-Layer NN



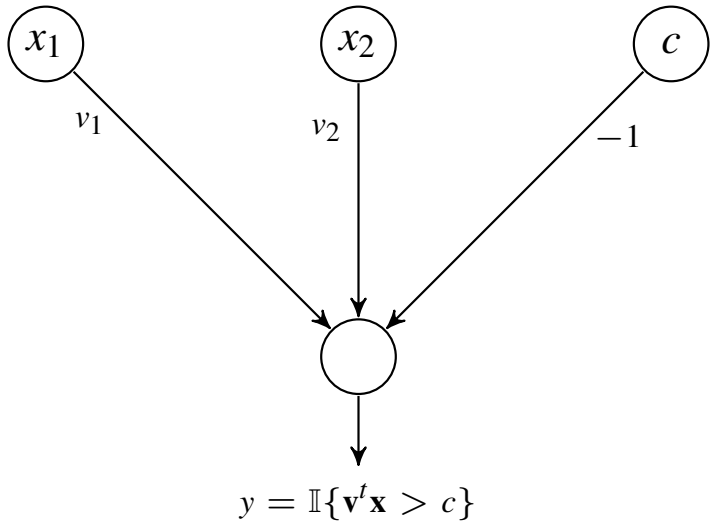
$$f(\mathbf{x}) \quad = \quad \mathbb{I}\{\ v_1 x_1 + v_2 x_2 + (-1)c \ > \ 0\ \} \quad = \quad \mathbb{I}\{\langle \mathbf{v}, \mathbf{x} \rangle > c\}$$

Equivalent to linear classifier

The linear classifier on the previous slide and f differ only in whether they encode the "blue" class as -1 or as 0:

$$\operatorname{sgn}(\langle \mathbf{v}, \mathbf{x} \rangle - c) \ = \ 2f(\mathbf{x}) - 1$$

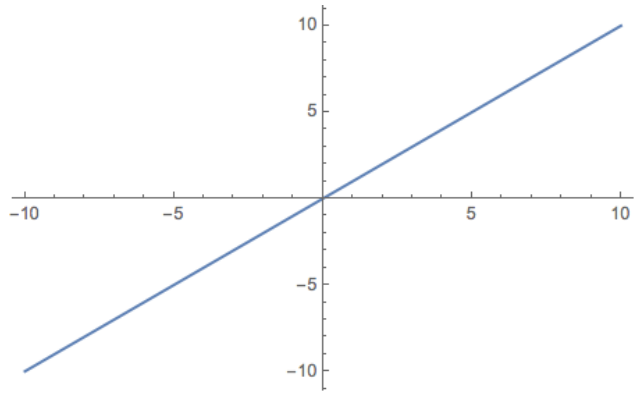
REMARKS



- This neural network represents a linear two-class classifier (on $\mathbb{R}^2$).
- We can more generally define a classifier on $\mathbb{R}^d$ by adding input units, one per dimension.
- It does not specify the training method.
- To train the classifier, we need a cost function and an optimization method.

Linear units

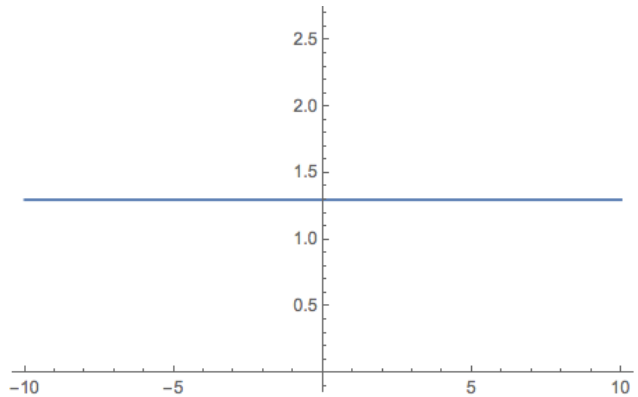
$$\phi(x) = x$$



This function simply "passes on" its incoming signal. These are used for example to represent inputs (data values).

Constant functions

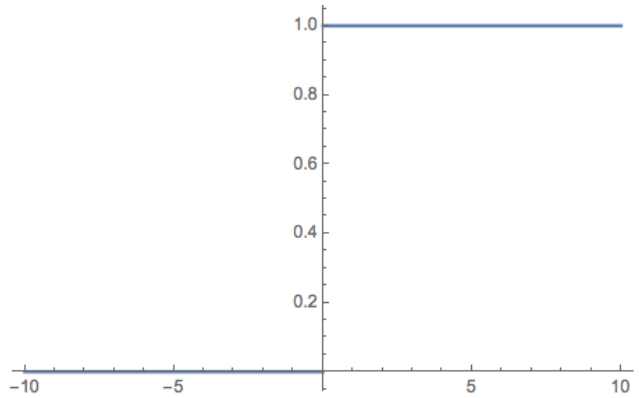
$$\phi(x) = c$$



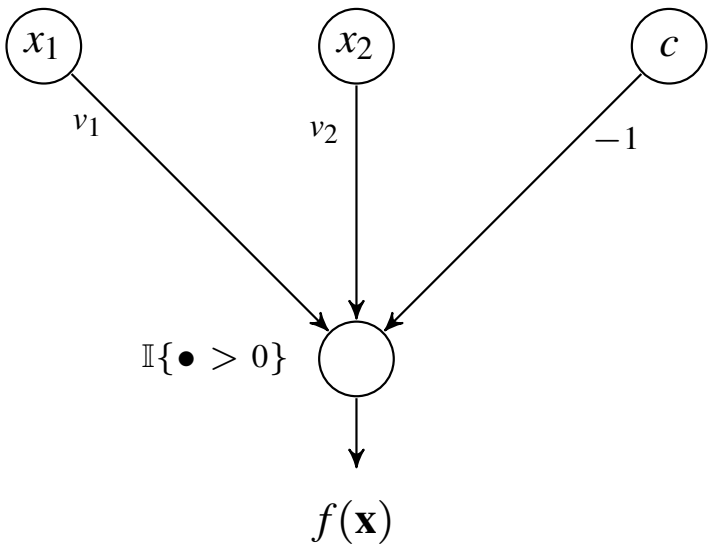
These can be used e.g. in combination with an indicator function to define a threshold, as in the linear classifier above.

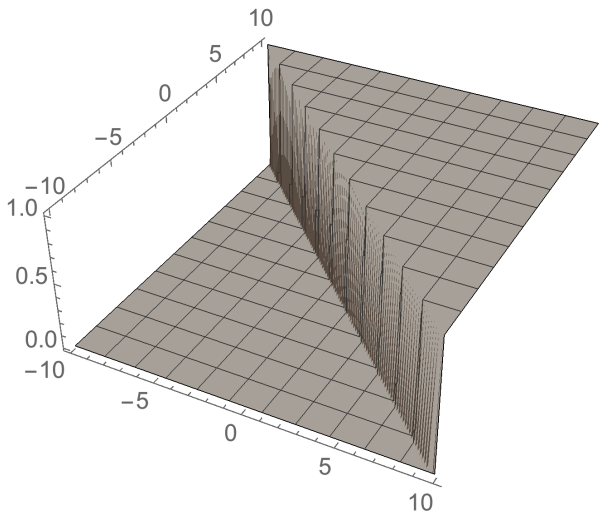
Indicator function

$$\phi(x) = \mathbb{I}\{x > 0\}$$



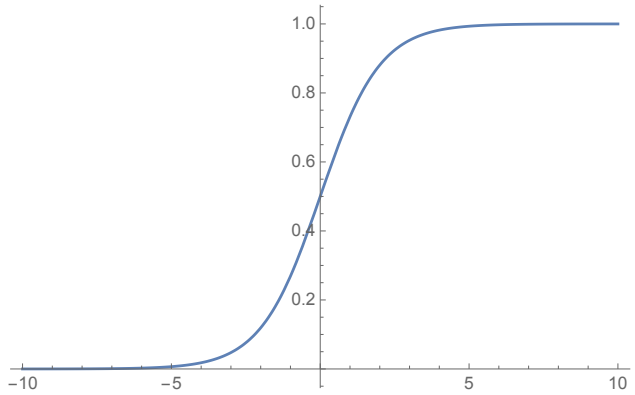
Example: Final unit is indicator



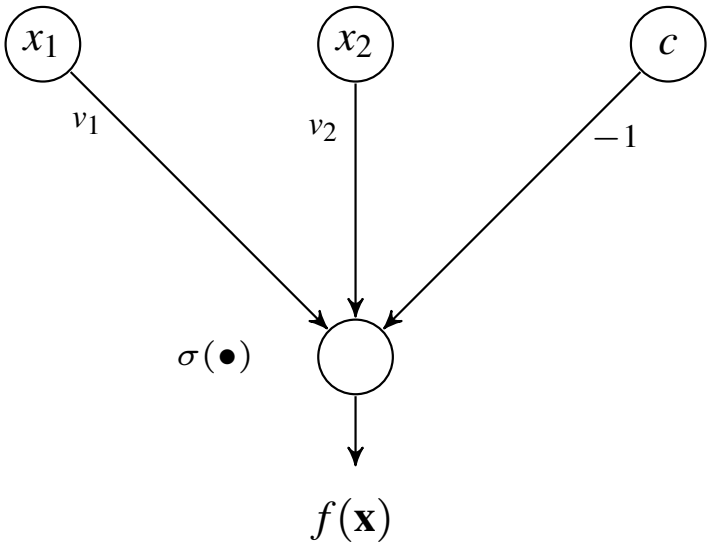


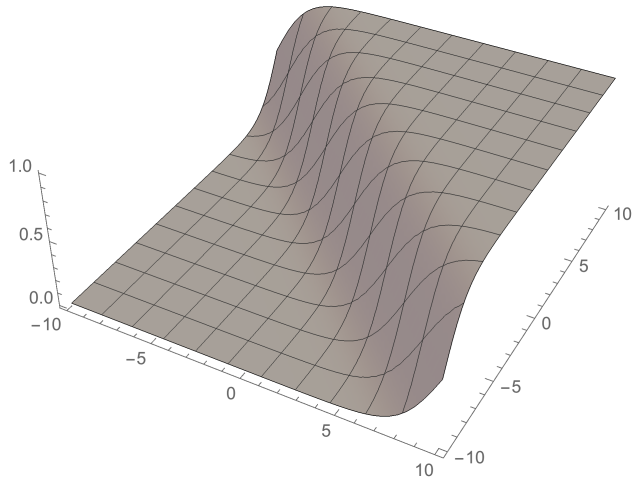
Sigmoids

$$\phi(x) = \frac{1}{1 + e^{-x}}$$



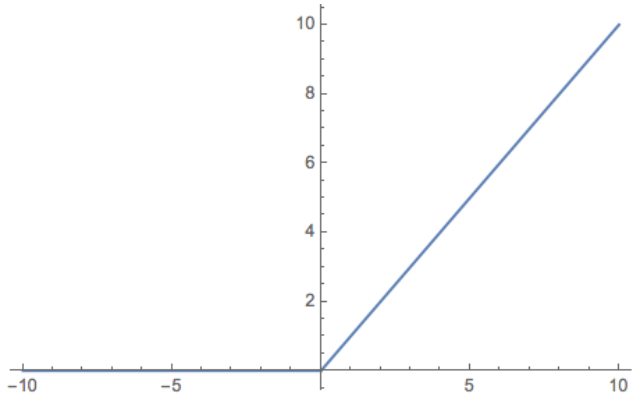
Example: Final unit is sigmoid





Rectified linear units

$$\phi(x) = \max\{0, x\}$$



These are currently perhaps the most commonly used unit in the "inner" layers of a neural network (those layers that are not the input or output layer).

Hidden units

- Any nodes (or "units") in the network that are neither input nor output nodes are called **hidden**.
- Every network has an input layer and an output layer.
- If there any additional layers (which hence consist of hidden units), they are called **hidden layers**.

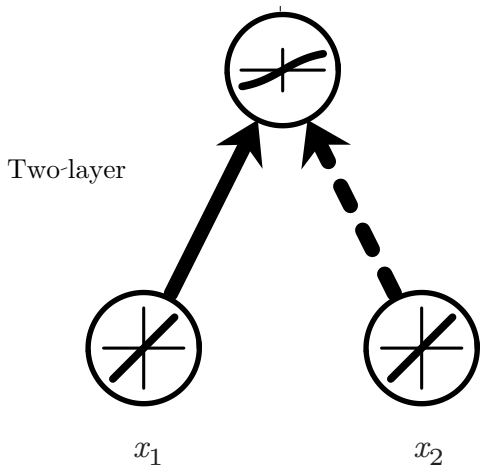
Linear and nonlinear networks

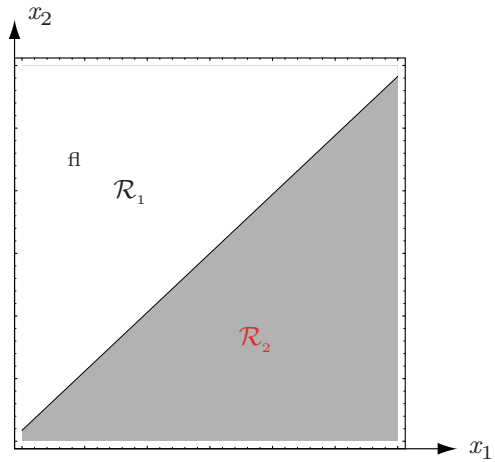
- If a network has no hidden units, then

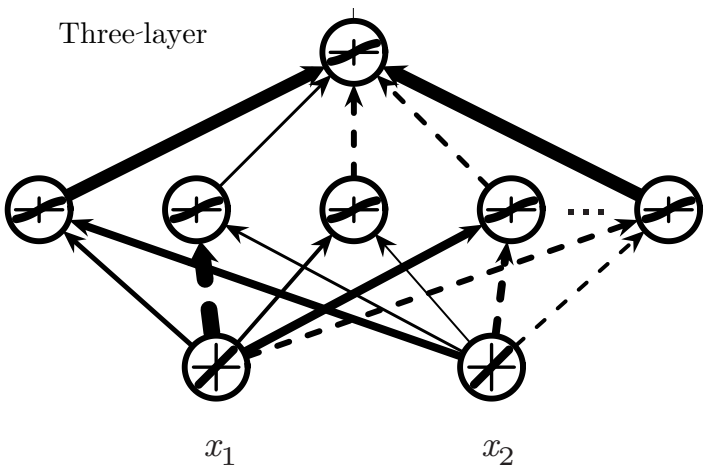
$$f_i(\mathbf{x}) = \phi_i(\langle \mathbf{w}_i, \mathbf{x} \rangle)$$

  That means: f is a linear functions, except perhaps for the final application of ϕ.
- For example: In a classification problem, a two layer network can only represent linear decision boundaries.
- Networks with at least one hidden layer can represent nonlinear decision surfaces.

Two vs Three Layers







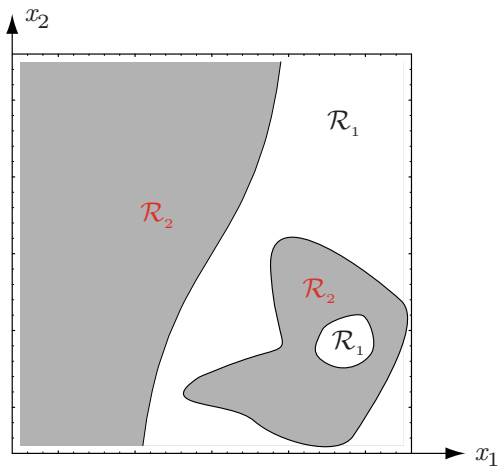
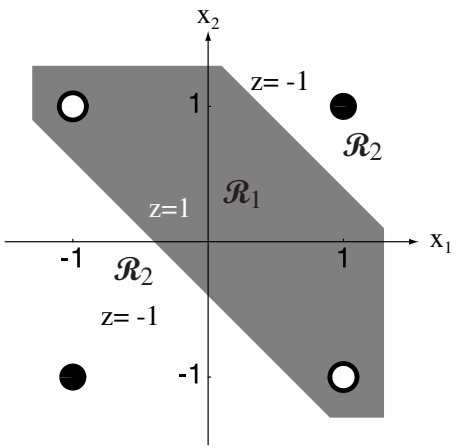
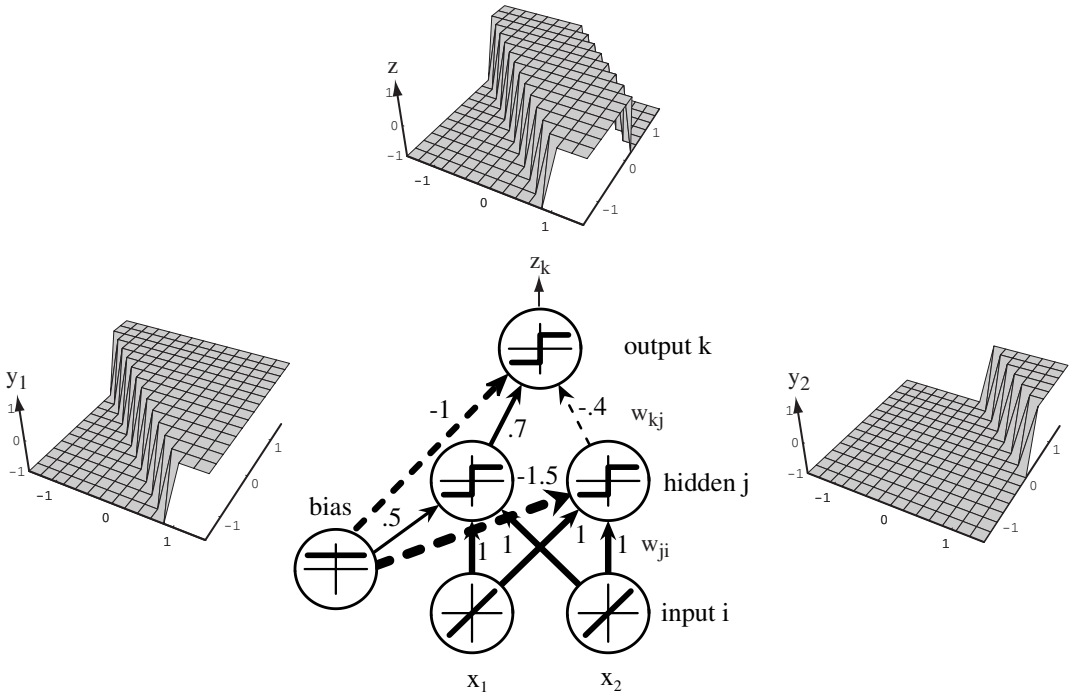


Illustration: R.O. Duda, P.E. Hart, D.G. Stork, *Pattern Classification*, Wiley 2001

THE XOR PROBLEM



Solution regions we would like to represent



Neural network representation

- Two ridges at different locations are substracted from each other.
- That generates a region bounded on both sides.
- A linear classifier cannot represent this decision region.
- Note this requires at least one hidden layer.

Illustration: R.O. Duda, P.E. Hart, D.G. Stork, *Pattern Classification*, Wiley 2001

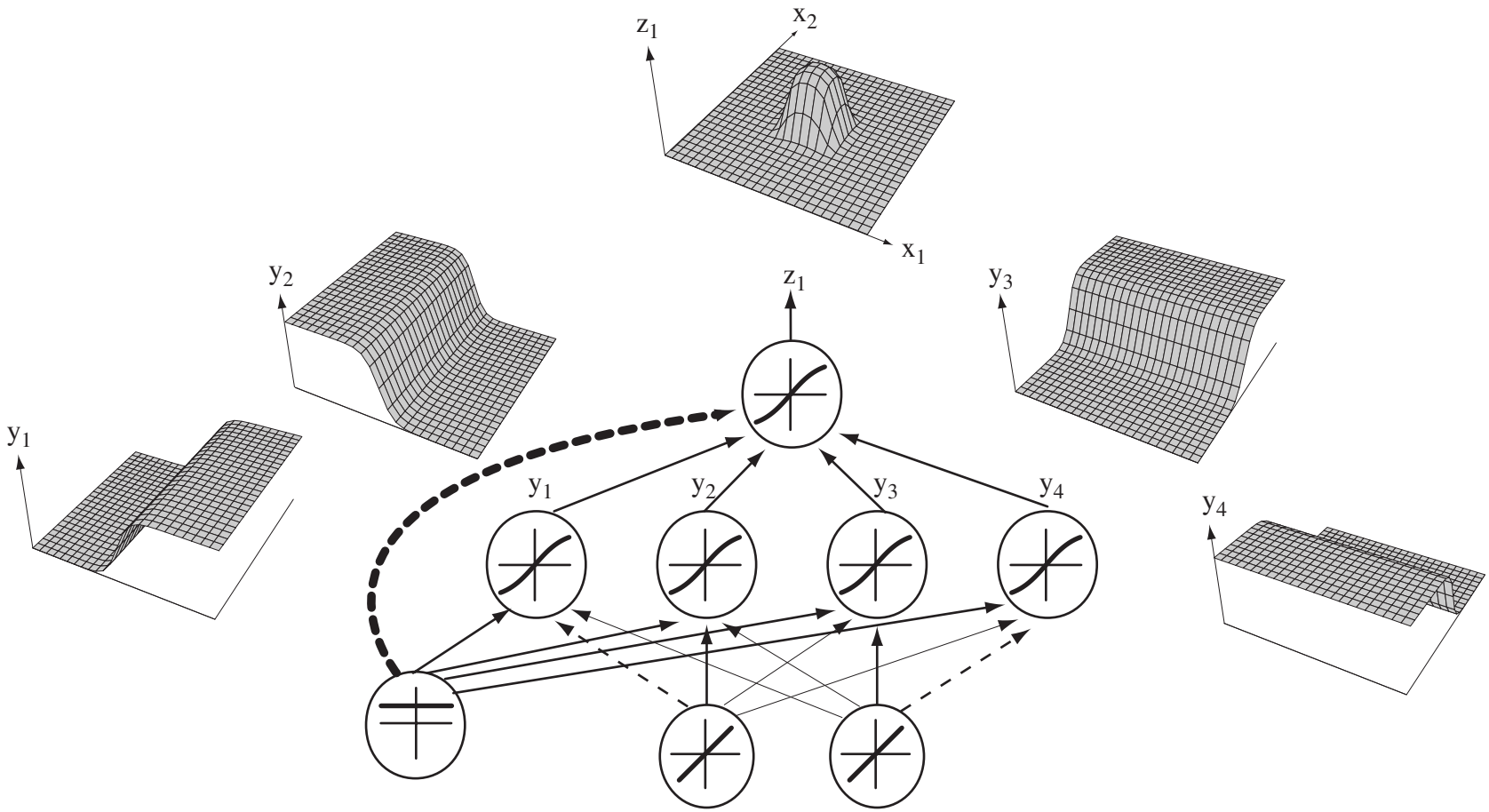


Illustration: R.O. Duda, P.E. Hart, D.G. Stork, *Pattern Classification*, Wiley 2001

We have observed

- We have seen that two-layer classification networks always represent linear class boundaries.
- With three layers, the boundaries can be non-linear.

Obvious question

- What happens if we use more than three layers? Do four layers again increase expressive power?

A neural network represents a (typically) complicated function f by simple functions $\phi_i^{(k)}$.

What functions can be represented?

A well-known result in approximation theory says: Every continuous function $f : [0,1]^d \to \mathbb{R}$ can be represented in the form

$$f(\mathbf{x}) = \sum_{j=1}^{2d+1} \xi_j \Big(\sum_{i=1}^{d} \tau_{ij}(x_i) \Big)$$

where ξ_i and τ_{ij} are functions $\mathbb{R} \to \mathbb{R}$. A similar result shows one can approximate f to arbitrary precision using specifically sigmoids, as

$$f(\mathbf{x}) \approx \sum_{j=1}^{M} w_j^{(2)} \sigma \Big(\sum_{i=1}^{d} w_{ij}^{(1)} x_i + c_i \Big)$$

for some finite M and constants c_i.

Note the representations above can both be written as neural networks with three layers (i.e. with one hidden layer).

Not examinable.

Depth rather than width

- The representations above can achieve arbitrary precision with a single hidden layer (roughly: a three-layer neural network can represent any continuous function).
- In the first representation, ξ_j and τ_{ij} are "simpler" than f because they map $\mathbb{R} \to \mathbb{R}$.
- In the second representation, the functions are more specific (sigmoids), and we typically need more of them (M is large).
- That means: The price of precision are many hidden units, i.e. the network grows wide.
- The last years have shown: We can obtain very good results by limiting layer width, and instead increasing depth (= number of layers).
- There is no coherent theory yet to properly explain this behavior.

Limiting width

- Limiting layer width means we limit the degrees of freedom of each function $f^{(k)}$.
- That is a notion of parsimony.
- Again: There seem to be a lot of interesting questions to study here, but so far, we have no real answers.

Task

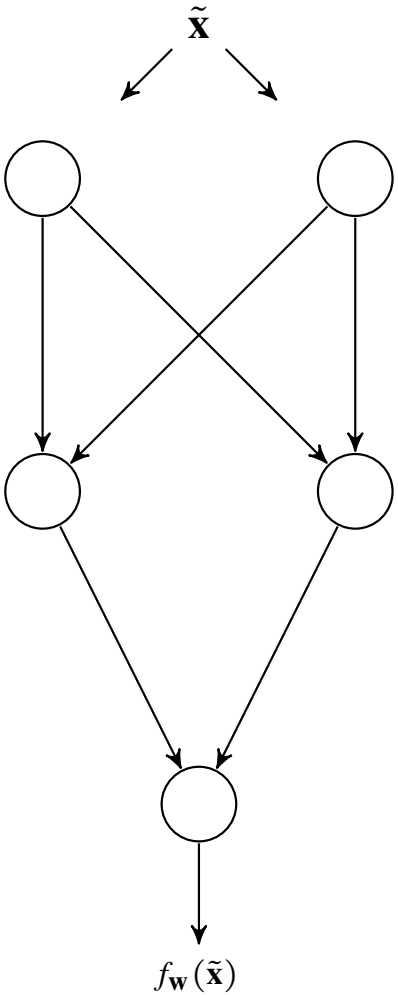
- We decide on a neural network "architecture": We fix the network diagram, including all functions ϕ at the units. Only the weights w on the edges can be changed during by training algorithm. Suppose the architecture we choose has d_1 input units and d_2 output units.
- We collect all weights into a vector $\mathbf{w}$. The entire network then represents a function $f_{\mathbf{w}}(\mathbf{x})$ that maps $\mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$.
- To "train" the network now means that, given training data, we have to determine a suitable parameter vector $\mathbf{w}$, i.e. we fit the network to data by fitting the weights.

More specifically: Classification

Suppose the network is meant to represent a two-class classifier.

- That means the output dimension is $d_2 = 1$, so $f_{\mathbf{w}}$ is a function $\mathbb{R}^{d_1} \to \mathbb{R}$.
- We are given data $\mathbf{x}_1, \mathbf{x}_2, \ldots$ with labels $y_1, y_2, \ldots$.
- We split this data into training, validation and test data, according to the requirements of the problem we are trying to solve.
- We then fit the network to the training data.

Training Neural Networks



- We run each training data point $\tilde{\mathbf{x}}_i$ through the network $f_{\mathbf{w}}$ and compare $f_{\mathbf{w}}(\tilde{\mathbf{x}}_i)$ to $\tilde{y}_i$ to measure the error.
- Recall how gradient descent works: We make "small" changes to $\mathbf{w}$, and choose the one which decreases the error most. That is one step of the gradient scheme.
- For each such changed value $\mathbf{w}'$, we again run each training data point $\tilde{\mathbf{x}}_i$ through the network $f_{\mathbf{w}'}$, and measure the error by comparing $f_{\mathbf{w}'}(\tilde{\mathbf{x}}_i)$ to $\tilde{y}_i$.

Error measure

- We have to specify how we compare the network's output $f_{\mathbf{w}}(\mathbf{x})$ to the correct answer y.
- To do so, we specify a function D with two arguments that serves as an error measure.
- The choice of D depends on the problem.

Typical error measures

- Classification problem:

$$D(\hat{y}, y) := y \log \hat{y} \qquad \text{(with convention } 0 \log 0 = 0\text{)}$$

- Regression problem:

$$D(\hat{y}, y) := \|y - \hat{y}\|^2$$

Training as an optimization problem

- Given: Training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ with labels y_i.
- We specify an error measure D, and define the total error on the training set as

$$J(\mathbf{w}) := \sum_{i=1}^{n} D(f_{\mathbf{w}}(\tilde{\mathbf{x}}_i), \tilde{y}_i)$$

Training problem

In summary, neural network training attempts to solve the optimization problem

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} J(\mathbf{w})$$

using gradient descent. For feed-forward networks, the gradient descent algorithm takes a specific form that is called *backpropagation*.

Backpropagation is gradient descent applied to $J(\mathbf{w})$ in a feed-forward network.

In practice: Stochastic gradient descent

- The vector $\mathbf{w}$ can be very high-dimensional. In high dimensions, computing a gradient is computationally expensive, because we have to make "small changes" to $\mathbf{w}$ in many different directions and compare them to each other.
- Each time the gradient algorithm computes $J(\mathbf{w}')$ for a changed value $\mathbf{w}'$, we have to apply the network to every data point, since $J(\mathbf{w}') = \sum_{i=1}^{n} D(f_{\mathbf{w}'}(\tilde{\mathbf{x}}_i), \tilde{y}_i)$.
- To save computation, the gradient algorithm typically computes $D(f_{\mathbf{w}'}(\tilde{\mathbf{x}}_i), \tilde{y}_i)$ only for some small subset of a the training data. This subset is called a *mini batch*, and the resulting algorithm is called **stochastic gradient descent**.

Neural network training optimization problem

$$\min_{\mathbf{w}} J(\mathbf{w})$$

The application of gradient descent to this problem is called *backpropagation*.

Backpropagation is gradient descent applied to $J(\mathbf{w})$ in a feed-forward network.

Deriving backpropagation

- We have to evaluate the derivative $\nabla_{\mathbf{w}} J(\mathbf{w})$.
- Since J is additive over training points, $J(\mathbf{w}) = \sum_n J_n(\mathbf{w})$, it suffices to derive $\nabla_{\mathbf{w}} J_n(\mathbf{w})$.

The next few slides were written for a different class, and you are not expected to know their content. I show them only to illustrate the interesting way in which gradient descent interleaves with the feed-forward architecture.

Deriving backpropagation

- We have to evaluate the derivative $\nabla_{\mathbf{w}} J(\mathbf{w})$.
- Since J is additive over training points, $J(\mathbf{w}) = \sum_n J_n(\mathbf{w})$, it suffices to derive $\nabla_{\mathbf{w}} J_n(\mathbf{w})$.

Not examinable.

Recall from calculus: Chain rule

Consider a composition of functions $f \circ g(x) = f(g(x))$.

$$\frac{d(f \circ g)}{dx} = \frac{df}{dg}\frac{dg}{dx}$$

If the derivatives of f and g are f' and g', that means: $\frac{d(f \circ g)}{dx}(x) = f'(g(x))g'(x)$

Application to feed-forward network

Let $\mathbf{w}^{(k)}$ denote the weights in layer k. The function represented by the network is

$$f_{\mathbf{w}}(\mathbf{x}) \;=\; f_{\mathbf{w}}^{(K)} \circ \cdots \circ f_{\mathbf{w}}^{(1)}(\mathbf{x}) \;=\; f_{\mathbf{w}^{(K)}}^{(K)} \circ \cdots \circ f_{\mathbf{w}^{(1)}}^{(1)}(\mathbf{x})$$

To solve the optimization problem, we have to compute derivatives of the form

$$\frac{d}{d\mathbf{w}} D(f_{\mathbf{w}}(\mathbf{x}_n), y_n) = \frac{dD(\,\bullet\,, y_n)}{df_{\mathbf{w}}}\frac{df_{\mathbf{w}}}{d\mathbf{w}}$$

Not examinable.

Decomposing the Derivatives

- The chain rule means we compute the derivates layer by layer.
- Suppose we are only interested in the weights of layer k, and keep all other weights fixed. The function f represented by the network is then

$$f_{\mathbf{w}^{(k)}}(\mathbf{x}) \;=\; f^{(K)} \circ \cdots \circ f^{(k+1)} \circ f^{(k)}_{\mathbf{w}^{(k)}} \circ f^{(k-1)} \circ \cdots \circ f^{(1)}(\mathbf{x})$$

- The first $k-1$ layers enter only as the function value of $\mathbf{x}$, so we define

$$\mathbf{z}^{(k)} := f^{(k-1)} \circ \cdots \circ f^{(1)}(\mathbf{x})$$

  and get

$$f_{\mathbf{w}^{(k)}}(\mathbf{x}) \;=\; f^{(K)} \circ \cdots \circ f^{(k+1)} \circ f^{(k)}_{\mathbf{w}^{(k)}}(\mathbf{z}^{(k)})$$

- If we differentiate with respect to $\mathbf{w}^{(k)}$, the chain rule gives

$$\frac{d}{d\mathbf{w}^{(k)}} f_{\mathbf{w}^{(k)}}(\mathbf{x}) \;=\; \frac{df^{(K)}}{df^{(K-1)}} \cdots \frac{df^{(k+1)}}{df^{(k)}} \;\cdot\; \frac{df^{(k)}_{\mathbf{w}^{(k)}}}{d\mathbf{w}^{(k)}}$$

Not examinable.

- Each $f^{(k)}$ is a vector-valued function $f^{(k)} : \mathbb{R}^{d_k} \to \mathbb{R}^{d_{k+1}}$.
- It is parametrized by the weights $\mathbf{w}^{(k)}$ of the kth layer and takes an input vector $\mathbf{z} \in \mathbb{R}^{d_k}$.
- We write $f^{(k)}(\mathbf{z}, \mathbf{w}^{(k)})$.

Layer-wise derivative

Since $f^{(k)}$ and $f^{(k-1)}$ are vector-valued, we get a Jacobian matrix

$$\frac{df^{(k+1)}}{df^{(k)}} = \begin{pmatrix} \frac{\partial f_1^{(k+1)}}{\partial f_1^{(k)}} & \cdots & \frac{\partial f_1^{(k+1)}}{\partial f_{d_k}^{(k)}} \\ \vdots & & \vdots \\ \frac{\partial f_{d_{k+1}}^{(k+1)}}{\partial f_1^{(k)}} & \cdots & \frac{\partial f_{d_{k+1}}^{(k+1)}}{\partial f_{d_k}^{(k)}} \end{pmatrix} =: \Delta^{(k)}(\mathbf{z}, \mathbf{w}^{(k+1)})$$

- $\Delta^{(k)}$ is a matrix of size $d_{k+1} \times d_k$.
- The derivatives in the matrix quantify how $f^{(k+1)}$ reacts to changes in the argument of $f^{(k)}$ if the weights $\mathbf{w}^{(k+1)}$ and $\mathbf{w}^{(k)}$ of both functions are fixed.

Let $\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(K)}$ be the current settings of the layer weights. These have either been computed in the previous iteration, or (in the first iteration) are initialized at random.

Step 1: Forward pass

We start with an input vector $\mathbf{x}$ and compute

$$\mathbf{z}^{(k)} := f^{(k)} \circ \cdots \circ f^{(1)}(\mathbf{x})$$

for all layers k.

Step 2: Backward pass

- Start with the last layer. Update the weights $\mathbf{w}^{(K)}$ by performing a gradient step on

  $$D\big(f^{(K)}(\mathbf{z}^{(K)}, \mathbf{w}^{(K)}), y\big)$$

  regarded as a function of $\mathbf{w}^{(K)}$ (so $\mathbf{z}^{(K)}$ and y are fixed). Denote the updated weights $\tilde{\mathbf{w}}^{(K)}$.
- Move backwards one layer at a time. At layer k, we have already computed updates $\tilde{\mathbf{w}}^{(K)}, \ldots, \tilde{\mathbf{w}}^{(k+1)}$. Update $\mathbf{w}^{(k)}$ by a gradient step, where the derivative is computed as

  $$\Delta^{(K-1)}(\mathbf{z}^{(K-1)}, \tilde{\mathbf{w}}^{(K)}) \cdot \ldots \cdot \Delta^{(k)}(\mathbf{z}^{(k)}, \tilde{\mathbf{w}}^{(k+1)}) \frac{df^{(k)}}{d\mathbf{w}^{(k)}}(\mathbf{z}, \mathbf{w}^{(k)})$$

On reaching level 1, go back to step 1 and recompute the $\mathbf{z}^{(k)}$ using the updated weights.

Not examinable.

SUMMARY: BACKPROPAGATION

- Backpropagation is a gradient descent method for the optimization problem

$$\min_{\mathbf{w}} J(\mathbf{w}) = \sum_{i=1}^{N} D(f_{\mathbf{w}}(\mathbf{x}_i), y_i)$$

  D must be chosen such that it is additive over data points.

- It alternates between forward passes that update the layer-wise function values $\mathbf{z}^{(k)}$ given the current weights, and backward passes that update the weights using the current $\mathbf{z}^{(k)}$.
- The layered architecture means we can (1) compute each $\mathbf{z}^{(k)}$ from $\mathbf{z}^{(k-1)}$ and (2) we can use the weight updates computed in layers $K, \ldots, k+1$ to update weights in layer k.

Features

- Raw measurement data is typically not used directly as input for a learning algorithm. Some form of preprocessing is applied first.
- We can think of this preprocessing as a function, e.g.

$$\mathbf{F}\text{: raw data space} \longrightarrow \mathbb{R}^d$$

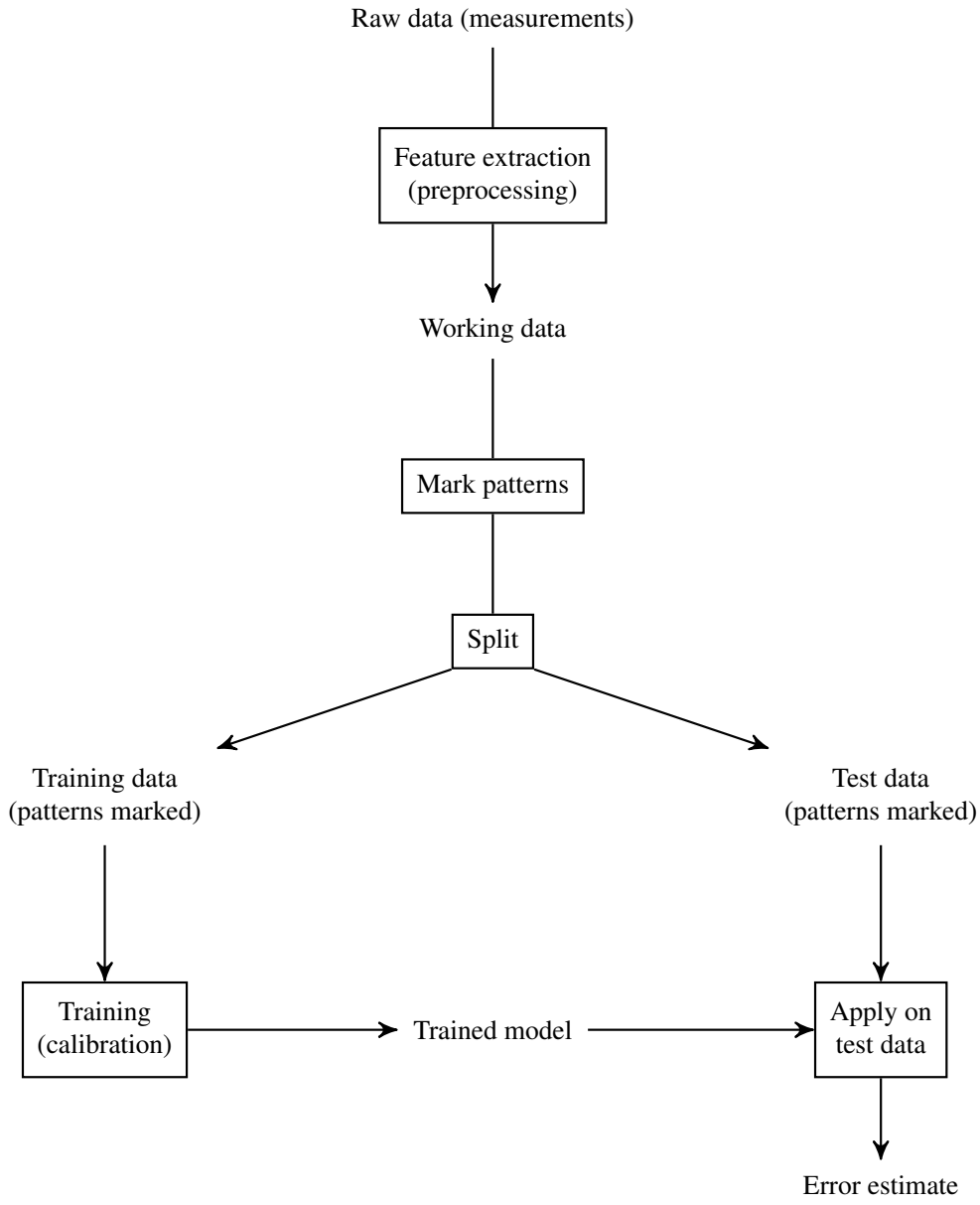
  ($\mathbb{R}^d$ is only an example, but a very common one.)
- If the raw measurements are $\mathbf{m}_1, \ldots, \mathbf{m}_N$, the data points which are fed into the learning algorithm are the images $\mathbf{x}_n := \mathbf{F}(\mathbf{m}_n)$.

Terminology

- **F** is called a **feature map**.
- Its dimensions (the dimensions of its range space) are called **features**.
- The preprocessing step (= application of **F** to the raw data) is called **feature extraction**.

Example Processing Pipeline

This is what a typical processing pipeline for a supervided learning propblem might look like.



Where does learning start?

- It is often a matter of definition where feature extraction stops and learning starts.
- If we have a perfect feature extractor, learning is trivial.
- For example:
  - Consider a classfication problem with two classes.
  - Suppose the feature extractor maps the raw data measurements of class 1 to a single point, and all data points in class to to a single distinct point.
  - Then classification is trivial.
  - That is of course what the classifier is supposed to do in the end (e.g. map to the points 0 and 1).

Multi-layer networks and feature extraction

- An interesting aspect of multi-layer neural networks is that their early layers can be intepreted as feature extraction.
- For certain types of problems (e.g. computer vision), features were long "hand-tuned" by humans.
- Features extracted by neural networks give much better results.
- Several important problems, such as object recognition and face recognition, have basically been solved in this way.

Deep Networks as Feature Extractors

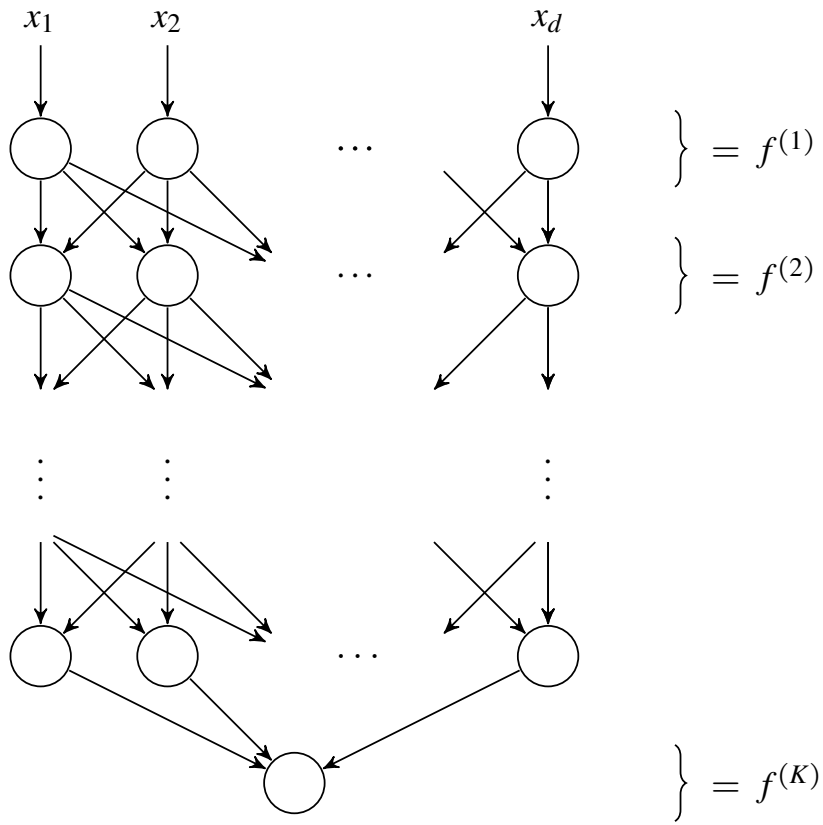
- The network on the right is a classifier $f : \mathbf{R}^d \to \{0, 1\}$.
- Suppose we subdivide the network into the first $K - 1$ layer and the final layer, by defining

$$\mathbf{F}(\mathbf{x}) := f^{(K-1)} \circ \ldots \circ f^{(1)}(\mathbf{x})$$

- The entire network is then

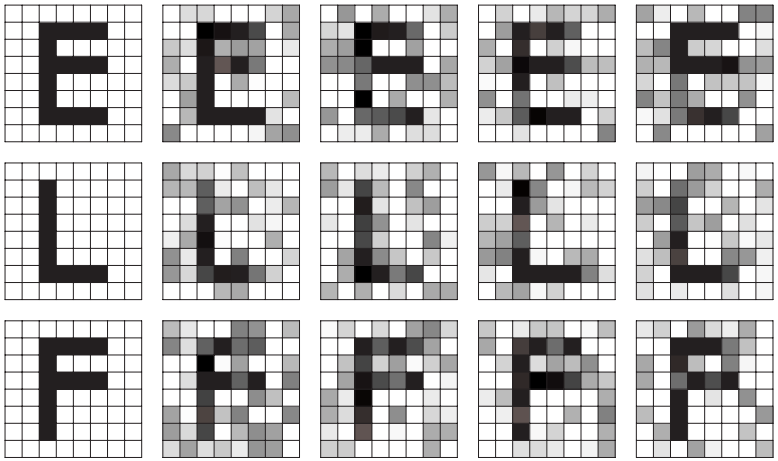
$$f(\mathbf{x}) = f^{(K)} \circ \mathbf{F}(\mathbf{x})$$

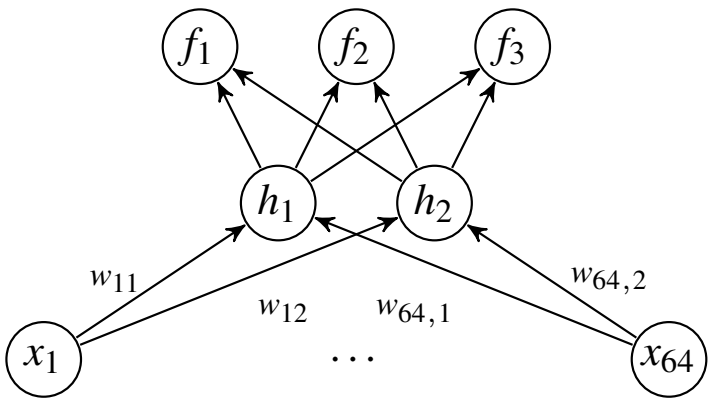
- The function $f^{(K)}$ is a two-class logistic regression classifier.
- We can hence think of f as a feature extraction $\mathbf{F}$ followed by linear classification $f^{(K)}$.



$$f^{(K)}(\bullet) = \sigma(\langle \mathbf{w}^{(K)}, \bullet \rangle)$$

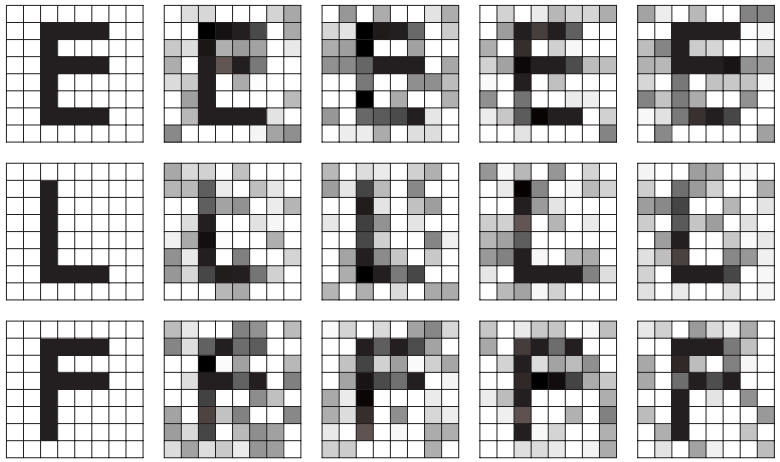
A Simple Example



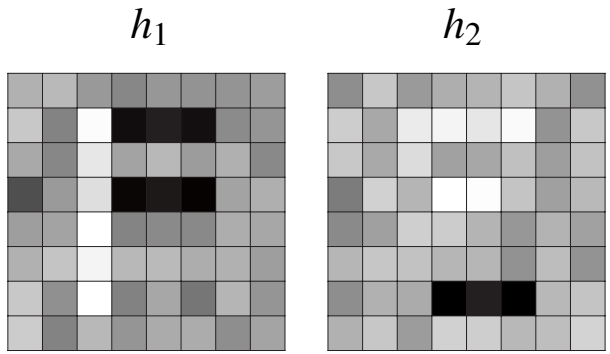


- Problem: Classify characters into three classes (E, F and L).
- Each digit given as a $8 \times 8 = 64$ pixel image
- Neural network: 64 input units (=pixels)
- 2 hidden units
- 3 binary output units, where $f_i(\mathbf{x}) = 1$ means image is in class i.
- Each hidden unit has 64 input weights, one per pixel. The weight values can be plottes as 8×8 images.

A Simple Example



training data (with random noise)



weight values of h_1 and h_2 plotted as images

- Dark regions = large weight values.
- Note the weights emphasize regions that distinguish characters.
- We can think of weight (= each pixel) as a feature.
- The features with large weights for h_1 distinguish {E,F} from L.
- The features for h_2 distinguish {E,L} from F.

Illustration: R.O. Duda, P.E. Hart, D.G. Stork, *Pattern Classification*, Wiley 2001

Example: Autoencoders

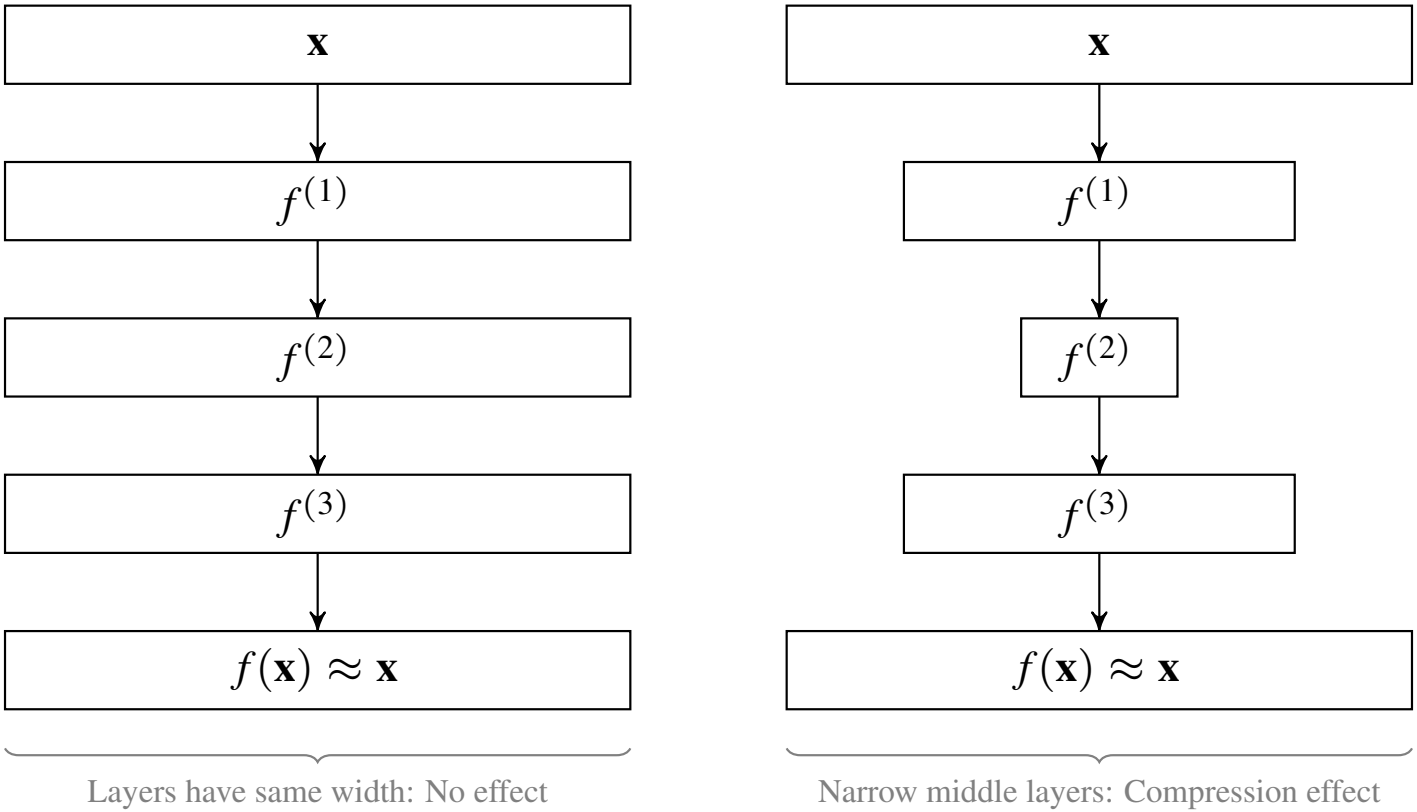
An example for the effect of layer are autoencoders.

- An **autoencoder** is a neural network that is trained on its own input: If the network has weights $\mathbf{W}$ and represents a function $f_{\mathbf{W}}$, training solves the optimization problem

  $$\min_{\mathbf{W}} \|\mathbf{x} - f_{\mathbf{W}}(\mathbf{x})\|^2$$

  or something similar for a different norm.
- That seems pointless at first glance: The network tries to approximate the identity function using its (possibly nonlinear) component functions.
- However: If the layers in the middle have much fewer nodes that those at the top and bottom, the network learns to *compress the input*.

AUTOENCODERS



- Train network on many images.
- Once trained: Input an image $\mathbf{x}$.
- Store $\mathbf{x}' := f^{(2)}(\mathbf{x})$. Note $\mathbf{x}'$ has fewer dimensions than $\mathbf{x} \rightarrow$ compression.
- To decompress $\mathbf{x}'$: Input it into $f^{(3)}$ and apply the remaining layers of the network $\rightarrow$ reconstruction $f(\mathbf{x}) \approx \mathbf{x}$ of $\mathbf{x}$.

AUTOENCODERS

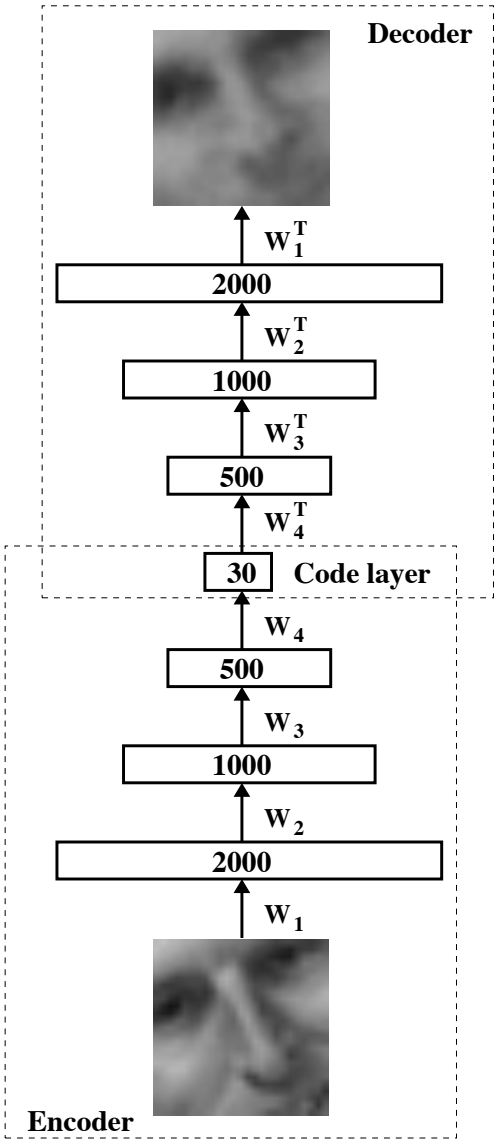


Illustration: K. Murphy, *Machine Learning: A Bayesian perspective*, MIT Press 2012

Definition

Suppose we define a small (here: 3×3) matrix

$$K = \begin{pmatrix} k_{-1,-1} & k_{-1,0} & k_{-1,1} \\ k_{0,-1} & k_{0,0} & k_{0,1} \\ k_{1,-1} & k_{1,0} & k_{1,1} \end{pmatrix}$$

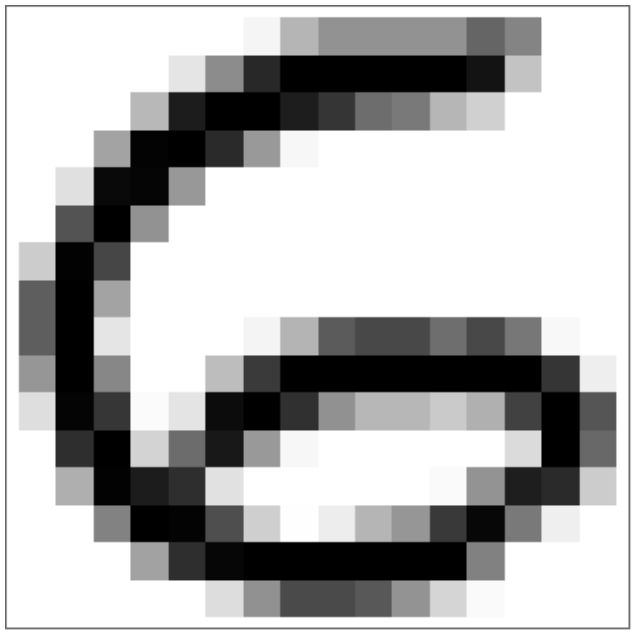
For a large matrix A, we define the **cross-correlation** of A and K as the matrix $A \odot K$ with entries

$$(A \odot K)_{ij} := a_{ij}k_{0,0} + a_{i-1,j-1}k_{-1,-1} + \ldots = \sum_{m,n=-1}^{1} a_{i+m,j+n}k_{m,n}$$

Remarks

- K is sometimes called a **kernel**. Caution: The term kernel is used for several, different concepts in both mathematics and machine learning.
- We can similarly define the cross-correlation if K is of size 5×5 etc. The numbers of rows and columns should be odd, so that k_{00} is at the center of K.

Cross-correlation for Images

$A =$  $=$

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 10 & 74 & 109 & 109 & 109 & 109 & 154 & 123 & 0 & 0 \\
0 & 0 & 0 & 0 & 26 & 115 & 215 & 255 & 255 & 255 & 255 & 255 & 236 & 60 & 0 & 0 \\
0 & 0 & 0 & 71 & 227 & 255 & 255 & 226 & 202 & 146 & 134 & 73 & 47 & 0 & 0 & 0 \\
0 & 0 & 92 & 252 & 255 & 213 & 102 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 31 & 246 & 250 & 103 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 172 & 255 & 109 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
51 & 253 & 185 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
161 & 255 & 92 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
161 & 255 & 26 & 0 & 0 & 0 & 11 & 75 & 164 & 183 & 183 & 145 & 183 & 136 & 7 & 0 \\
105 & 255 & 120 & 0 & 0 & 66 & 197 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 202 & 17 \\
33 & 251 & 201 & 4 & 27 & 243 & 255 & 207 & 110 & 72 & 72 & 53 & 79 & 190 & 255 & 170 \\
0 & 209 & 255 & 44 & 147 & 231 & 102 & 8 & 0 & 0 & 0 & 0 & 0 & 36 & 255 & 151 \\
0 & 80 & 253 & 227 & 209 & 30 & 0 & 0 & 0 & 0 & 0 & 5 & 108 & 225 & 213 & 51 \\
0 & 0 & 125 & 255 & 252 & 177 & 48 & 1 & 18 & 74 & 105 & 198 & 248 & 134 & 16 & 0 \\
0 & 0 & 0 & 93 & 209 & 249 & 255 & 255 & 255 & 255 & 255 & 255 & 126 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 34 & 110 & 181 & 181 & 167 & 108 & 42 & 5 & 0 & 0 & 0
\end{pmatrix}$$

- Recall that we can represent a grayscale image as a matrix A.
- We can then define a kernel matrix K and compute the cross-correlation $A \odot K$.

EFFECT OF CROSS-CORRELATION ON IMAGES

- Consider again a 3×3 kernel

$$K = \begin{pmatrix} k_{-1,-1} & k_{-1,0} & k_{-1,1} \\ k_{0,-1} & k_{0,0} & k_{0,1} \\ k_{1,-1} & k_{1,0} & k_{1,1} \end{pmatrix} \quad \text{with} \quad (A \odot K)_{ij} = \sum_{m,n=-1}^{1} a_{i+m,\, j+n} k_{m,n}$$

- Consider the pixel value a_{ij} at location i, j in A. In the new image $A \odot K$, a_{ij} is the sum of element-wise producs of K and the direct neighborhood of a_{ij}:

$$(A \odot K)_{ij} = \text{ sum of entries of } \begin{pmatrix} k_{-1,-1}a_{i-1,j-1} & k_{-1,0}a_{i-1,j} & k_{-1,1}a_{i-1,j+1} \\ k_{0,-1}a_{i,j-1} & k_{0,0}a_{ij} & k_{0,1}a_{i,j+1} \\ k_{1,-1}a_{i+1,j-1} & k_{1,0}a_{i+1,j} & k_{1,1}a_{i+1,j+1} \end{pmatrix}$$

- In other words, $(A \odot K)_{ij}$ is a weighted average of a_{ij} and its neighbors.
- The next few slides illustrate the effect of different choices of K.

EXAMPLES

For the identity kernel, nothing happens:

$A =$ 

$$K = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$A \odot K =$ 

EXAMPLES

If all entries of K are identical, each pixel in the image is "averaged together" with its neighbors. That results in blurring:

$A =$ 

$$K = \begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$$

$A \odot K =$ 

EXAMPLES

Since diagonal neighbors are further away than horizontal/vertical ones, we can give them smaller weights. This is also called a "Gaussian blur":

$A =$

$$K = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

$A \odot K =$

We can increase the size of K, which means we are mixing a_{ij} with more neighbors. Here is a 5×5 Gaussian blur:

$A =$ 

$$K = \frac{1}{256}\begin{pmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix}$$

$A \odot K =$ 

The opposite effect is sharpening: We give the neighbors negative weights. If two adjacent points look different, $A \odot K$ substracts them from each other, so they look even more different:

$A =$ 

$$K = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

$A \odot K =$ 

Note the entries of K add up to 1.

A more drastic form of sharpening is edge detection:

$A =$ 

$$K = \begin{pmatrix} -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} \\ -\frac{1}{8} & 1 & -\frac{1}{8} \\ -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} \end{pmatrix}$$

$A \odot K =$ 

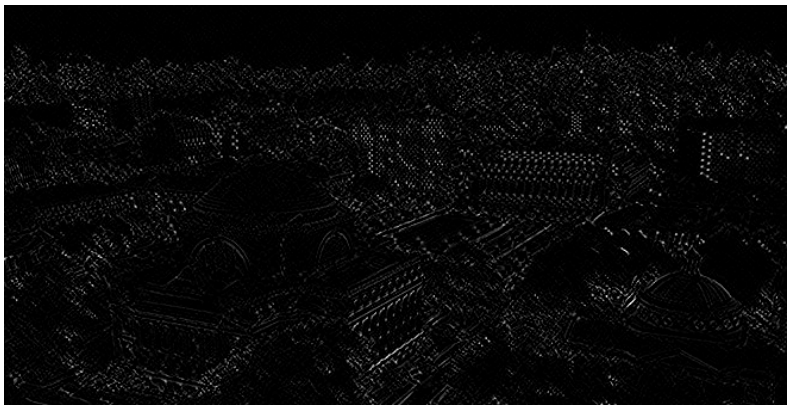
Here, the entries of K add up to 0, so $(A \odot K)_{ij}$ is visible only if a_{ij} is very different from its neighbors.

EXAMPLES

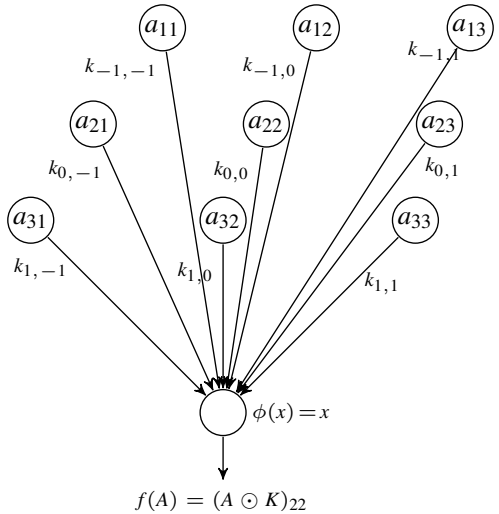
This kernel find points that are similar to their lower left and upper right neighbor, and different from their upper left and lower right one. That means it detects diagonal edges:

$A =$ 

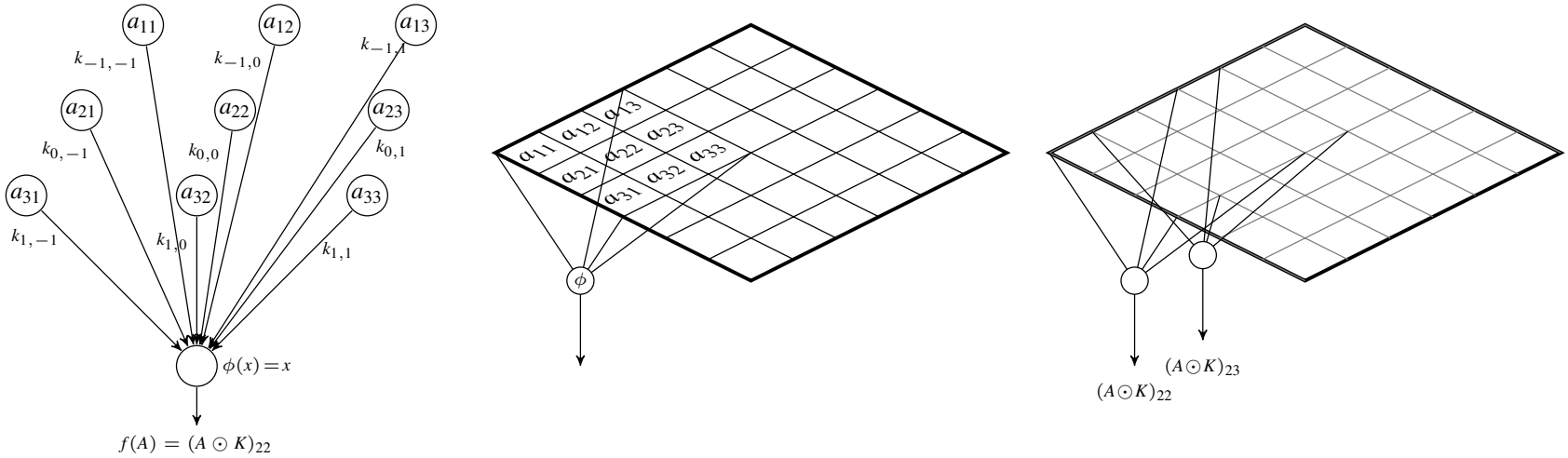
$$K = \begin{pmatrix} -1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & -1 \end{pmatrix}$$

$A \odot K =$ 

Cross-correlation as a Neural Network

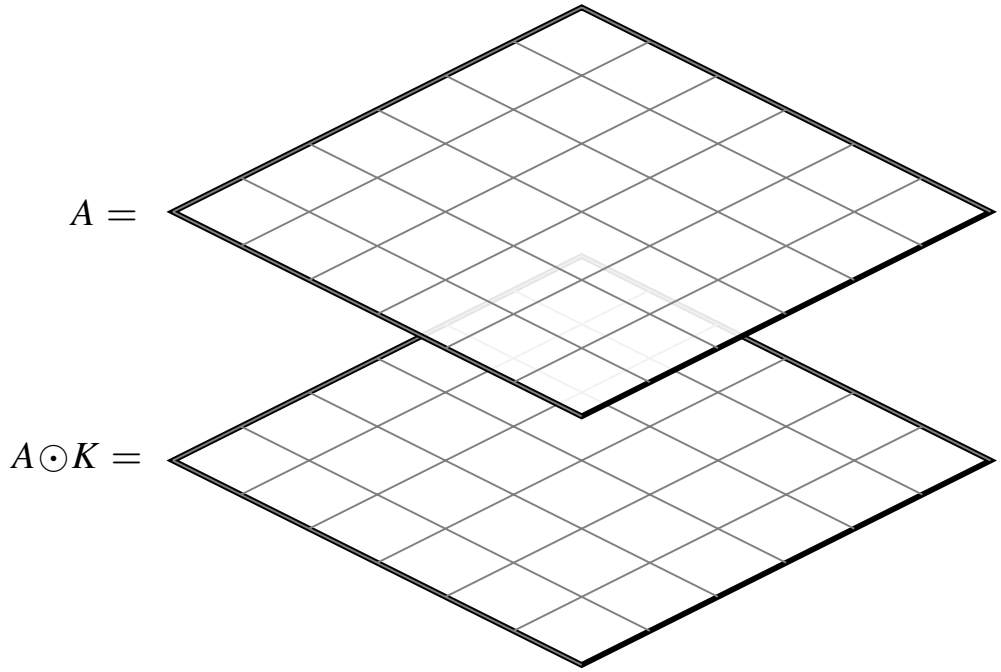


- Suppose we build a neural network one input unit for each entry of $\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$.
- We use the entries of K as weights and connect everything to a single linear unit ("linear unit" means $\phi(x) = x$).
- The network then computes the sum of the weighted inputs, which by definition of $A \odot K$ is just $(A \odot K)_{22}$.
- We can obtain another entry $(A \odot K)_{ij}$ by replacing the input values with another submatrix of A.



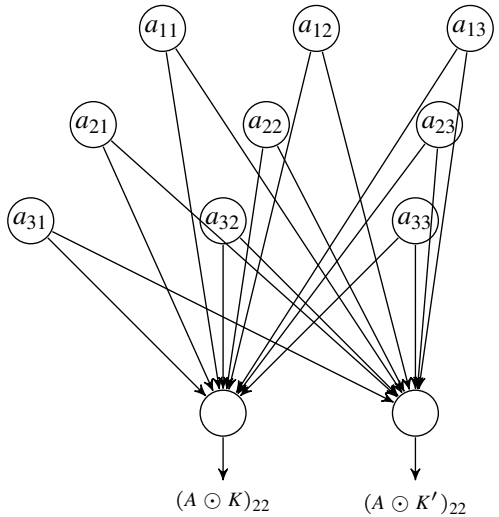
(i) (ii) (iii)

- Neural network layers whose units are arranged in a two-dimensional grid are often visualized as "sheets" as in (ii) and (iii).
- The network (i) collects information from a small portion of the input layer, as visualized in (ii).
- We can use a similar network (with different input values but identical weights) to similarly compute $(A \odot K)_{23}$, $(A \odot K)_{24}$, etc as in (iii).
- In that manner, we can compute every entry of $(A \odot K)$ and arrange these entries on another grid of units as the next layer.
- In other words: We attach a network of the form (i) to every 3×3 patch of input values. *All these networks use the same weights, given by the matrix K.* The two-layer network so obtained computes $(A \odot K)$. If we changed the weights to some other matrix K', it would compute $(A \odot K')$.



- Here, the input layer representing A and the consecutive layer representing $A \odot K$ are visualized as sheets.
- The layer that computes $A \odot K$ is often called a **convolutional layer**, although *cross-correlation layer* would be more accurate. (There is another operation called a convolution that is similar to cross-correlation, but not identical.)
- Neural networks that contain convolutional layers are called **convolutional neural networks**, even if not every layer is a convolution. Typically, the first hidden layer performs a convolution.
- Almost all networks used for image processing and computer vision problems are convolutional neural networks.

Computing Several Cross-Correlations in Parralel

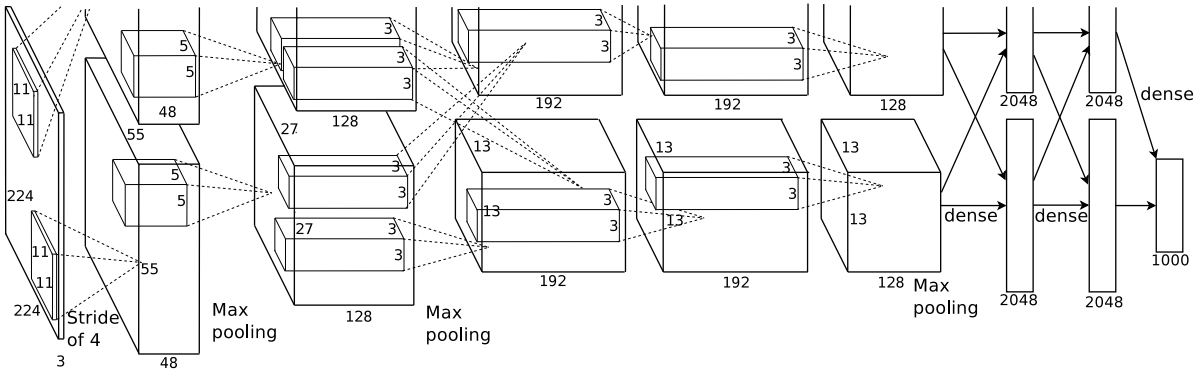


- We start with the same network as before that computes $(A \odot K)_{22}$.
- For each input vertex, we add a second connection and collect all of these in a second (linear) unit. That is, the second layer now has two units.
- The connections to the first node on the second layer still use the weights given by K. (The weights are omitted above since the figure would get too crowded.)
- Now specify a second 3×3 matrix K'. Use its entries as weights for the additional connections, collected by the second linear unit.
- The network now computes $(A \odot K)_{22}$ (as output of one unit in the second layer) and $(A \odot K')_{22}$ (as output of the other one).

Object Recognition Tasks

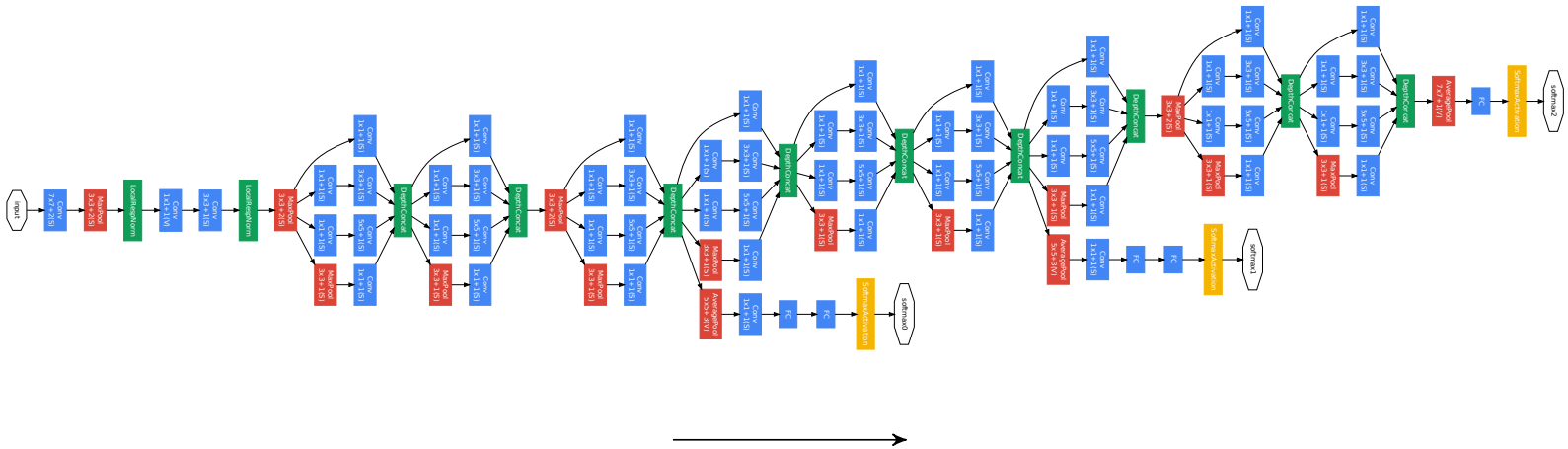
- An important benchmark problem is object recognition.
- The task is, roughly: An image is fed into a multiclass classifier, and the classifier should output the label of a/the "dominant" object in the image.
- For a picture of a car with background, the label would be "car", possibly plus a specific type or model.
- The current state of the art for this problem are (convolutional) neural networks whose input is the entire image (i.e. there is no prior feature extraction step).
- The next two slides illustrate models that performed best in comparisons organized as a contest in 2012 and 2014.

STATE-OF-THE-ART IN 2012



- This is an illustration (taken from the research article) of the convolutional network that first demonstrated enourmous improvements in computer vision benchmark tasks.
- "Stride of 4" refers to a convolutional layer that applies 96 kernels in parallel.
- Each of the big blocks in the figure represents a convolutional layer.
- In between the convolutional layers, additional operations are performed ("pooling" and a form of normalization).
- "Pooling" refers to operations that collect outputs from a rectangular patch adjacent units and summarize them in a single unit. That reduces layer size.
- "Dense" refers to a layer that is fully connected (all possible edges from one layer to the next are present). These are located at towards the output end of the network, where layer size has already been reduced.

STATE-OF-THE-ART IN 2014



- Layers: Convolution (blue), pooling (red), various others.
- This network was designed by Google (the one of the previous page in academia).

How do we evaluate which methods work?

- The basic evaluation of data mining/machine learning methods is conducted by individual research groups and reported in scientific articles.
- These results use different data sets, different cross-validation setups, etc. That makes them hard to compare.
- It is easy to cheat, too. That is not in anyone's long-term interest as a researcher, but it happens.
- It is easy to make mistakes, e.g. by getting your cross-validation wrong.

Benchmark data sets

- Benchmark data sets are sets of labelled data used by many researchers to make results more comparable.
- Early examples in computer vision are the *Berkeley Segmentation Dataset and Benchmark* (2001, for image segmentation) and the *Caltech 101* dataset (2004, for object categorization).

Challenges

- To make evaluation (not just data) comparable, some research groups organize competitions (often called "challenges" in computer vision and machine learning).
- The organizers specify a task (e.g. a classification problem) and a performance goal (e.g. "achieve minimal classification error on the test data").
- Research groups can sign up to participate.
- A set of labelled data is made available to participants, for use as training data.
- The organizers hold out a test data set (which is kept secret). At the end of the competition, all participating groups submit their final trained model, the organizers run it on the test data, and report the results.

ILSVRC

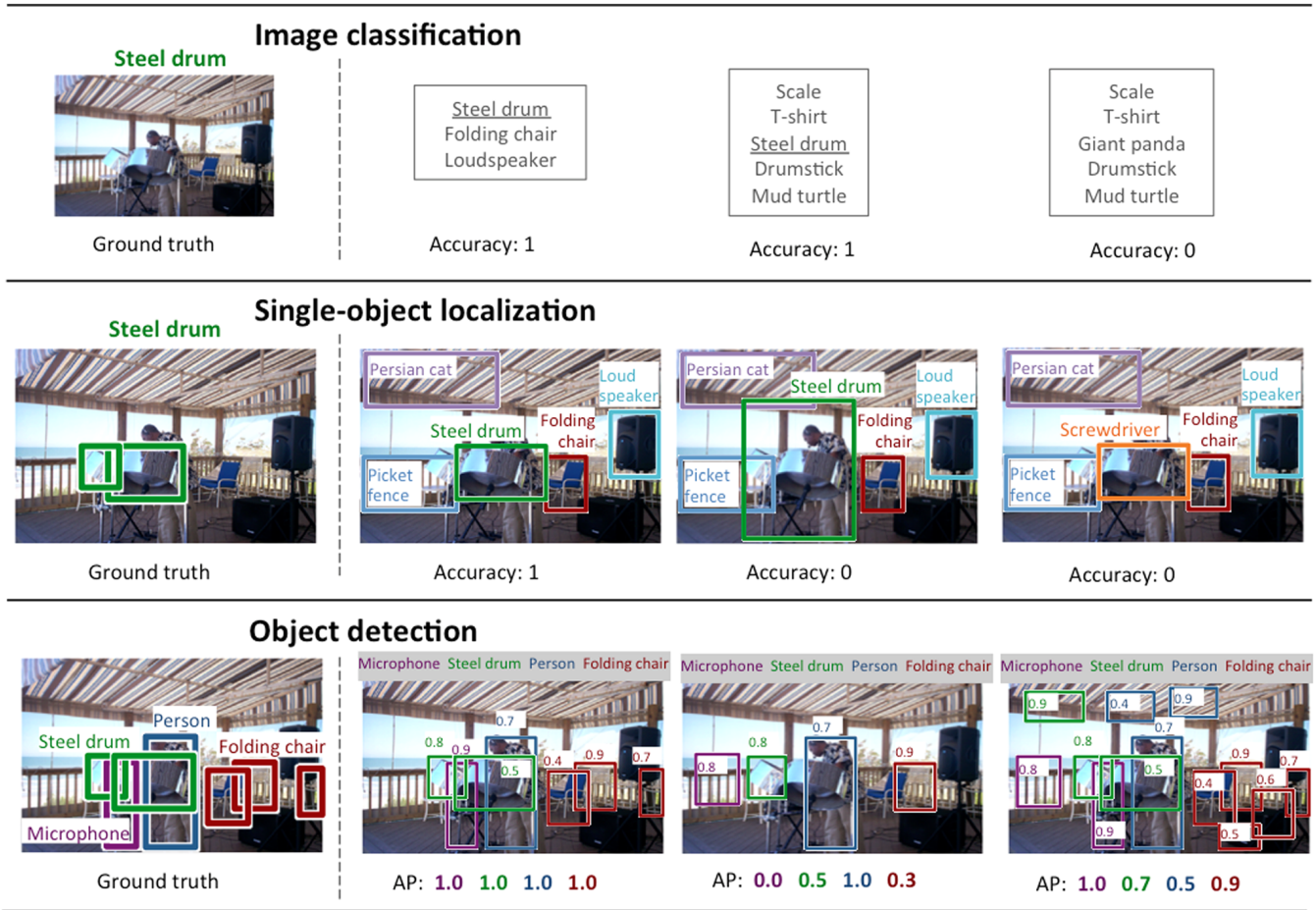
- The best-known example is the *ImageNet Large-Scale Visual Recognition Challenge* (or ILSVRC). which evaluates how well an algorithm can perform certain vision tasks, like classifiying and locating objects in images.
- In 2012, a "deep" neural network drastically improved on previous ILSVRC results. That was one of the triggers for the current interest of the tech industry in machine learning. The network is the one picture on slide 323.

ILSVRC Tasks

(1) *Image classification* (2010–2014): Algorithms produce a list of object categories present in the image.
(2) *Single-object localization* (2011–2014): Algorithms produce a list of object categories present in the image, along with an axis-aligned bounding box indicating the position and scale of *one* instance of each object category.
(3) *Object detection* (2013–2014): Algorithms produce a list of object categories present in the image along with an axis-aligned bounding box indicating the position and scale of *every* instance of each object category.

from Russakovsky et al: "ImageNet Large Scale Visual Recognition Challenge"

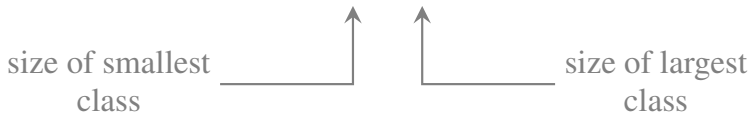
ILSVRC Tasks



from Russakovsky et al: "ImageNet Large Scale Visual Recognition Challenge"

Cross Validation Setup

Year	Train images (per class)	Val images (per class)	Test images (per class)
Image classification annotations (1000 object classes)			
ILSVRC2010	1,261,406 (668–3047)	50,000 (50)	150,000 (150)
ILSVRC2011	1,229,413 (384–1300)	50,000 (50)	100,000 (100)
ILSVRC2012-14	1,281,167 (732–1300)	50,000 (50)	100,000 (100)



- The data is split into a large training set, plus a validation and a test set.
- Research groups download the training set.
- The validation data sits on a server on which research groups can upload their trained models. The server runs the model on the validation data and reports the accuracy estimate to the researchers, who can use this feedback to improve their model.
- The test data is withheld. After a submission deadline, all submitted models are run on the test data to produce an "official" result.

from Russakovsky et al: "ImageNet Large Scale Visual Recognition Challenge"

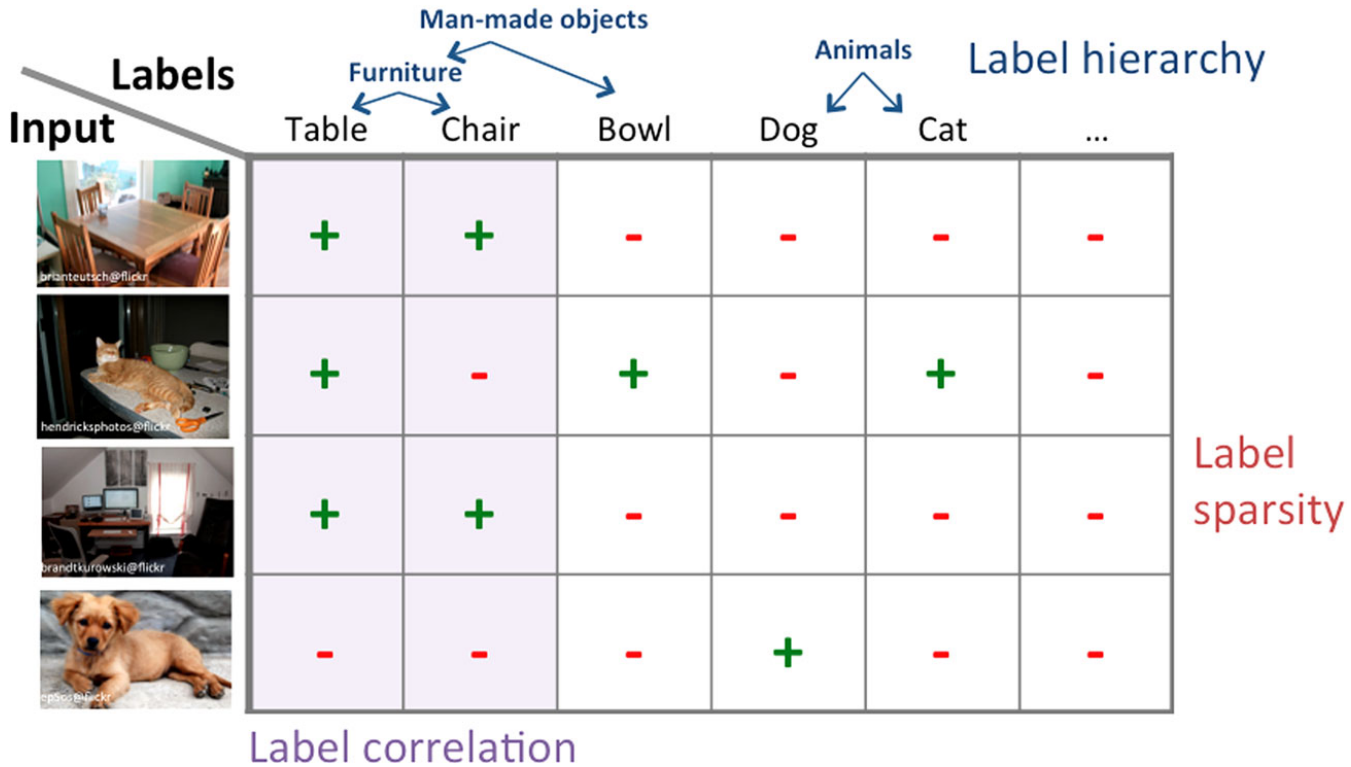
Example Rules

- Each research group is limited to two validation steps per week. (One team famously cheated its way around this rule in 2015.)
- There are separate contests that do or do not permit additional training data to be used.

Crowdsourcing

- The data is collected from image search engines.
- It does not come with reliable labels for training, validation and testing.
- The class labels are added by crowdsourcing.
- The labels are structured hierarchically, i.e. there is a meta-category “cars” which contains specific types of cars as subcategories. The challenge task is to predict the most specific labels (the leaves in the hierarchy tree).
- The next two slides illustrate how the categories are structured.

ILSVRC Data

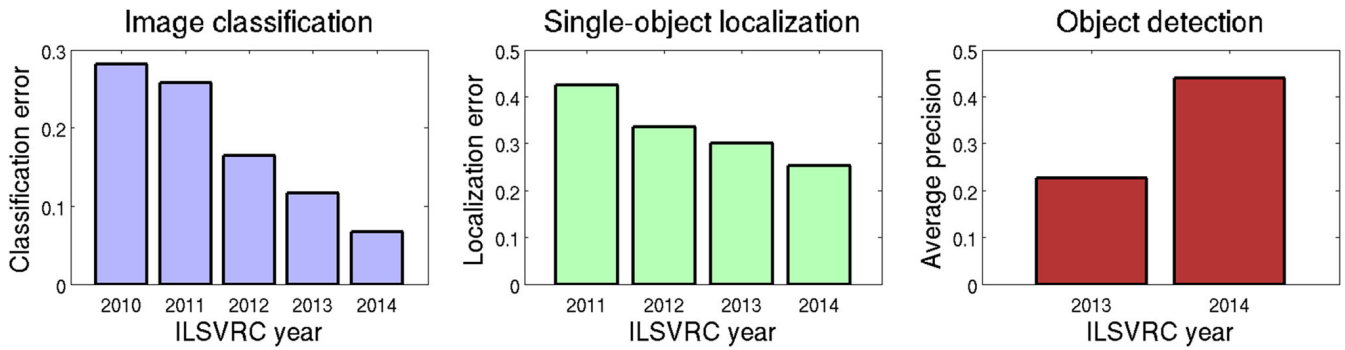


from Russakovsky et al: "ImageNet Large Scale Visual Recognition Challenge"

- wind instrument: a musical instrument in which the sound is produced by an enclosed column of air that is moved by the breath (such as trumpet, french horn, harmonica, flute, etc)
  - (17) trumpet: a brass musical instrument with a narrow tube and a flared bell, which is played by means of valves. often has 3 keys on top
  - (18) french horn: a brass musical instrument consisting of a conical tube that is coiled into a spiral, with a flared bell at the end
  - (19) trombone: a brass instrument consisting of a long tube whose length can be varied by a u-shaped slide
  - (20) harmonica
  - (21) flute: a high-pitched musical instrument that looks like a straight tube and is usually played sideways (please do not confuse with oboes, which have a distinctive straw-like mouth piece and a slightly flared end)
  - (22) oboe: a slender musical instrument roughly 65cm long with metal keys, a distinctive straw-like mouthpiece and often a slightly flared end (please do not confuse with flutes)
  - (23) saxophone: a musical instrument consisting of a brass conical tube, often with a u-bend at the end
- food: something you can eat or drink (includes growing fruit, vegetables and mushrooms, but does not include living animals)
  - food with bread or crust: pretzel, bagel, pizza, hotdog, hamburgers, etc
    - (24) pretzel
    - (25) bagel, beigel
    - (26) pizza, pizza pie
    - (27) hotdog, hot dog, red hot
    - (28) hamburger, beefburger, burger
  - (29) guacamole
  - (30) burrito
  - (31) popsicle (ice cream or water ice on a small wooden stick)
  - fruit
    - (32) fig
    - (33) pineapple, ananas
    - (34) banana
    - (35) pomegranate
    - (36) apple
    - (37) strawberry
    - (38) orange
    - (39) lemon
  - vegetables
    - (40) cucumber, cuke
    - (41) artichoke, globe artichoke
    - (42) bell pepper
    - (43) head cabbage
  - (44) mushroom
- items that run on electricity (plugged in or using batteries); including clocks, microphones, traffic lights, computers, etc
  - (45) remote control, remote
  - electronics that blow air
    - (46) hair dryer, blow dryer
    - (47) electric fan: a device for creating a current of air by movement of a surface or surfaces (please do not consider hair dryers)
  - electronics that can play music or amplify sound
    - (48) tape player
    - (49) iPod
  - (50) microphone, mike
  - computer and computer peripherals: mouse, laptop, printer, keyboard, etc
    - (51) computer mouse
    - (52) laptop, laptop computer
    - (53) printer (please do not consider typewriters to be printers)
    - (54) computer keyboard
  - (55) lamp
  - electric cooking appliance (an appliance which generates heat to cook food or boil water)
    - (56) microwave, microwave oven
    - (57) toaster
    - (58) waffle iron
    - (59) coffee maker: a kitchen appliance used for brewing coffee automatically
  - (60) vacuum, vacuum cleaner
  - (61) dishwasher, dish washer, dishwashing machine
  - (62) washer, washing machine: an electric appliance for washing clothes
  - (63) traffic light, traffic signal, stoplight
  - (64) tv or monitor: an electronic device that represents information in visual form
  - (65) digital clock: a clock that displays the time of day digitally
- kitchen items: tools,utensils and appliances usually found in the kitchen
  - electric cooking appliance (an appliance which generates heat to cook food or boil water)
    - (56) microwave, microwave oven
    - (57) toaster
    - (58) waffle iron
    - (59) coffee maker: a kitchen appliance used for brewing coffee automatically
  - (61) dishwasher, dish washer, dishwashing machine
  - (66) stove
  - things used to open cans/bottles: can opener or corkscrew
    - (67) can opener (tin opener)
    - (68) corkscrew
  - (69) cocktail shaker
  - non-electric item commonly found in the kitchen: pot, pan, utensil, bowl, etc
    - (90) tie: a long piece of cloth worn for decorative purposes around the neck or shoulders, resting under the shirt collar and knotted at the throat (NOT a bow tie)
  - headdress, headgear: clothing for the head (hats, helmets, bathing caps, etc)
    - (87) bathing cap, swimming cap: a cap worn to keep hair dry while swimming or showering
    - (91) hat with a wide brim
    - (92) helmet: protective headgear made of hard material to resist blows
  - (93) miniskirt, mini: a very short skirt
  - (94) brassiere, bra: an undergarment worn by women to support their breasts
  - (95) sunglasses

- living organism (other than people): dogs, snakes, fish, insects, sea urchins, starfish, etc.
  - living organism which can fly
    - (96) bee
    - (97) dragonfly
    - (98) ladybug
    - (99) butterfly
    - (100) bird
  - living organism which cannot fly (please don't include humans)
    - living organism with 2 or 4 legs (please don't include humans):
      - mammals (but please do not include humans)
        - feline (cat-like) animal: cat, tiger or lion
          - (101) domestic cat
          - (102) tiger
          - (103) lion
        - canine (dog-like animal): dog, hyena, fox or wolf
          - (104) dog, domestic dog, canis familiaris
          - (105) fox: wild carnivorous mammal with pointed muzzle and ears and a bushy tail (please do not confuse with dogs)
        - animals with hooves: camels, elephants, hippos, pigs, sheep, etc
          - (106) elephant
          - (107) hippopotamus, hippo
          - (108) camel
          - (109) swine: pig or boar
          - (110) sheep: woolly animal, males have large spiraling horns (please do not confuse with antelope which have long legs)
          - (111) cattle: cows or oxen (domestic bovine animals)
          - (112) zebra
          - (113) horse
          - (114) antelope: a graceful animal with long legs and horns directed upward and backward
        - (115) squirrel
        - (116) hamster: short-tailed burrowing rodent with large cheek pouches
        - (117) otter
        - (118) monkey
        - (119) koala bear
        - (120) bear (other than pandas)
        - (121) skunk (mammal known for its ability fo spray a liquid with a strong odor; they may have a single thick stripe across back and tail, two thinner stripes, or a series of white spots and broken stripes
        - (122) rabbit
        - (123) giant panda: an animal characterized by its distinct black and white markings
        - (124) red panda: Reddish-brown Old World raccoon-like carnivore
      - (125) frog, toad
      - (126) lizard: please do not confuse with snake (lizards have legs)
      - (127) turtle
      - (128) armadillo
      - (129) porcupine, hedgehog
    - living organism with 6 or more legs: lobster, scorpion, insects, etc.
      - (130) lobster: large marine crustaceans with long bodies and muscular tails; three of their five pairs of legs have claws
      - (131) scorpion
      - (132) centipede: an arthropod having a flattened body of 15 to 173 segments each with a pair of legs, the foremost pair being modified as prehensors
      - (133) tick (a small creature with 4 pairs of legs which lives on the blood of mammals and birds)
      - (134) isopod: a small crustacean with seven pairs of legs adapted for crawling
      - (135) ant
    - living organism without legs: fish, snake, seal, etc. (please don't include plants)
      - living organism that lives in water: seal, whale, fish, sea cucumber, etc.
        - (136) jellyfish
        - (137) starfish, sea star
        - (138) seal
        - (139) whale
        - (140) ray: a marine animal with a horizontally flattened body and enlarged winglike pectoral fins with gills on the underside
        - (141) goldfish: small golden or orange-red fishes
      - living organism that slides on land: worm, snail, snake
        - (142) snail
        - (143) snake: please do not confuse with lizard (snakes do not have legs)
- vehicle: any object used to move people or objects from place to place
  - a vehicle with wheels
    - (144) golfcart, golf cart
    - (145) snowplow: a vehicle used to push snow from roads

ILSVRC Results



from Russakovsky et al: "ImageNet Large Scale Visual Recognition Challenge"

Discuss overfitting