

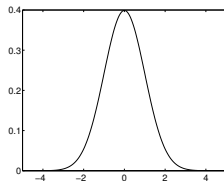
**TOOLS: MAXIMUM LIKELIHOOD**

# GAUSSIAN DISTRIBUTION

## Gaussian density in one dimension

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

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



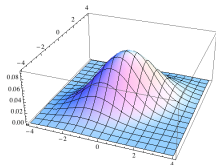
## Gaussian density in $d$ dimensions

The quadratic function

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

is replaced by a quadratic form:

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



## 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  $\theta$ ) 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  $P_\theta$  be a density function  $p(x|\theta)$ .

# MAXIMUM LIKELIHOOD ESTIMATION

## Setting

- ▶ Given: Data  $x_1, \dots, x_n$ , 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

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

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 distribution  $P$ .

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

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

## 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, and for numerical stability.

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:

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

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

## Likelihood and logarithm trick

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

## 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!

# 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  $\Sigma$ ,

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

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  $\mu$ .

## EXAMPLE: GAUSSIAN MEAN MLE

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

Multiplication by  $\Sigma$  gives

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

### Conclusion

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

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

# EXAMPLE: GAUSSIAN WITH UNKNOWN COVARIANCE

## Model: Multivariate Gaussians

The model  $\mathcal{P}$  is now

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

where  $\Delta_d$  is the set of positive definite  $d \times d$ -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  $\mu$  does not depend on  $\Sigma$ , we can compute  $\hat{\mu}_{\text{ML}}$  first. We then estimate  $\Sigma$  using the criterion

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

## Solution

The ML estimator of  $\Sigma$  is

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

# CLASSIFICATION

# ASSUMPTIONS AND TERMINOLOGY

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

We assume:

1. All measurements can be represented as elements of  $\mathbb{R}^d$  ( $d$  dimensional Euclidean space).
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,

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

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

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

## Supervised vs. unsupervised

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

## Types of supervised learning methods

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

# A VERY SIMPLE CLASSIFIER

## Algorithm

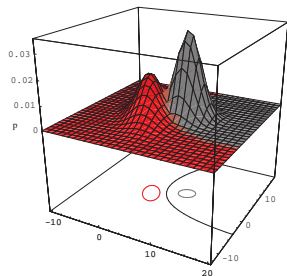
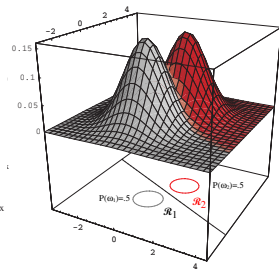
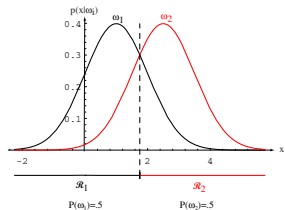
1. On training data, fit a Gaussian into each class (by MLE).  
Result: Densities  $g(\mathbf{x}|\mu_{\oplus}, \Sigma_{\oplus})$  and  $g(\mathbf{x}|\mu_{\ominus}, \Sigma_{\ominus})$
2. Classify test point according to which density assigns larger value:

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

## Resulting classifier

- ▶ Hyperplane if  $\Sigma_{\oplus} = \Sigma_{\ominus} = \text{constant} \cdot \text{diag}(1, \dots, 1)$  (=isotropic Gaussians)
- ▶ Quadratic hypersurface 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

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

## Our program in the following

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

# MEASURING PERFORMANCE: LOSS FUNCTIONS

## Definition

A **loss function** is a function

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

which we read as

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

## Example: The two most common loss functions

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

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

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  $p$ , that is,

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

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

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

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

# NAIVE BAYES CLASSIFIERS



# BAYES EQUATION

## Simplest form

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

Then

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

and we obtain

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

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  $\mathbf{Y}$ ,

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

## Classification

We define a classifier as

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

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

With the Bayes equation, we obtain

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

If the class-conditional distribution is continuous, we use

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

# BAYES-OPTIMAL CLASSIFIER

## Optimal classifier

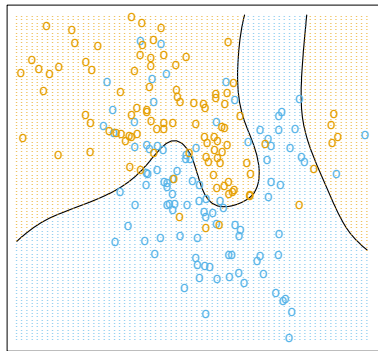
- ▶ In the risk framework, the best possible classifier is the one which minimizes the risk.
- ▶ Which classifier is optimal depends on the chosen cost function.

## Zero-one loss

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

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

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



# EXAMPLE: SPAM FILTERING

## Representing emails

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

For example, if axis  $j$  represents the term "the",  $x_j = 3.00$  means that "the" occurs three times out of every hundred in an email  $\mathbf{x}$ . This representation is called a **vector space model of text**.

## Example dimensions

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

## With Bayes equation

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

## Simplifying assumption

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

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

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

## In spam example

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

## Class prior

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

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

## Class-conditional distributions

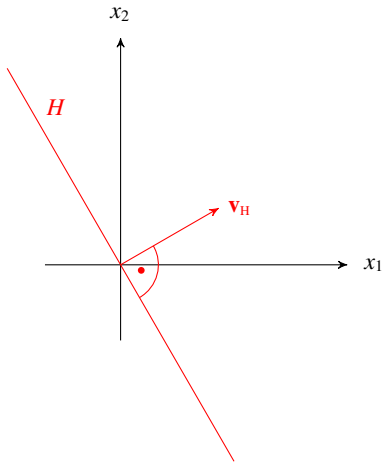
The class conditionals  $p(x|y)$  usually require a modeling assumption. Under a given model:

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

# LINEAR CLASSIFICATION





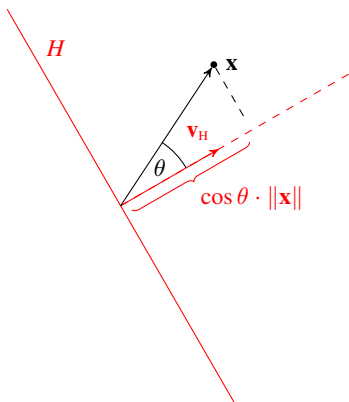


## Hyperplanes

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

- ▶ A  $\mathbb{R}^2$ -hyperplane is a line, a  $\mathbb{R}^3$ -hyperplane is a plane.

# WHICH SIDE OF THE PLANE ARE WE ON?



## Distance from the plane

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

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

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

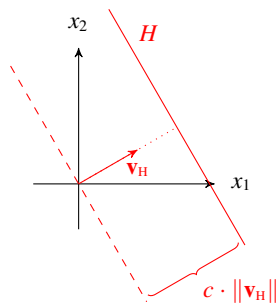
$$d(\mathbf{x}, H) = \frac{\langle \mathbf{x}, \mathbf{v}_H \rangle}{\|\mathbf{v}_H\|} = \cos \theta \cdot \|\mathbf{x}\| .$$

## Which side of the plane?

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

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

# AFFINE HYPERPLANES



## Affine Hyperplanes

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

## Which side of the plane?

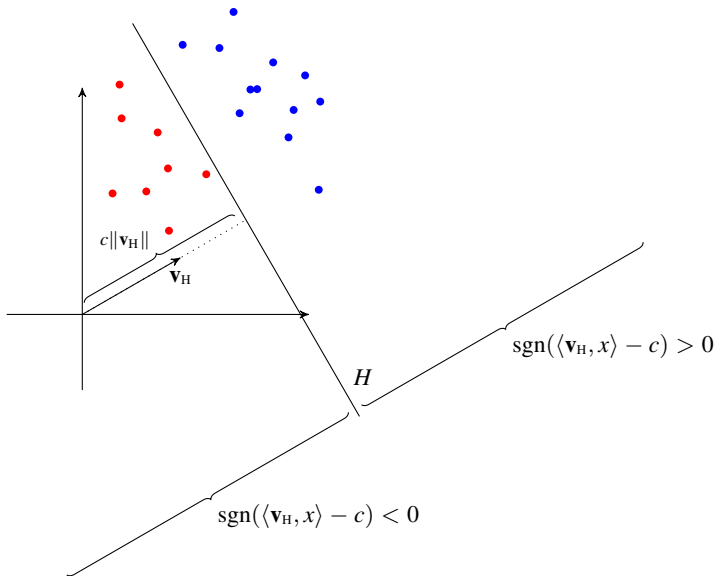
- ▶ Which side of  $H_{\mathbf{w}}$  a point  $\mathbf{x}$  is on is determined by

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

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

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

# CLASSIFICATION WITH AFFINE HYPERPLANES



# LINEAR CLASSIFIERS

## Definition

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

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

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

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

## Definition

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

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

# THE PERCEPTRON ALGORITHM

# RISK MINIMIZATION

## Definition

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

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

## Selecting a classifier

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

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

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

Note: We write “ $f \in \dots$ ”, rather than “ $f = \dots$ ”, since there may be more than one minimizer.

## Approximation with data: Empirical risk minimization

We approximate the risk criterion by the empirical risk

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

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

# HOMOGENEOUS COORDINATES

## Parameterizing the hypothesis space

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

We now have

$$\langle \mathbf{x}, \mathbf{v}_H \rangle - c = \left\langle \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix}, \mathbf{z} \right\rangle \quad \text{and} \quad f(\mathbf{x}) = \text{sgn} \left\langle \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix}, \mathbf{z} \right\rangle$$

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



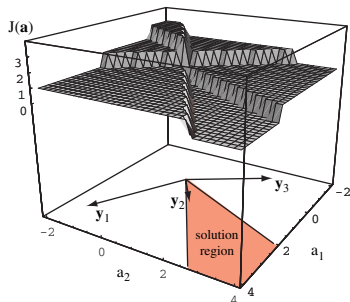
# FITTING A LINEAR CLASSIFIER

## Numerical minimization of the empirical risk

Naive strategy:

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

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



## Solution region

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

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

# THE PERCEPTRON CRITERION

## Perceptron cost function

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

The approximation

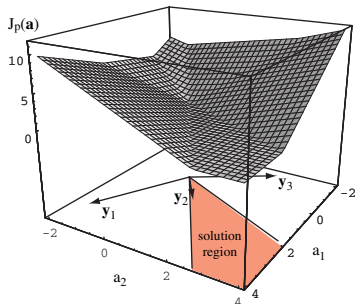
$$C_P(f) := \sum_{i=1}^n \mathbb{I}\{f(\tilde{\mathbf{x}}_i) \neq \tilde{y}_i\} \left\langle \mathbf{z}, \begin{pmatrix} 1 \\ \tilde{\mathbf{x}}_i \end{pmatrix} \right\rangle$$

is called the **Perceptron cost function**.

## Cost functions

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

training method = cost function + minimization algorithm



## The Perceptron

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

## Algorithm

Repeat until  $C_P(\mathbf{z}^k) = 0$ :

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

where  $k$  enumerates iterations.

## Step size

The step size parameter  $\alpha$  is called the **learning rate**. Common choices are

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

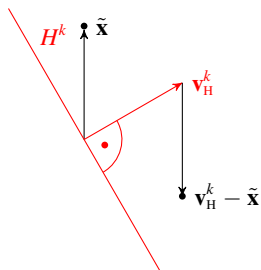
# THE GRADIENT ALGORITHM

## Gradient of the cost function

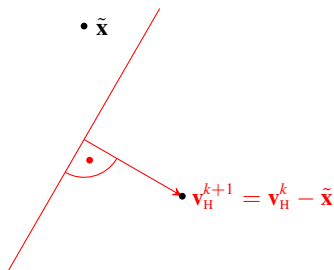
$$\begin{aligned}\nabla_{\mathbf{z}} C_P(\mathbf{z}) &= \sum_{i=1}^n \mathbb{I}\{f_H(\tilde{\mathbf{x}}_i) \neq \tilde{y}_i\} \nabla_{\mathbf{z}} \left| \langle \mathbf{z}, \begin{pmatrix} 1 \\ \tilde{\mathbf{x}}_i \end{pmatrix} \rangle \right| = \sum_{i=1}^n \mathbb{I}\{f_H(\tilde{\mathbf{x}}_i) \neq \tilde{y}_i\} \cdot \text{sgn}(\langle \mathbf{z}, \begin{pmatrix} 1 \\ \tilde{\mathbf{x}}_i \end{pmatrix} \rangle) \cdot \begin{pmatrix} 1 \\ \tilde{\mathbf{x}}_i \end{pmatrix} \\ &= \sum_{i=1}^n \mathbb{I}\{f_H(\tilde{\mathbf{x}}_i) \neq \tilde{y}_i\} \cdot (-\tilde{y}_i) \begin{pmatrix} 1 \\ \tilde{\mathbf{x}}_i \end{pmatrix} .\end{aligned}$$

## Effect for a single training point

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



Step  $k + 1$



Simplifying assumption:  $H$  contains origin

# DOES THE PERCEPTRON WORK?

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

## Alternative Algorithm

Repeat until  $C_P(\mathbf{z}) = 0$ :

1. For all  $i = 1, \dots, n$ :  $\mathbf{z}^k := \mathbf{z}^k + \mathbb{I}\{f_H(\tilde{\mathbf{x}}_i) \neq \tilde{y}_i\}(\tilde{y}_i) \begin{pmatrix} 1 \\ \tilde{\mathbf{x}}_i \end{pmatrix}$
2.  $k := k + 1$

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

## Theorem: Perceptron convergence

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

# MAXIMUM MARGIN CLASSIFIERS

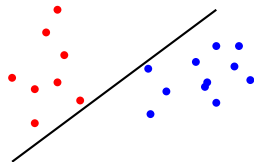
# MAXIMUM MARGIN IDEA

## Setting

Linear classification, two linearly separable classes.

## Recall Perceptron

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



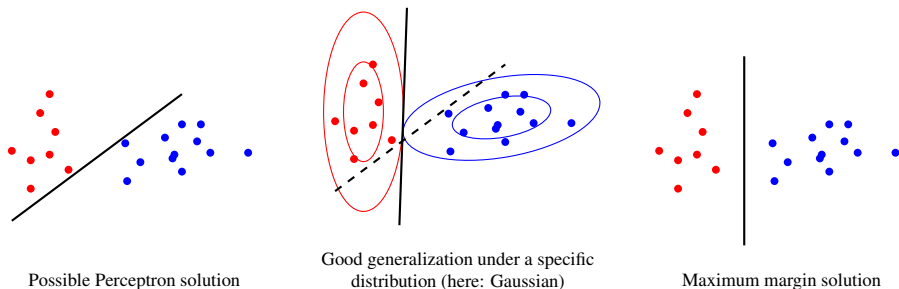
## Maximum margin idea

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



## Example: Gaussian data

- ▶ The ellipses represent lines of constant standard deviation (1 and 2 STD respectively).
- ▶ The 1 STD ellipse contains  $\sim 65\%$  of the probability mass ( $\sim 95\%$  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

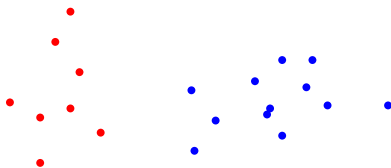
- ▶ Philosophy: Without distribution assumptions, best guess is symmetric.
- ▶ In the Gaussian example, the max-margin solution would *not* be optimal.



# SUBSTITUTING CONVEX SETS

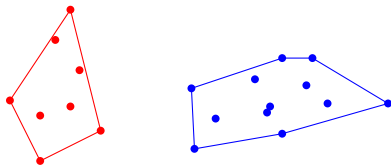
## Observation

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



## In geometric terms

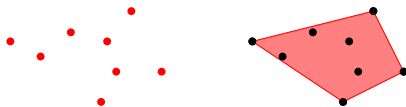
Substitute each class by the smallest convex set which contains all point in the class:



# SUBSTITUTING CONVEX SETS

## Definition

If  $C$  is a set of points, the smallest convex set containing all points in  $C$  is called the **convex hull** of  $C$ , denoted  $\text{conv}(C)$ .



Corner points of the convex set are called **extreme points**.

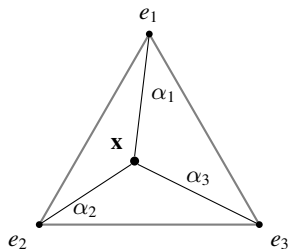
## Barycentric coordinates

Every point  $x$  in a convex set can be represented as a convex combination of the extreme points  $\{e_1, \dots, e_m\}$ .

There are weights  $\alpha_1, \dots, \alpha_m \in \mathbb{R}_+$  such that

$$\mathbf{x} = \sum_{i=1}^m \alpha_i e_i \quad \text{and} \quad \sum_{i=1}^m \alpha_i = 1.$$

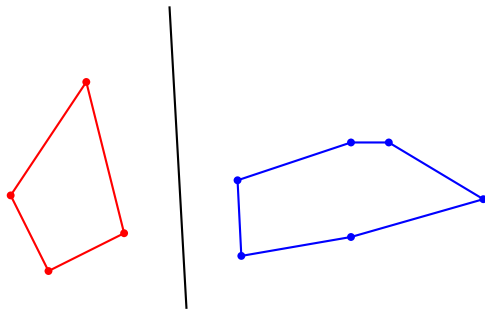
The coefficients  $\alpha_i$  are called **barycentric coordinates** of  $x$ .



# CONVEX HULLS AND CLASSIFICATION

## Key idea

A hyperplane separates two classes if and only if it separates their convex hull.



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

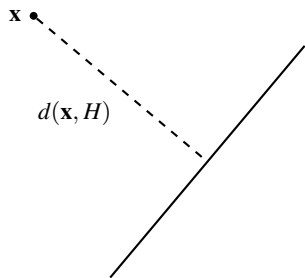
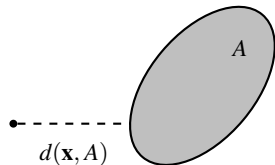
# DISTANCES TO SETS

## 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

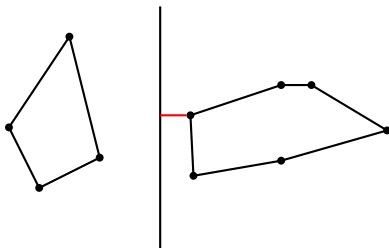
## Definition

The **margin** of a classifier hyperplane  $H$  given two training classes  $\mathcal{X}_\ominus, \mathcal{X}_\oplus$  is the shortest distance between the plane and any point in either set:

$$\text{margin} = \min_{x \in \mathcal{X}_\ominus \cup \mathcal{X}_\oplus} d(x, H)$$

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

$$\text{margin} = \min\{d(H, \text{conv}(\mathcal{X}_\ominus)), d(H, \text{conv}(\mathcal{X}_\oplus))\}$$



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

# LINEAR CLASSIFIER WITH MARGIN

## Recall: Specifying affine plane

Normal vector  $\mathbf{v}_H$ .

$$\langle \mathbf{v}_H, \mathbf{x} \rangle - c \begin{cases} > 0 & \mathbf{x} \text{ on positive side} \\ < 0 & \mathbf{x} \text{ on negative side} \end{cases}$$

Scalar  $c \in \mathbb{R}$  specifies shift (plane through origin if  $c = 0$ ).

## Plane with margin

Demand

$$\langle \mathbf{v}_H, \mathbf{x} \rangle - c > 1 \text{ or } < -1$$

$\{-1, 1\}$  on the right works for any margin: Size of margin determined by  $\|\mathbf{v}_H\|$ .

Understanding the “ $> 1$ ” piece is critical. Notice (assuming  $c = 0$  for simplicity):

$$\begin{aligned} \langle \mathbf{v}_H, \mathbf{x} \rangle &> 1 \\ \|\mathbf{v}_H\| \|\mathbf{x}\| \cos \theta &> 1 \\ d(\mathbf{x}, H) &> \frac{1}{\|\mathbf{v}_H\|}. \end{aligned}$$

Because  $\langle \mathbf{v}_H, \mathbf{x} \rangle$  is in units of  $\|\mathbf{v}_H\|$ , the 1 is arbitrary; margin is *increased* by *reducing* the length of  $\mathbf{v}_H$ .

## Finding the hyperplane

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

$$\begin{aligned} \min_{\mathbf{v}_H, c} \quad & \|\mathbf{v}_H\| \\ \text{s.t.} \quad & \tilde{y}_i(\langle \mathbf{v}_H, \tilde{\mathbf{x}}_i \rangle - c) \geq 1 \quad \text{for } i = 1, \dots, n \end{aligned}$$

## Definition

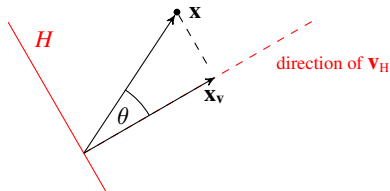
The classifier obtained by solving this optimization problem is called a **support vector machine**.

## Intuition

Just minimizing  $\|\mathbf{v}_H\|$  may seem inadequate, but consider that reducing  $\|\mathbf{v}_H\|$  increases the margin (see previous slide), and to continue increasing the margin (while still satisfying the separability constraints!), at some point the direction of  $\mathbf{v}_H$  will also have to change.

## AGAIN, WHY *minimize* $\|\mathbf{v}_H\|$ ?

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



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

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

It does not depend on the length of  $\mathbf{v}_H$ .

- ▶ The scalar product  $\langle \mathbf{x}, \mathbf{v}_H \rangle$  *does* increase if the length of  $\mathbf{v}_H$  increases.
- ▶ To compute the distance  $\|\mathbf{x}_v\|$  from  $\langle \mathbf{x}, \mathbf{v}_H \rangle$ , we have to scale out  $\|\mathbf{v}_H\|$ :

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

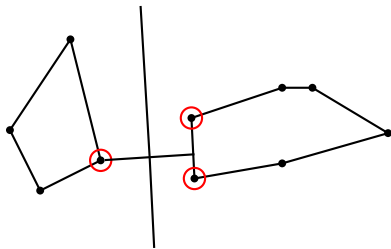


# SUPPORT VECTORS

## 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.
- ▶ Small changes in the support vectors can result in significant changes of the classifier.
- ▶ In practice, the approach is combined with "slack variables" to permit overlapping classes. As a side effect, slack variables soften the impact of changes in the support vectors.

# DUAL OPTIMIZATION PROBLEM

Solving the SVM optimization problem

$$\begin{aligned} \min_{\mathbf{v}_H, c} \quad & \|\mathbf{v}_H\| \\ \text{s.t.} \quad & \tilde{y}_i (\langle \mathbf{v}_H, \tilde{\mathbf{x}}_i \rangle - c) \geq 1 \quad \text{for } i = 1, \dots, n \end{aligned}$$

is difficult, because the constraint is a function. It is possible to transform this problem into a problem which seems more complicated, but has simpler constraints:

$$\begin{aligned} \max_{\boldsymbol{\alpha} \in \mathbb{R}^n} \quad & W(\boldsymbol{\alpha}) := \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j \tilde{y}_i \tilde{y}_j \langle \tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j \rangle \\ \text{s.t.} \quad & \sum_{i=1}^n \tilde{y}_i \alpha_i = 0 \\ & \alpha_i \geq 0 \quad \text{for } i = 1, \dots, n \end{aligned}$$

This is called the optimization problem **dual** to the minimization problem above. It is usually derived using Lagrange multipliers. We will use a more geometric argument.

# DERIVING THE DUAL PROBLEM

## Idea

We can find the maximum-margin plane as follows:

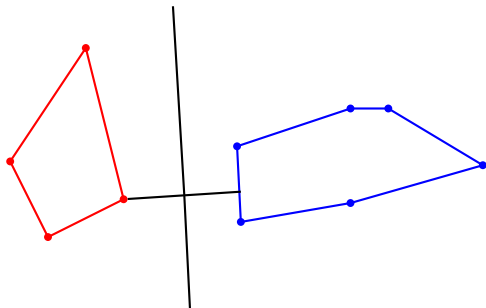
1. Find shortest line connecting the convex hulls.
2. Place classifier orthogonal to line in the middle.

Convexity of sets ensures that this classifier has correct orientation.

## As optimization problem

$$\min_{\substack{\mathbf{u} \in \text{conv}(\mathcal{X}_\ominus) \\ \mathbf{v} \in \text{conv}(\mathcal{X}_\oplus)}}$$

$$\|\mathbf{u} - \mathbf{v}\|^2$$



# BARYCENTRIC COORDINATES

## Dual optimization problem

$$\min_{\substack{\mathbf{u} \in \text{conv}(\mathcal{X}_\ominus) \\ \mathbf{v} \in \text{conv}(\mathcal{X}_\oplus)}} \|\mathbf{u} - \mathbf{v}\|^2$$

As points in the convex hulls,  $\mathbf{u}$  and  $\mathbf{v}$  can be represented by barycentric coordinates:

$$\mathbf{u} = \sum_{i=1}^{n_1} \alpha_i \tilde{\mathbf{x}}_i \quad \mathbf{v} = \sum_{i=n_1+1}^{n_1+n_2} \alpha_i \tilde{\mathbf{x}}_i \quad (\text{where } n_1 = |\mathcal{X}_\ominus|, n_2 = |\mathcal{X}_\oplus|)$$

The extreme points suffice to represent any point in the sets. If  $\tilde{\mathbf{x}}_i$  is not an extreme point, we can set  $\alpha_i = 0$ .  
Substitute into minimization problem:

$$\begin{aligned} \min_{\alpha_1, \dots, \alpha_n} \quad & \left\| \sum_{i \in \mathcal{X}_\ominus} \alpha_i \tilde{\mathbf{x}}_i - \sum_{i \in \mathcal{X}_\oplus} \alpha_i \tilde{\mathbf{x}}_i \right\|_2^2 \\ \text{s.t.} \quad & \sum_{i \in \mathcal{X}_\ominus} \alpha_i = \sum_{i \in \mathcal{X}_\oplus} \alpha_i = 1 \\ & \alpha_i \geq 0 \end{aligned}$$

# DUAL OPTIMIZATION PROBLEM

## Dual problem

$$\begin{aligned}\left\| \sum_{i \in \mathcal{X}_\ominus} \alpha_i \tilde{\mathbf{x}}_i - \sum_{i \in \mathcal{X}_\oplus} \alpha_i \tilde{\mathbf{x}}_i \right\|_2^2 &= \left\| \sum_{i \in \mathcal{X}_\ominus} \tilde{y}_i \alpha_i \tilde{\mathbf{x}}_i + \sum_{i \in \mathcal{X}_\oplus} \tilde{y}_i \alpha_i \tilde{\mathbf{x}}_i \right\|_2^2 \\ &= \left\langle \sum_{i=1}^n \tilde{y}_i \alpha_i \tilde{\mathbf{x}}_i, \sum_{j=1}^n \tilde{y}_j \alpha_j \tilde{\mathbf{x}}_j \right\rangle = \sum_{i,j} \tilde{y}_i \tilde{y}_j \alpha_i \alpha_j \langle \tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j \rangle\end{aligned}$$

Note: Minimizing this term under the constraints is equivalent to *maximizing*

$$\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \tilde{y}_i \tilde{y}_j \alpha_i \alpha_j \langle \tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j \rangle$$

under the same constraints, since  $\sum_i \alpha_i = 2$  is constant. That is just the dual problem defined four slides back. The specific form of the dual problem arises from the Lagrange multiplier derivation; this geometric argument clarifies why these terms are sensible.

## Output of dual problem

$$\mathbf{v}_H^* := \mathbf{v}^* - \mathbf{u}^* = \sum_{i=1}^n \tilde{y}_i \alpha_i^* \tilde{\mathbf{x}}_i$$

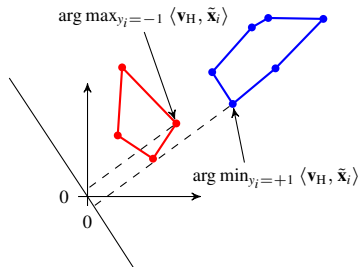
This vector describes a hyperplane through the origin. We still have to compute the offset.

## Computing the offset

$$c^* := \frac{\max_{\tilde{y}_i = -1} \langle \mathbf{v}_H^*, \tilde{\mathbf{x}}_i \rangle + \min_{\tilde{y}_i = +1} \langle \mathbf{v}_H^*, \tilde{\mathbf{x}}_i \rangle}{2}$$

## Explanation

- ▶ The max and min are computed with respect to the  $\mathbf{v}_H$  plane *containing the origin*.
- ▶ That means the max and min determine a support vector in each class.
- ▶ We then compute the shift as the mean of the two distances.



## Output of dual optimization

- ▶ Optimal values  $\alpha_i^*$  for the variables  $\alpha_i$
- ▶ If  $\tilde{\mathbf{x}}_i$  support vector:  $\alpha_i^* > 0$ , if not:  $\alpha_i^* = 0$

Note:  $\alpha_i^* = 0$  holds even if  $\tilde{\mathbf{x}}_i$  is an extreme point, but not a support vector.

## SVM Classifier

The classification function can be expressed in terms of the variables  $\alpha_i$ :

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

Intuitively: To classify a data point, it is sufficient to know which side of each support vector it is on.

# SOFT-MARGIN CLASSIFIERS

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

## Motivation 1: Nonseparable data

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

## Motivation 2: Robustness

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



# SLACK VARIABLES

## Idea

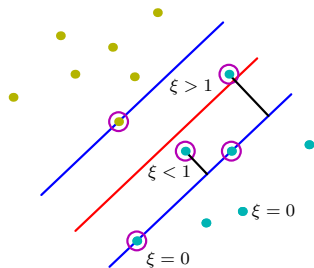
Permit training data to cross the margin, but impose cost which increases the further beyond the margin we are.

## Formalization

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

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

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



## Soft-margin optimization problem

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

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

## Role of $\gamma$

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

## Soft-margin dual problem

The slack variables vanish in the dual problem.

$$\begin{aligned} \max_{\boldsymbol{\alpha} \in \mathbb{R}^n} \quad & W(\boldsymbol{\alpha}) := \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j \tilde{y}_i \tilde{y}_j (\langle \tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j \rangle + \frac{1}{\gamma} \mathbb{I}\{i=j\}) \\ \text{s.t.} \quad & \sum_{i=1}^n \tilde{y}_i \alpha_i = 0 \\ & \alpha_i \geq 0 \quad \text{for } i = 1, \dots, n \end{aligned}$$

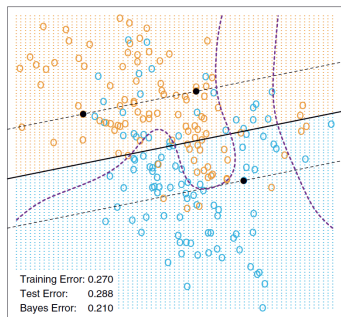
## Soft-margin classifier

The classifier looks exactly as for the original SVM:

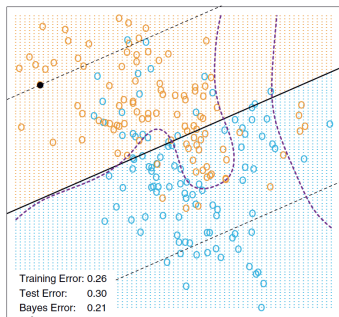
$$f(\mathbf{x}) = \text{sgn} \left( \sum_{i=1}^n \tilde{y}_i \alpha_i^* \langle \tilde{\mathbf{x}}_i, \mathbf{x} \rangle - c \right)$$

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

# INFLUENCE OF MARGIN PARAMETER



$\gamma = 100000$



$\gamma = 0.01$

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

# TOOLS: OPTIMIZATION METHODS

# OPTIMIZATION PROBLEMS

## Terminology

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

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

which we read as "find  $\mathbf{x}_0 = \arg \min_{\mathbf{x}} f(\mathbf{x})$ ".

A **constrained optimization problem** adds additional requirements on  $\mathbf{x}$ ,

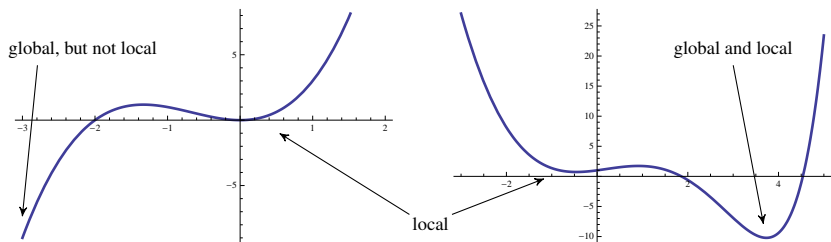
$$\begin{array}{ll} \min_{\mathbf{x}} & f(\mathbf{x}) \\ \text{subject to} & \mathbf{x} \in G, \end{array}$$

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

$$\begin{array}{ll} \min_{\mathbf{x}} & f(\mathbf{x}) \\ \text{subject to} & g(\mathbf{x}) \geq 0 \end{array}$$

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

# TYPES OF MINIMA



## Local and global minima

A minimum of  $f$  at  $x$  is called:

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

## Analytic criteria for local minima

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

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

In  $\mathbb{R}^d$ ,

$$\nabla f(\mathbf{x}) = 0 \quad \text{and} \quad H_f(\mathbf{x}) = \left( \frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{x}) \right)_{i,j=1,\dots,n} \text{ positive definite.}$$

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

## Numerical methods

All numerical minimization methods perform roughly the same steps:

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

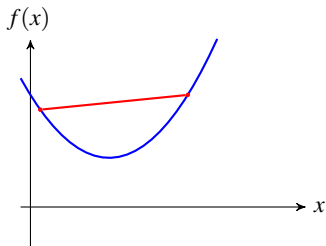
The information  $f'(x_n), f''(x_n)$  etc is always *local at  $x_n$* , and we can only decide whether a point is a local minimum, not whether it is global.



# CONVEX FUNCTIONS

## Definition

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



## Analytic criterion

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

## Implications for optimization

If  $f$  is convex, then:

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

# GRADIENT DESCENT

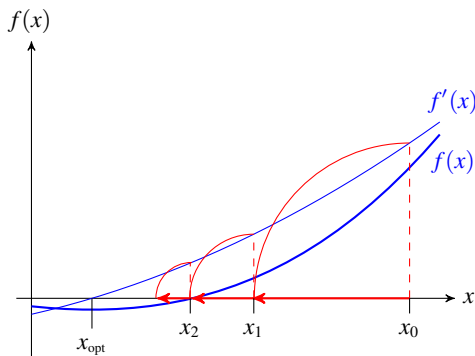
## Algorithm

Gradient descent searches for a minimum of  $f$ .

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

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

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



# NEWTON'S METHOD: ROOTS

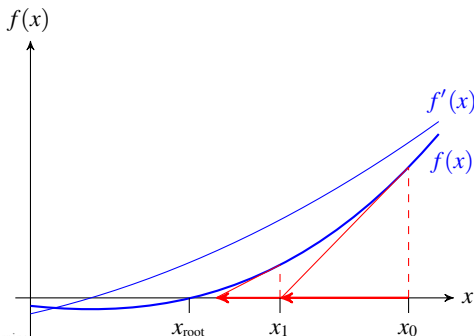
## Algorithm

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

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

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

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



## Function evaluation

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

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

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

### Example: Square root

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

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

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

# NEWTON'S METHOD: MINIMA

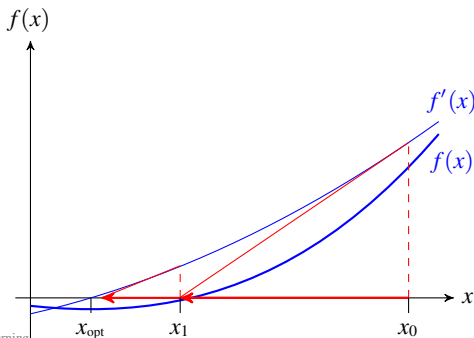
## Algorithm

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

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

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

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



# MULTIPLE DIMENSIONS

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

## Gradient descent

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

## Newton's method for minima

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

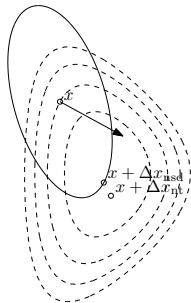
The inverse of  $H_f(\mathbf{x})$  exists only if the matrix is positive definite (not if it is only semidefinite), i.e.  $f$  has to be strictly convex.

The Hessian measures the curvature of  $f$ .

## Effect of the Hessian

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

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



## Convergence

- ▶ The algorithm always converges if  $f'' > 0$  (or  $H_f$  positive definite).
- ▶ The speed of convergence separates into two phases:
  - ▶ In a (possibly small) region around the minimum,  $f$  can always be approximated by a quadratic function.
  - ▶ Once the algorithm reaches that region, the error decreases at quadratic rate. Roughly speaking, the number of correct digits in the solution doubles in each step.
  - ▶ Before it reaches that region, the convergence rate is linear.

## High dimensions

- ▶ The required number of steps hardly depends on the dimension of  $\mathbb{R}^d$ . Even in  $\mathbb{R}^{10000}$ , you can usually expect the algorithm to reach high precision in half a dozen steps.
- ▶ Caveat: The individual steps can become very expensive, since we have to invert  $H_f$  in each step, which is of size  $d \times d$ .

# NEXT: CONSTRAINED OPTIMIZATION

## So far

- ▶ If  $f$  is differentiable, we can search for local minima using gradient descent.
- ▶ If  $f$  is sufficiently nice (convex and twice differentiable), we know how to speed up the search process using Newton's method.

## Constrained problems

- ▶ The numerical minimizers use the criterion  $\nabla f(x) = 0$  for the minimum.
- ▶ In a constrained problem, the minimum is *not* identified by this criterion.

## Next steps

We will figure out how the constrained minimum can be identified. We have to distinguish two cases:

- ▶ Problems involving only equalities as constraints (sometimes easy).
- ▶ Problems also involving inequalities (a bit more complex).



## Objective

$$\begin{aligned} & \min f(\mathbf{x}) \\ & \text{subject to } g(\mathbf{x}) = 0 \end{aligned}$$

## Idea

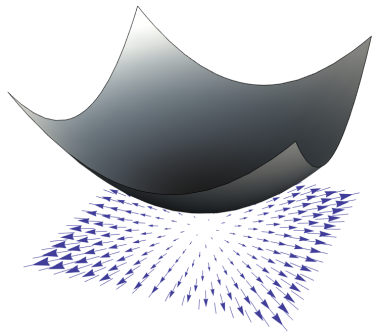
- ▶ The feasible set is the set of points  $\mathbf{x}$  which satisfy  $g(\mathbf{x}) = 0$ ,

$$G := \{\mathbf{x} \mid g(\mathbf{x}) = 0\} .$$

If  $g$  is reasonably smooth,  $G$  is a smooth surface in  $\mathbb{R}^d$ .

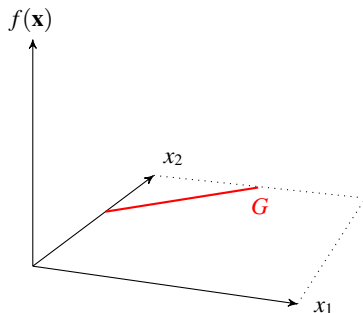
- ▶ We restrict the function  $f$  to this surface and call the restricted function  $f_g$ .
- ▶ The constrained optimization problem says that we are looking for the minimum of  $f_g$ .

# LAGRANGE OPTIMIZATION



$$f(\mathbf{x}) = x_1^2 + x_2^2$$

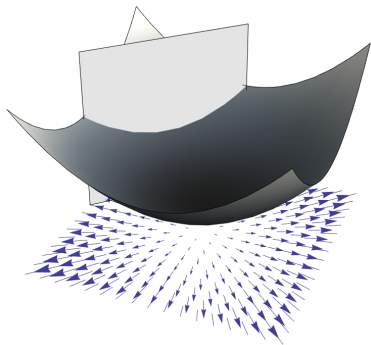
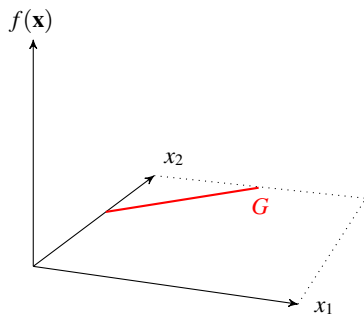
The blue arrows are the gradients  $\nabla f(\mathbf{x})$  at various values of  $\mathbf{x}$ .



Constraint  $g$ .

Here,  $g$  is linear, so the graph of  $g$  is a (sloped) affine plane. The intersection of the plane with the  $x_1$ - $x_2$ -plane is the set  $G$  of all points  $\mathbf{x}$  with  $g(\mathbf{x}) = 0$ .

# LAGRANGE OPTIMIZATION



- ▶ We can make the function  $f_g$  given by the constraint  $g(\mathbf{x}) = 0$  visible by placing a plane vertically through  $G$ . The graph of  $f_g$  is the intersection of the graph of  $f$  with the plane.
- ▶ Here,  $f_g$  has parabolic shape.
- ▶ The gradient of  $f$  at the minimum of  $f_g$  is *not* 0.

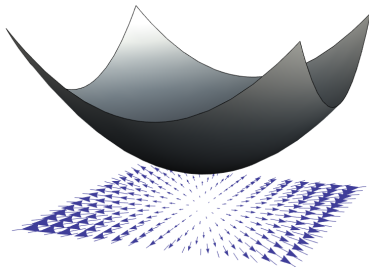
# GRADIENTS AND CONTOURS

## Fact

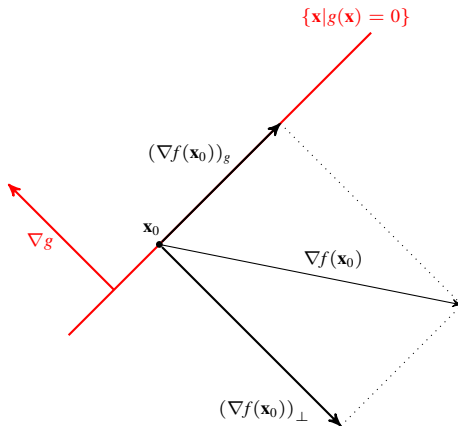
*Gradients are orthogonal to contour lines.*

## Intuition

- ▶ The gradient points in the direction in which  $f$  grows most rapidly.
- ▶ Contour lines are sets along which  $f$  does not change.



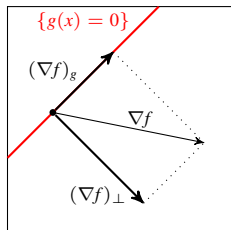
# THE CRUCIAL BIT



# AGAIN, IN DETAIL.

## Idea

- ▶ Decompose  $\nabla f$  into a component  $(\nabla f)_g$  in the set  $\{\mathbf{x} \mid g(\mathbf{x}) = 0\}$  and a remainder  $(\nabla f)_\perp$ .
- ▶ The two components are orthogonal.
- ▶ If  $f_g$  is minimal within  $\{\mathbf{x} \mid g(\mathbf{x}) = 0\}$ , the component within the set vanishes.
- ▶ The remainder need not vanish.



## Consequence

- ▶ We need a criterion for  $(\nabla f)_g = 0$ .

## Solution

- ▶ If  $(\nabla f)_g = 0$ , then  $\nabla f$  is orthogonal to the set  $g(\mathbf{x}) = 0$ .
- ▶ Since gradients are orthogonal to contours, and the set is a contour of  $g$ ,  $\nabla g$  is also orthogonal to the set.
- ▶ Hence: At a minimum of  $f_g$ , the two gradients point in the same direction:  
 $\nabla f + \lambda \nabla g = 0$  for some scalar  $\lambda \neq 0$ .

# SOLUTION: CONSTRAINED OPTIMIZATION

## Solution

The constrained optimization problem

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g(\mathbf{x}) = 0 \end{aligned}$$

is solved by solving the equation system

$$\begin{aligned} \nabla f(\mathbf{x}) + \lambda \nabla g(\mathbf{x}) &= 0 \\ g(\mathbf{x}) &= 0 \end{aligned}$$

The vectors  $\nabla f$  and  $\nabla g$  are  $D$ -dimensional, so the system contains  $D + 1$  equations for the  $D + 1$  variables  $x_1, \dots, x_D, \lambda$ .

## Objective

For a function  $f$  and a convex function  $g$ , solve

$$\begin{aligned} & \min f(\mathbf{x}) \\ & \text{subject to } g(\mathbf{x}) \leq 0 \end{aligned}$$

i.e. we replace  $g(\mathbf{x}) = 0$  as previously by  $g(\mathbf{x}) \leq 0$ . This problem is called an optimization problem with **inequality constraint**.

## Feasible set

We again write  $G$  for the set of all points which satisfy the constraint,

$$G := \{\mathbf{x} \mid g(\mathbf{x}) \leq 0\} .$$

$G$  is often called the **feasible set** (the same name is used for equality constraints).

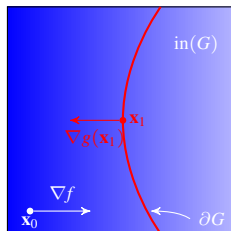


# TWO CASES

## Case distinction

1. The location  $\mathbf{x}$  of the minimum can be in the *interior* of  $G$
2.  $\mathbf{x}$  may be on the *boundary* of  $G$ .

## Decomposition of $G$



lighter shade of blue = larger value of  $f$

$$G = \text{in}(G) \cup \partial G = \text{interior} \cup \text{boundary}$$

Note: The interior is given by  $g(\mathbf{x}) < 0$ , the boundary by  $g(\mathbf{x}) = 0$ .

## Criteria for minimum

1. **In interior:**  $f_g = f$  and hence  $\nabla f_g = \nabla f$ . We have to solve a standard optimization problem with criterion  $\nabla f = 0$ .
2. **On boundary:** Here,  $\nabla f_g \neq \nabla f$ . Since  $g(\mathbf{x}) = 0$ , the geometry of the problem is the same as we have discussed for equality constraints, with criterion  $\nabla f = \lambda \nabla g$ .

**However:** In this case, the sign of  $\lambda$  matters.

# ON THE BOUNDARY

## Observation

- ▶ An extremum on the boundary is a minimum only if  $\nabla f$  points *into*  $G$ .
- ▶ Otherwise, it is a maximum instead.

## Criterion for minimum on boundary

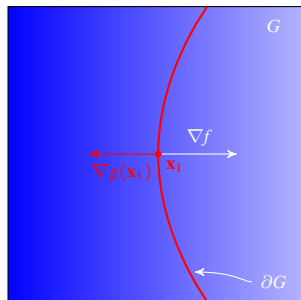
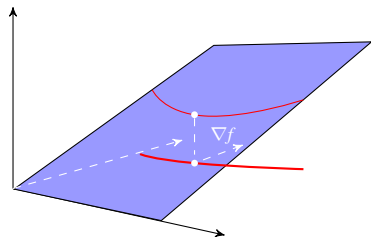
Since  $\nabla g$  points *away* from  $G$  (since  $g$  increases away from  $G$ ),  $\nabla f$  and  $\nabla g$  have to point in opposite directions:

$$\nabla f = \lambda \nabla g \quad \text{with } \lambda < 0$$

## Convention

To make the sign of  $\lambda$  explicit, we constrain  $\lambda$  to positive values and instead write:

$$\begin{aligned} \nabla f &= -\lambda \nabla g \\ \text{s.t. } \lambda &> 0 \end{aligned}$$



# COMBINING THE CASES

## Combined problem

$$\begin{aligned} & \nabla f = -\lambda \nabla g \\ \text{s.t. } & g(\mathbf{x}) \leq 0 \\ & \lambda = 0 \text{ if } \mathbf{x} \in \text{in}(G) \\ & \lambda > 0 \text{ if } \mathbf{x} \in \partial G \end{aligned}$$

## Can we get rid of the "if $\mathbf{x} \in \cdot$ " distinction?

Yes: Note that  $g(\mathbf{x}) < 0$  if  $\mathbf{x}$  in interior and  $g(\mathbf{x}) = 0$  on boundary. Hence, we always have either  $\lambda = 0$  or  $g(\mathbf{x}) = 0$  (and never both).

That means we can substitute

$$\begin{aligned} & \lambda = 0 \text{ if } \mathbf{x} \in \text{in}(G) \\ & \lambda > 0 \text{ if } \mathbf{x} \in \partial G \end{aligned}$$

by

$$\lambda \cdot g(\mathbf{x}) = 0 \quad \text{and} \quad \lambda \geq 0.$$

# SOLUTION: INEQUALITY CONSTRAINTS

## Combined solution

The optimization problem with inequality constraints

$$\begin{aligned} & \min f(\mathbf{x}) \\ & \text{subject to } g(\mathbf{x}) \leq 0 \end{aligned}$$

can be solved by solving

$$\begin{aligned} \text{s.t.} \quad & \left. \begin{aligned} \nabla f(\mathbf{x}) &= -\lambda \nabla g(\mathbf{x}) \\ \lambda g(\mathbf{x}) &= 0 \\ g(\mathbf{x}) &\leq 0 \\ \lambda &\geq 0 \end{aligned} \right\} \leftarrow \begin{aligned} & \text{system of } d + 1 \text{ equations for } d + 1 \\ & \text{variables } x_1, \dots, x_D, \lambda \end{aligned} \end{aligned}$$

These conditions are known as the **Karush-Kuhn-Tucker** (or **KKT**) conditions.

## Haven't we made the problem more difficult?

- ▶ To simplify the minimization of  $f$  for  $g(\mathbf{x}) \leq 0$ , we have made  $f$  more complicated and added a variable and two constraints.
- ▶ However: In the original problem, we *do not know how to minimize  $f$* , since the usual criterion  $\nabla f = 0$  does not work.
- ▶ By adding  $\lambda$  and additional constraints, we have reduced the problem to solving a system of equations.

## Summary: Conditions

Condition	Ensures that...	Purpose
$\nabla f(\mathbf{x}) = -\lambda \nabla g(\mathbf{x})$	If $\lambda = 0$ : $\nabla f$ is 0 If $\lambda > 0$ : $\nabla f$ is anti-parallel to $\nabla g$	Opt. criterion inside $G$ Opt. criterion on boundary
$\lambda g(\mathbf{x}) = 0$	$\lambda = 0$ in interior of $G$	Distinguish cases in( $G$ ) and $\partial G$
$\lambda \geq 0$	$\nabla f$ cannot flip to orientation of $\nabla g$	Optimum on $\partial G$ is minimum

# WHY SHOULD $g$ BE CONVEX?

## More precisely

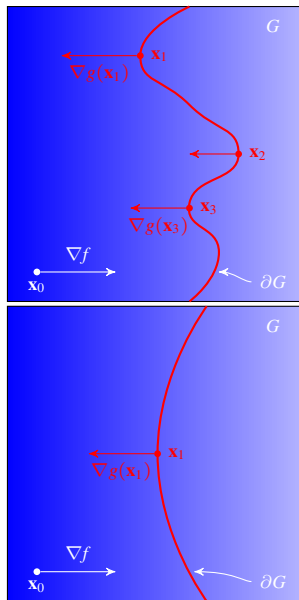
If  $g$  is a convex function, then  $G = \{\mathbf{x} \mid g(\mathbf{x}) \leq 0\}$  is a convex set. Why do we require convexity of  $G$ ?

## Problem

If  $G$  is not convex, the KKT conditions do not guarantee that  $\mathbf{x}$  is a minimum. (The conditions still hold, i.e. if  $G$  is not convex, they are necessary conditions, but not sufficient.)

## Example (Figure)

- ▶  $f$  is a linear function (lighter color = larger value)
- ▶  $\nabla f$  is identical everywhere
- ▶ If  $G$  is not convex, there can be several points (here:  $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ ) which satisfy the KKT conditions. Only  $\mathbf{x}_1$  minimizes  $f$  on  $G$ .
- ▶ If  $G$  is convex, such problems cannot occur.



## Numerical methods for constrained problems

Once we have transformed our problem using Lagrange multipliers, we still have to solve a problem of the form

$$\begin{aligned} \nabla f(\mathbf{x}) &= -\lambda \nabla g(\mathbf{x}) \\ \text{s.t.} \quad \lambda g(\mathbf{x}) &= 0 \quad \text{and} \quad g(\mathbf{x}) \leq 0 \quad \text{and} \quad \lambda \geq 0 \end{aligned}$$

numerically.

# BARRIER FUNCTIONS

## Idea

A constraint in the problem

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

can be expressed as an indicator function:

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

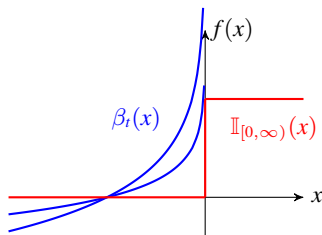
The constant must be chosen large enough to enforce the constraint.

**Problem:** The indicator function is piece-wise constant and not differentiable at 0. Newton or gradient descent are not applicable.

## Barrier function

A **barrier function** approximates  $\mathbb{I}_{[0, \infty)}$  by a smooth function, e.g.

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





## Interior point methods

We can (approximately) solve

$$\min f(x) \text{ s.t. } g_i(x) < 0 \quad \text{for } i = 1, \dots, m$$

by solving

$$\min f(x) + \sum_{i=1}^m \beta_{i,t}(x) .$$

with one barrier function  $\beta_{i,t}$  for each constraint  $g_i$ .

We do not have to adjust a multiplicative constant since  $\beta_i(x) \rightarrow \infty$  as  $x \nearrow 0$ .

## Constrained problems: General solution strategy

1. Convert constraints into solvable problem using Lagrange multipliers.
2. Convert constraints of transformed problem into barrier functions.
3. Apply numerical optimization (usually Newton's method).

## Original optimization problem

$$\min_{\mathbf{v}_H, c} \|\mathbf{v}_H\|_2 \quad \text{s.t.} \quad y_i(\langle \mathbf{v}_H, \tilde{\mathbf{x}}_i \rangle - c) \geq 1 \quad \text{for } i = 1, \dots, n$$

Problem with inequality constraints  $g_i(\mathbf{v}_H) \leq 0$  for  $g_i(\mathbf{v}_H) := 1 - y_i(\langle \mathbf{v}_H, \tilde{\mathbf{x}}_i \rangle - c)$ .

## Transformed problem

If we transform the problem using Lagrange multipliers  $\alpha_1, \dots, \alpha_n$ , we obtain:

$$\begin{aligned} \max_{\boldsymbol{\alpha} \in \mathbb{R}^n} \quad & W(\boldsymbol{\alpha}) := \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j \tilde{y}_i \tilde{y}_j \langle \tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j \rangle \\ \text{s.t.} \quad & \sum_{i=1}^n y_i \alpha_i = 0 \\ & \alpha_i \geq 0 \quad \text{for } i = 1, \dots, n \end{aligned}$$

This is precisely the "dual problem" we obtained before using geometric arguments. We can find the max-margin hyperplane using an interior point method.

## Minimization problems

Most methods that we encounter in this class can be phrased as minimization problem. For example:

Problem	Objective function
ML estimation	negative log-likelihood
Classification	empirical risk
Regression	fitting or prediction error
Unsupervised learning	suitable cost function (later)

## More generally

The lion's share of algorithms in statistics or machine learning fall into either of two classes:

1. Optimization methods.
2. Simulation methods (e.g. Markov chain Monte Carlo algorithms).

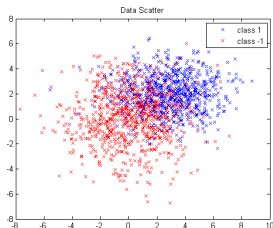
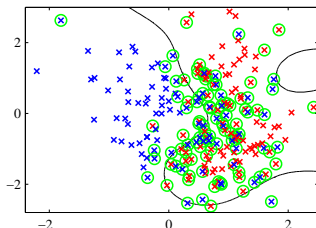
# KERNELS

# MOTIVATION

## Classifiers discussed so far

- ▶ Both assume linear decision boundary
- ▶ Perceptron: Linear separability; placement of boundary rather arbitrary

## More realistic data



# MOTIVATION: KERNELS

## Idea

- ▶ The SVM uses the scalar product  $\langle \mathbf{x}, \tilde{\mathbf{x}}_i \rangle$  as a measure of similarity between  $\mathbf{x}$  and  $\tilde{\mathbf{x}}_i$ , and of distance to the hyperplane.
- ▶ Since the scalar product is linear, the SVM is a linear method.
- ▶ By using a *nonlinear* function instead, we can make the classifier nonlinear.

## More precisely

- ▶ Scalar product can be regarded as a two-argument function

$$\langle \cdot, \cdot \rangle : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$$

- ▶ We will replace this function with a function  $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  and substitute

$$k(\mathbf{x}, \mathbf{x}') \quad \text{for every occurrence of} \quad \langle \mathbf{x}, \mathbf{x}' \rangle$$

in the SVM formulae.

- ▶ Under certain conditions on  $k$ , all optimization/classification results for the SVM still hold. Functions that satisfy these conditions are called **kernel functions**.

# THE MOST POPULAR KERNEL

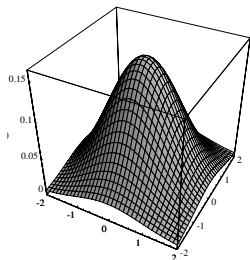
## RBF Kernel

$$k_{\text{RBF}}(\mathbf{x}, \mathbf{x}') := \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{2\sigma^2}\right) \quad \text{for some } \sigma \in \mathbb{R}_+$$

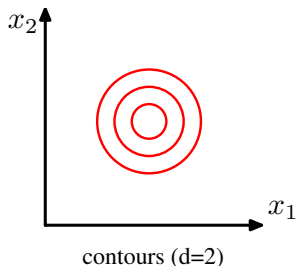
is called an **RBF kernel** (RBF = radial basis function). The parameter  $\sigma$  is called **bandwidth**.

Other names for  $k_{\text{RBF}}$ : Gaussian kernel, squared-exponential kernel.

If we fix  $\mathbf{x}'$ , the function  $k_{\text{RBF}}(\cdot, \mathbf{x}')$  is (up to scaling) a spherical Gaussian density on  $\mathbb{R}^d$ , with mean  $\mathbf{x}'$  and standard deviation  $\sigma$ .



function surface (d=2)



contours (d=2)

# CHOOSING A KERNEL

## Theory

To define a kernel:

- ▶ We have to define a function of two arguments and prove that it is a kernel.
- ▶ This is done by checking a set of necessary and sufficient conditions known as “Mercer’s theorem”.

## Practice

The data analyst does not define a kernel, but tries some well-known standard kernels until one seems to work. Most common choices:

- ▶ The RBF kernel.
- ▶ The "linear kernel"  $k_{\text{SP}}(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$ , i.e. the standard, linear SVM.

## Once kernel is chosen

- ▶ Classifier can be trained by solving the optimization problem using standard software.
- ▶ SVM software packages include implementations of most common kernels.



# WHICH FUNCTIONS WORK AS KERNELS?

## Formal definition

A function  $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  is called a **kernel** on  $\mathbb{R}^d$  if there is *some* function  $\phi : \mathbb{R}^d \rightarrow \mathcal{F}$  into *some* space  $\mathcal{F}$  with inner product  $\langle \cdot, \cdot \rangle_{\mathcal{F}}$  such that

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{F}} \quad \text{for all } \mathbf{x}, \mathbf{x}' \in \mathbb{R}^d .$$

## In other words

- ▶  $k$  is a kernel if it can be interpreted as a scalar product on some other space.
- ▶ If we substitute  $k(\mathbf{x}, \mathbf{x}')$  for  $\langle \mathbf{x}, \mathbf{x}' \rangle$  in all SVM equations, we implicitly train a *linear* SVM on the space  $\mathcal{F}$ .
- ▶ The SVM still works: It still uses scalar products, just on another space.

## The mapping $\phi$

- ▶  $\phi$  has to transform the data into data on which a linear SVM works well.
- ▶ This is usually achieved by choosing  $\mathcal{F}$  as a higher-dimensional space than  $\mathbb{R}^d$ .

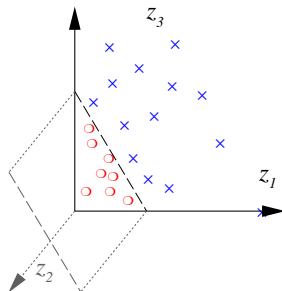
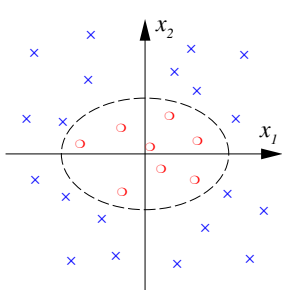
# MAPPING INTO HIGHER DIMENSIONS

## Example

How can a map into higher dimensions make class boundary (more) linear?

Consider

$$\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3 \quad \text{where} \quad \phi \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} := \begin{pmatrix} x_1^2 \\ 2x_1x_2 \\ x_2^2 \end{pmatrix}$$



# MAPPING INTO HIGHER DIMENSIONS

## Problem

In previous example: We have to know what the data looks like to choose  $\phi$ !

## Solution

- ▶ Choose high dimension  $h$  for  $\mathcal{F}$ .
- ▶ Choose components  $\phi_i$  of  $\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \dots, \phi_h(\mathbf{x}))$  as different nonlinear mappings.
- ▶ If two points differ in  $\mathbb{R}^d$ , some of the nonlinear mappings will amplify differences.

## The RBF kernel is an extreme case

- ▶ The function  $k_{\text{RBF}}$  can be shown to be a kernel, however:
- ▶  $\mathcal{F}$  is infinite-dimensional for this kernel.

# DETERMINING WHETHER $k$ IS A KERNEL

## Mercer's theorem

A mathematical result called *Mercer's theorem* states that, if the function  $k$  is positive, i.e.

$$\int_{\mathbb{R}^d \times \mathbb{R}^d} k(\mathbf{x}, \mathbf{x}') f(\mathbf{x}) f(\mathbf{x}') d\mathbf{x} d\mathbf{x}' \geq 0$$

for all functions  $f$ , then it can be written as

$$k(\mathbf{x}, \mathbf{x}') = \sum_{j=1}^{\infty} \lambda_j \phi_j(\mathbf{x}) \phi_j(\mathbf{x}') .$$

The  $\phi_j$  are functions  $\mathbb{R}^d \rightarrow \mathbb{R}$  and  $\lambda_i \geq 0$ . This means the (possibly infinite) vector  $\phi(\mathbf{x}) = (\sqrt{\lambda_1} \phi_1(\mathbf{x}), \sqrt{\lambda_2} \phi_2(\mathbf{x}), \dots)$  is a feature map.

## Kernel arithmetic

Various functions of kernels are again kernels: If  $k_1$  and  $k_2$  are kernels, then e.g.

$$k_1 + k_2$$

$$k_1 \cdot k_2$$

$$\text{const.} \cdot k_1$$

are again kernels.

## Kernels in general

- ▶ Many linear machine learning and statistics algorithms can be "kernelized".
- ▶ The only conditions are:
  1. The algorithm uses a scalar product.
  2. In all relevant equations, the data (and all other elements of  $\mathbb{R}^d$ ) appear *only inside a scalar product*.
- ▶ This approach to making algorithms non-linear is known as the "kernel trick".

## Optimization problem

$$\begin{aligned} \min_{\mathbf{v}_H, c} \quad & \|\mathbf{v}_H\|_{\mathcal{F}}^2 + \gamma \sum_{i=1}^n \xi_i^2 \\ \text{s.t.} \quad & y_i (\langle \mathbf{v}_H, \phi(\tilde{\mathbf{x}}_i) \rangle_{\mathcal{F}} - c) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0 \end{aligned}$$

Note:  $\mathbf{v}_H$  now lives in  $\mathcal{F}$ , and  $\|\cdot\|_{\mathcal{F}}$  and  $\langle \cdot, \cdot \rangle_{\mathcal{F}}$  are norm and inner product on  $\mathcal{F}$ .

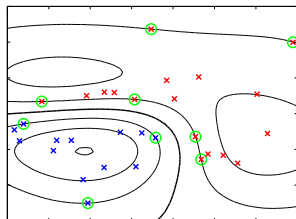
## Dual optimization problem

$$\begin{aligned} \max_{\boldsymbol{\alpha} \in \mathbb{R}^n} \quad & W(\boldsymbol{\alpha}) := \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j \tilde{y}_i \tilde{y}_j (k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j) + \frac{1}{\gamma} \mathbb{I}\{i=j\}) \\ \text{s.t.} \quad & \sum_{i=1}^n y_i \alpha_i = 0 \quad \text{and} \quad \alpha_i \geq 0 \end{aligned}$$

## Classifier

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

# SVM WITH RBF KERNEL



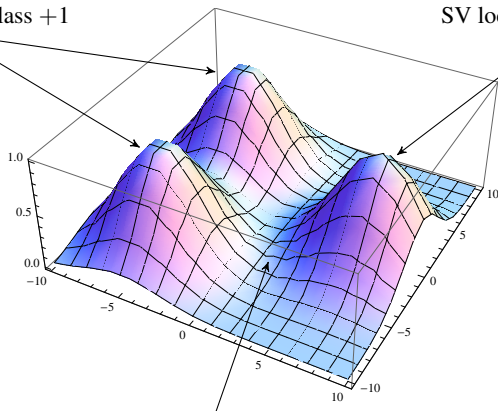
$$f(\mathbf{x}) = \text{sign} \left( \sum_{i=1}^n y_i \alpha_i^* k_{\text{RBF}}(\mathbf{x}_i, \mathbf{x}) \right)$$

- ▶ Circled points are support vectors. The two contour lines running through support vectors are the nonlinear counterparts of the convex hulls.
- ▶ The thick black line is the classifier.
- ▶ Think of a Gaussian-shaped function  $k_{\text{RBF}}(\cdot, \mathbf{x}')$  centered at each support vector  $\mathbf{x}'$ . These functions add up to a function surface over  $\mathbb{R}^2$ .
- ▶ The lines in the image are contour lines of this surface. The classifier runs along the bottom of the "valley" between the two classes.
- ▶ Smoothness of the contours is controlled by  $\sigma$

# DECISION BOUNDARY WITH RBF KERNEL

SV locations of class +1

SV location of class -1



The decision boundary runs here.

The decision boundary of the classifier coincides with the set of points where the surfaces for class +1 and class -1 have equal value.



# SUMMARY: SVMs

## Basic SVM

- ▶ Linear classifier for linearly separable data.
- ▶ Positions of affine hyperplane is determined by maximizing margin.
- ▶ Maximizing the margin is a convex optimization problem.

## Full-fledged SVM

Ingredient	Purpose
Maximum margin	Good generalization properties
Slack variables	Overlapping classes
	Robustness against outliers
Kernel	Nonlinear decision boundary

## Use in practice

- ▶ Software packages (e.g. libsvm, SVMLite)
- ▶ Choose a kernel function (e.g. RBF)
- ▶ Cross-validate margin parameter  $\gamma$  and kernel parameters (e.g. bandwidth)

# TOOLS: MODEL SELECTION AND CROSS VALIDATION

## Objective

- ▶ Cross validation is a method which tries to select the best model from a given set of models.
- ▶ Assumption: Quality measure is predictive performance.
- ▶ "Set of models" can simply mean "set of different parameter values".

## Terminology

The problem of choosing a good model is called **model selection**.

## Model selection problem for SVM

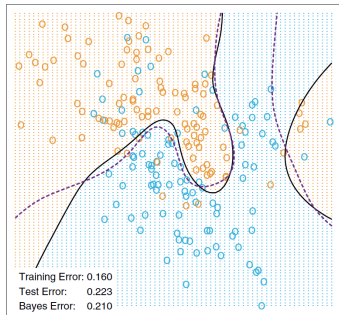
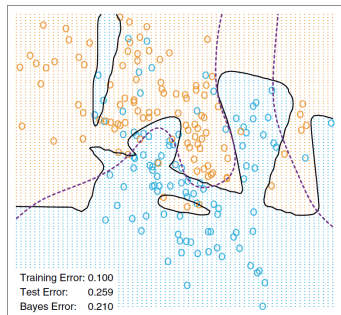
- ▶ The SVM is a *family* of models indexed by the margin parameter  $\gamma$  and the kernel parameter(s)  $\sigma$ .
- ▶ Our goal is to find a value of  $(\gamma, \sigma)$  for which we can expect small generalization error.

## Naive approach

- ▶ We could include  $(\gamma, \sigma)$  into the optimization problem, i.e. train by minimizing over  $\alpha$  and  $(\gamma, \sigma)$ .
- ▶ This leads to a phenomenon called **overfitting**: The classifier adapts too closely to specific properties of the training data, rather than the underlying distribution.

# OVERFITTING: ILLUSTRATION

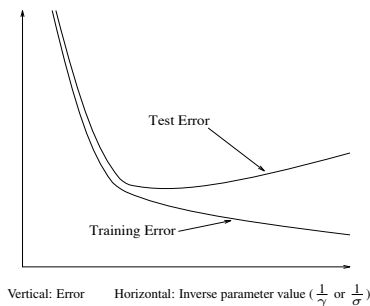
Overfitting is best illustrated with a *nonlinear* classifier.



- ▶ The classifier in this example only has a "bandwidth" parameter  $\sigma$ , similar to the parameter  $\sigma$  of the RBF kernel.
- ▶ Small  $\sigma$  permits curve with sharp bends; large  $\sigma$ : Smooth curve.

# TRAINING VS TEST ERROR

## Conceptual illustration



- ▶ 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.

# MODEL SELECTION BY CROSS VALIDATION

(From now on, we just write  $\gamma$  to denote the entire set of model parameters.)

## Cross Validation: Procedure

### Model selection:

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



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

### Model assessment:

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

## Meaning

- ▶ The quality measure by which we are comparing different classifiers  $f(\cdot; \gamma)$  (for different parameter values  $\gamma$ ) is the risk

$$R(f(\cdot; \gamma)) = \mathbb{E}[L(y, f(x; \gamma))] .$$

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

## 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 this data, we will in general underestimate it.



## Procedure in detail

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

1. For each value  $\gamma_j$ , train a classifier  $f(\cdot; \gamma_j)$  on the training set.
2. Use the validation set to estimate  $R(f(\cdot; \gamma_j))$  as the empirical risk

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

$n_v$  is the size of the validation set.

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

# K-FOLD CROSS VALIDATION

## 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  $k=3$

1	2	3	4	5
Train	Train	Validation	Train	Train

# K-FOLD CROSS VALIDATION: PROCEDURE

## Risk estimation

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

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

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

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

## Selecting a model

Choose the parameter value  $\gamma^*$  for which estimated risk is minimal.

## Model assessment

Report risk estimate for  $f(\cdot, \gamma^*)$  computed on *test* data.

# HOW TO CHOOSE $K$ ?

## Extremal cases

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

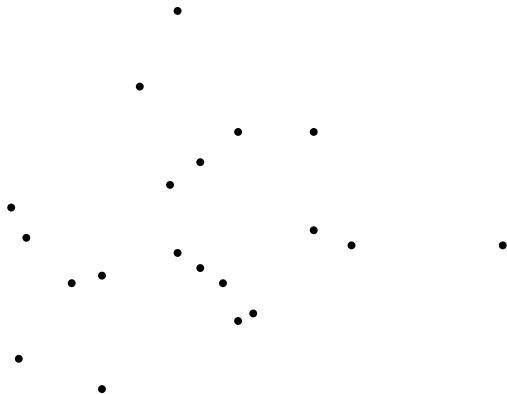
An often-cited problem with loocv is that we have to train many ( $= n$ ) classifiers, but there is also a deeper problem.

## 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.

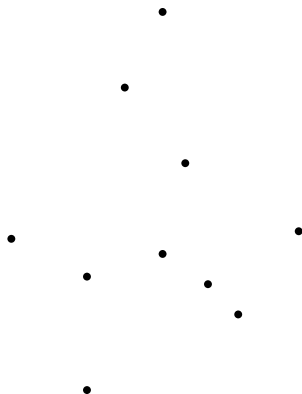
# ILLUSTRATION

$$K = 2, n = 20$$



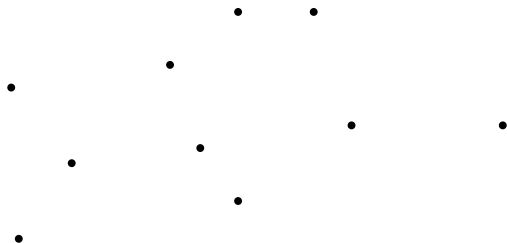
# ILLUSTRATION

$$K = 2, n = 20$$



# 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.



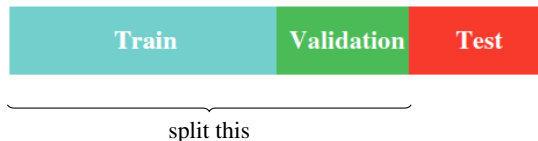
# SUMMARY: CROSS VALIDATION

## 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.
- ▶ Example above is margin parameter  $\gamma$ , but can be used for any parameter of a supervised learning algorithm.
- ▶ Note: Cross validation procedure does not involve the test data.



# MULTIPLE CLASSES

# 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

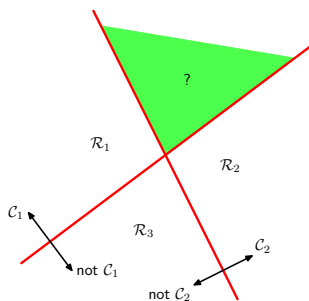
Approaches:

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

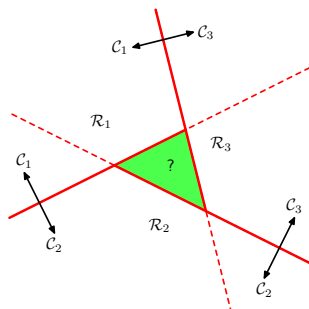
The SVM is particularly problematic.

# ONE-VERSUS-X CLASSIFICATION

## One-versus-all



## One-versus-one



- ▶ One linear classifier per class.
  - ▶ Classifies "in class  $k$ " versus "not in class  $k$ ".
  - ▶ Positive class =  $C_k$ .  
Negative class =  $\cup_{j \neq k} C_j$ .
  - ▶ Problem: Ambiguous regions (green in figure).
- ▶ 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.

# MULTICLASS DISCRIMINANTS

## Linear classifier

- ▶ Recall: Decision rule is  $f(\mathbf{x}) = \text{sgn}(\langle \mathbf{x}, \mathbf{v}_H \rangle - c)$
- ▶ 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.
- ▶ SVM: The  $K$  classifiers in the multiple discriminant approach are trained on separate problems, so the individual lengths of  $\mathbf{v}_H$  computed by max-margin algorithm are not comparable.

## Workarounds

- ▶ Often: One-against-all approaches.
- ▶ It is possible to define a single optimization problem for all classes, but training time scales quadratically in number of classes.

# 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. according to

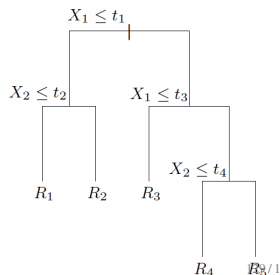
$$\mathbf{x} \in \begin{cases} \text{Class +} & \text{if } x_3 > 0.5 \\ \text{Class -} & \text{if } x_3 \leq 0.5 \end{cases}$$

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

## Tree classifier

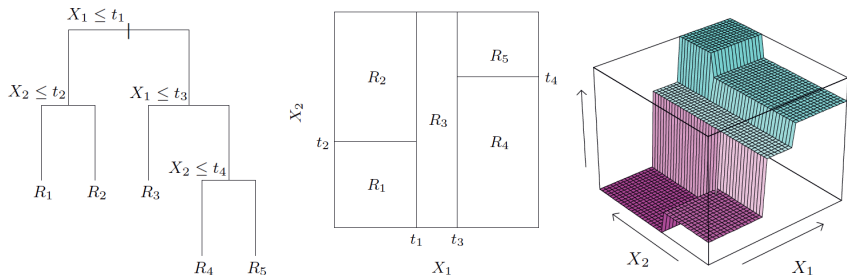
A **tree classifier** is a binary tree in which

- ▶ Each inner node is a rule of the form  $x_i > t_i$ .
- ▶ The threshold values  $t_i$  are the parameters which specify the tree.
- ▶ Each leaf is a class label.





# TREES



- ▶ Each leaf of the tree corresponds to a region  $R_m$  of  $\mathbb{R}^d$ .
- ▶ Classes  $k \in \{1, \dots, K\}$  (not restricted to two classes).
- ▶ Training: Each node is assigned class to which most points in  $R_m$  belong,

$$k(m) := \arg \max_k \#\{x_i \in R_m \text{ with } y_i = k\}$$

# FINDING A SPLIT POINT

- ▶ In training algorithm, we have to fix a region  $R_m$  and split it along an axis  $j$  at a point  $t_j$ .
- ▶ The split results in two new regions  $R_m^1$  and  $R_m^2$ .
- ▶ On each region, we obtain a new class assignment  $k^1(m)$  and  $k^2(m)$ .
- ▶ Strategy is again: Define cost of split at  $t_j$  and minimize it to find  $t_j$ .

## Cost of a split

$$Q(R_m, t_j) := \frac{\sum_{\tilde{x}_i \in R_m^1} \mathbb{I}\{\tilde{y}_i \neq k^1(m)\} + \sum_{\tilde{x}_i \in R_m^2} \mathbb{I}\{\tilde{y}_i \neq k^2(m)\}}{\#\{x_i \in R_m\}}$$

In words:

$Q$  = proportion of training points in  $R_m$  that get misclassified  
if we choose to split at  $t_j$

# TRAINING ALGORITHM

## Overall algorithm

- ▶ At each step: Current tree leaves define regions  $R_1, \dots, R_M$ .
- ▶ For each  $R_m$ , find the best split.
- ▶ Continue splitting regions until tree has depth  $D$  (input parameter).

## Step of training algorithm

At each step: Current tree leaves define regions  $R_1, \dots, R_M$ .

For each region  $R_m$ :

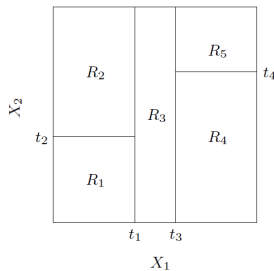
1. For each axis  $j$ : Compute best splitting point  $t_j$  as

$$t_j := \arg \min Q(R_m, t_j)$$

2. Select best splitting axis:

$$j := \arg \min_j Q(R_m, t_j)$$

3. Split  $R_m$  along axis  $j$  at  $t_j$



# EXAMPLE: SPAM FILTERING

## Data

- ▶ 4601 email messages
- ▶ Classes: email, spam

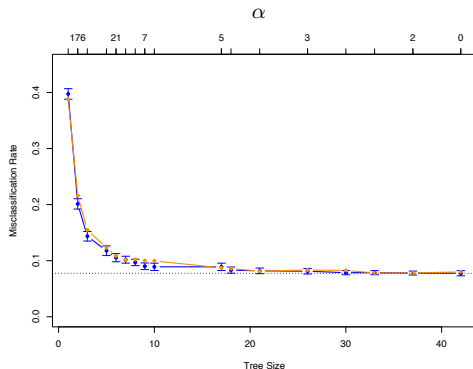
	george	you	your	hp	free	hpl	!	our	re	edu	remove
spam	0.00	2.26	1.38	0.02	0.52	0.01	0.51	0.51	0.13	0.01	0.28
email	1.27	1.27	0.44	0.90	0.07	0.43	0.11	0.18	0.42	0.29	0.01

## Tree classifier

- ▶ 17 nodes
- ▶ Performance:

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

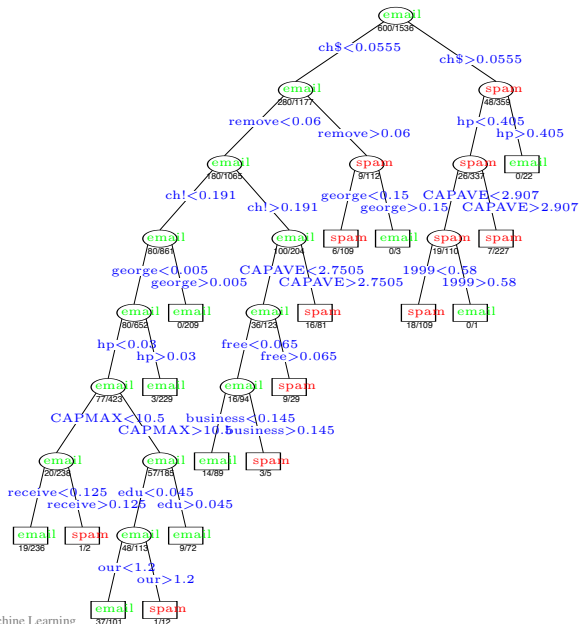
# INFLUENCE OF TREE SIZE



## Tree Size

- ▶ 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.

# SPAM FILTERING: TREE



# DECISION STUMPS

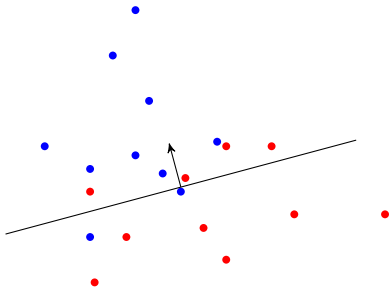
- ▶ The simplest possible tree classifier is a tree of depth 1. Such a classifier is called a **decision stump**.
- ▶ A decision stump is parameterized by a pair  $(j, t_j)$  of an axis  $j$  and a splitting point  $t_j$ .
- ▶ Splits  $\mathbb{R}^d$  into two regions.
- ▶ Decision boundary is an affine hyperplane which is perpendicular to axis  $j$  and intersects the axis at  $t_j$ .
- ▶ Often used in Boosting algorithms and other ensemble methods.

# BOOSTING



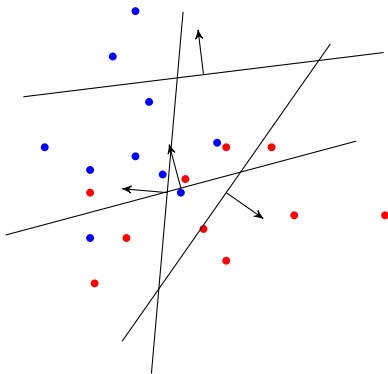
# ENSEMBLES

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



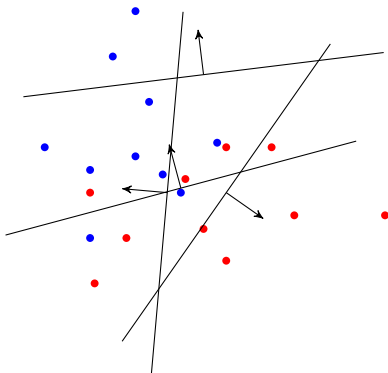
# ENSEMBLES

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



# ENSEMBLES

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:  $0.5 + \epsilon$ .
- ▶ 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  $f_1, \dots, f_m$  (with output  $\pm 1$ ):

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

## 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.

# DOES THE MAJORITY MAKE THE RIGHT CHOICE?

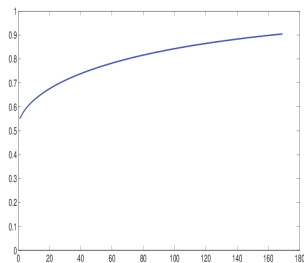
## Condorcet's rule

If the individual votes are independent, the answer is

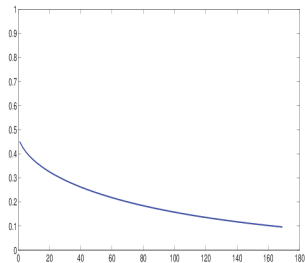
$$\Pr\{\text{majority makes correct decision}\} = \sum_{j=\frac{m+1}{2}}^m \frac{m!}{j!(m-j)!} p^j (1-p)^{m-j}$$

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

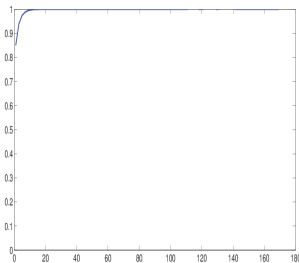
## Probability as function of the number of votes



$p = 0.55$



$p = 0.45$



$p = 0.85$

## 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.

## Bagging

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

## Boosting

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

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

## Weighted data

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

## Training on weighted data

Minimize the *weighted* misclassification 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  $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \dots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$
- ▶ Algorithm parameter: Number  $M$  of weak learners

## Training algorithm

1. Initialize the observation weights  $w_i = \frac{1}{n}$  for  $i = 1, 2, \dots, n$ .
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  $\alpha_m = \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$
  - 2.4 Set  $w_i \leftarrow w_i \cdot \exp(\alpha_m \cdot \mathbb{I}(y_i \neq g_m(x_i)))$  for  $i = 1, 2, \dots, n$ .
3. Output

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

## Weight updates

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

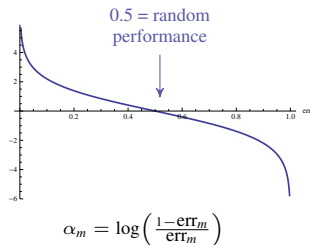
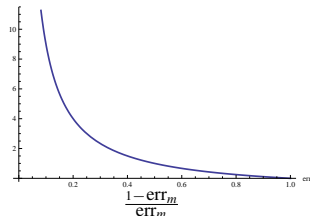
$$w_i^{(m)} = w_i^{(m-1)} \cdot \exp(\alpha_m \cdot \mathbb{I}(y_i \neq g_m(x_i)))$$

Hence:

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

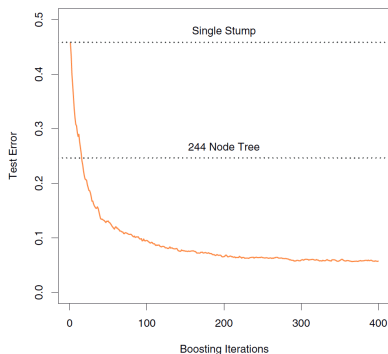
## Weighted classifier

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



# EXAMPLE

## 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.

# BOOSTING: PROPERTIES

## 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.

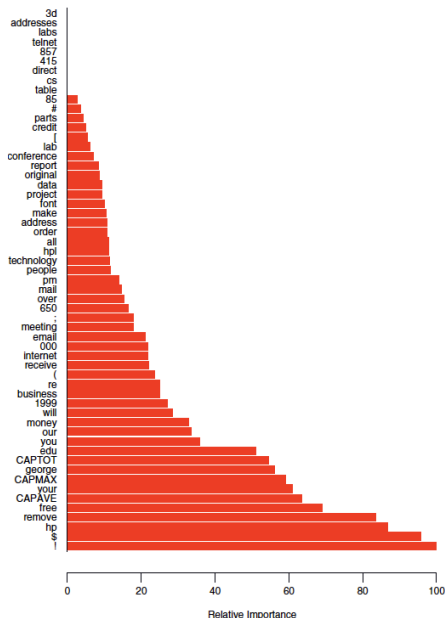
## AdaBoost with Decision Stumps

- ▶ Once AdaBoost has trained a classifier, the weights  $\alpha_m$  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  $\alpha$ , 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.

# 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"

## 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 rectangular 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 classifying *all* data as negative.
- ▶ The error rate will be precisely the proportion of points in positive class.

## Addressing class imbalance

- ▶ We have to change cost function: False negatives (= classify face as background) 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.

# CLASSIFIER CASCADES

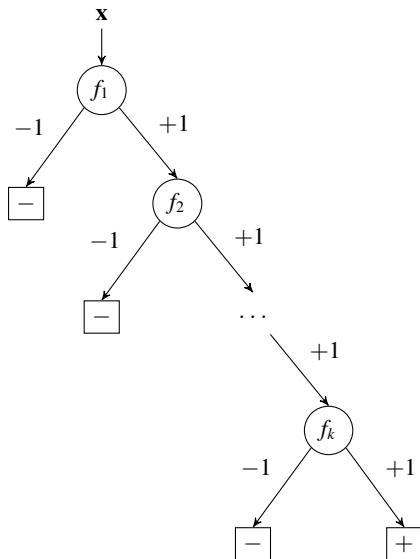
## Training a cascade

Use imbalanced loss (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,  $f(\mathbf{x}) = +1$ .



# WHY DOES A CASCADE WORK?

We have to consider two rates

$$\begin{array}{ll} \text{false positive rate} & \text{FPR}(f_j) = \frac{\#\text{negative points classified as "+1"}}{\#\text{negative training points at stage } j} \\ \text{detection rate} & \text{DR}(f_j) = \frac{\#\text{correctly classified positive points}}{\#\text{positive training points at stage } j} \end{array}$$

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

## Class imbalance

In face detection example:

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

# WHY DOES A CASCADE WORK?

## Cascade detection rate

The rates of the overall cascade classifier  $f$  are

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

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

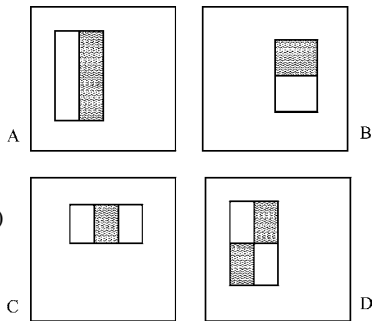
# FEATURE EXTRACTION

## Extraction method

1. Enumerate possible windows (different shapes and locations) by  $j = 1, \dots, d$ .
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  
 $\mathbf{x}_i := (x_{i1}, \dots, x_{id}) \in \mathbb{R}^d$ .



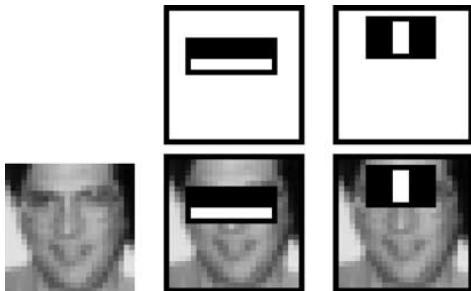
## The dimension is huge

- ▶ 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: A patch of  $24 \times 24$  pixels has  $d \approx 180000$  features.



# SELECTED FEATURES

First two selected features



200 features are selected in total.

## Training procedure

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

## Use of training data

Each training step uses:

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

# EXAMPLE RESULTS



- ▶ <http://vimeo.com/12774628>
- ▶ <https://www.youtube.com/watch?v=aTErTqOIkss>

# BAGGING AND RANDOM FORESTS

# BACKGROUND: RESAMPLING TECHNIQUES

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.

# BOOTSTRAP: BASIC ALGORITHM

## Given

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

## Bootstrap algorithm

1. Generate  $B$  **bootstrap samples**  $\mathcal{B}_1, \dots, \mathcal{B}_B$ . 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:

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

# EXAMPLE: VARIANCE ESTIMATION

## Mean and Variance

$$\mu := \int_{\mathbb{R}^d} \mathbf{x} p(\mathbf{x}) d\mathbf{x} \qquad \sigma^2 := \int_{\mathbb{R}^d} (\mathbf{x} - \mu)^2 p(\mathbf{x}) d\mathbf{x}$$

## 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

## Bootstrap algorithm

1. For  $b = 1, \dots, B$ , generate a bootstrap sample  $\mathcal{B}_b$ . In detail:  
For  $i = 1, \dots, n$ :
  - ▶ Sample an index  $j \in \{1, \dots, n\}$ .
  - ▶ 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:

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

3. Compute the bootstrap estimate:

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



# HOW OFTEN DO WE SEE EACH SAMPLE?

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

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

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

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

For large  $n$ :

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

- ▶ 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 occurrences of each  $\tilde{\mathbf{x}}_i$  is  $B$ .

Bootstrap estimate averages over reshuffled samples.

## 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:  $\hat{S}_{BS}$  typically has lower variance than  $\hat{S}$ .
- ▶ This is the property we will use for classification 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.

## 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 **a**ggregation)

# REPRESENTATION OF CLASS LABELS

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

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

# REPRESENTATION OF PREDICTED CLASS LABELS

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

$$f_b \text{ predicts } \mathbf{x}_i \text{ to be in class } k \quad \text{as} \quad f_b(\mathbf{x}_i) = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \leftarrow k\text{th entry}$$

This way, we can average together multiple predictions:

$$\frac{1}{B} (f_1(\mathbf{x}_i) + \dots + f_B(\mathbf{x}_i)) = \begin{pmatrix} p_1(\mathbf{x}_i) \\ \vdots \\ p_k(\mathbf{x}_i) \\ \vdots \\ p_K(\mathbf{x}_i) \end{pmatrix}$$

We can interpret  $p_k(\mathbf{x}_i)$  as the proportion of bootstrap classifiers  $f_1, \dots, f_B$  that classify  $\mathbf{x}_i$  to be in class  $k$ .

## Training

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

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

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

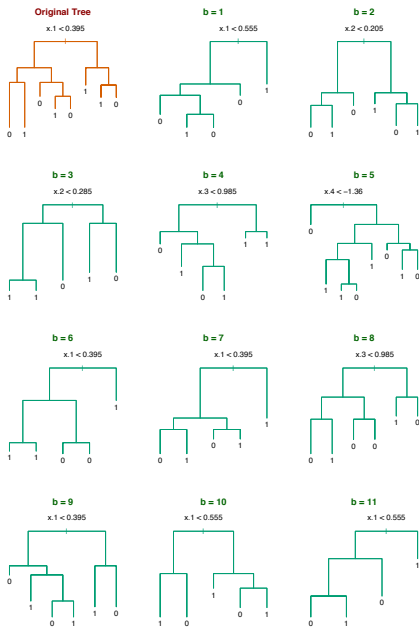
This is a vector of the form  $f_{\text{avg}}(\mathbf{x}) = (p_1(\mathbf{x}), \dots, p_k(\mathbf{x}))$ .

- ▶ The Bagging classifier is given by

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

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  $\{1, \dots, d\}$ .



# RANDOM FORESTS: ALGORITHM

## Training

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

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

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

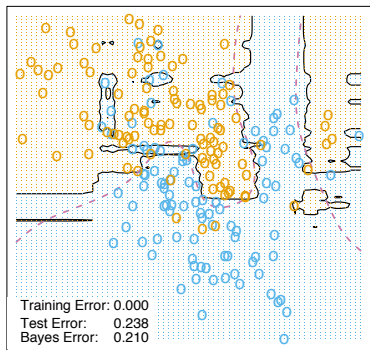
$$f_{\text{Bagging}}(\mathbf{x}) := \arg \max_k \{p_1(\mathbf{x}), \dots, p_k(\mathbf{x})\}$$

## 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.



# SUMMARY: CLASSIFICATION

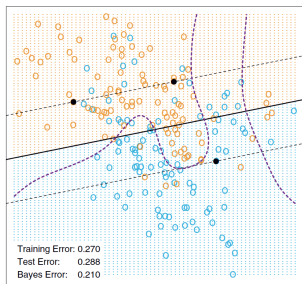
## Approaches we have discussed

- ▶ Linear classifiers
  - ▶ Perceptron, SVM
  - ▶ Nonlinear versions using kernels
- ▶ Trees (depth 1: linear and axis-parallel, depth  $\geq 2$ : non-linear)
- ▶ Ensemble methods

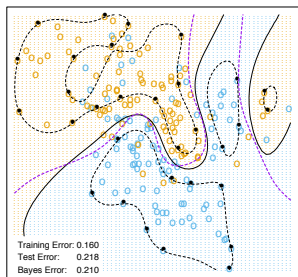
## What should we use?

- ▶ RBF SVMs, AdaBoost and Random Forests perform well on many problems.
- ▶ All have strengths and weaknesses. E.g.:
  - ▶ High dimension, limited data: SVM may have the edge.
  - ▶ Many dimensions, but we believe only a few are important: AdaBoost with stumps.
- ▶ In general: Feature extraction (what do we measure?) is crucial.
- ▶ Consider combination of different methods by voting.

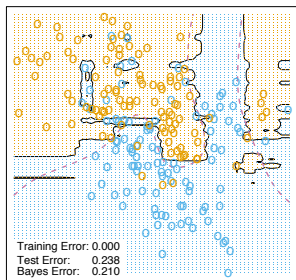
# EVERY METHOD HAS ITS IDIOSYNCRASIES



Linear SVM



RBF SVM



Random forest

# HISTORY

- ▶ Ca. 1957: Perceptron (Rosenblatt)



- ▶ 1970s: Vapnik and Chervonenkis develop learning theory
- ▶ 1986: Neural network renaissance (backpropagation algorithm by Rumelhart, Hinton, Williams)
- ▶ 1993: SVM (Boser, Guyon, Vapnik)
- ▶ 1997: Boosting (Freund and Schapire)

# REGRESSION

# REGRESSION: PROBLEM DEFINITION

## Data

- ▶ Measurements:  $\mathbf{x} \in \mathbb{R}^d$  (also: independent variable, covariate)
- ▶ Labels:  $y \in \mathbb{R}$  (also: dependent variable, response)

## Task

Find a predictor  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  such that (approximately)  $f(x) = y$  for data  $(x, y)$ . The predictor is called a **regression function**.

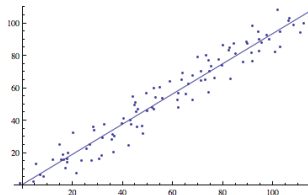
## Definition: Linear regression

A regression method is called **linear** if the predictor  $f$  is a linear function, i.e. a line if  $d = 1$  (more generally, an affine hyperplane).

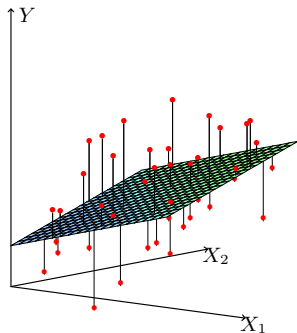


# LINEAR REGRESSION

$$\mathbf{x} \in \mathbb{R}^d \quad \text{and} \quad y \in \mathbb{R}$$



$d = 1$



$d = 2$

# LINEAR REGRESSION

## Implications of linearity

A linear function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  is always of the form

$$f(\mathbf{x}) = \beta_0 + \sum_{j=1}^d \beta_j x_j \quad \text{for } \beta_0, \beta_1, \dots, \beta_d \in \mathbb{R},$$

where  $x_j$  is the  $j$ th entry of  $\mathbf{x}$ . Recall representation of hyperplanes in classification!

## Consequence

Finding  $f$  boils down to finding  $\beta \in \mathbb{R}^{d+1}$ .

## Relation to classification

- ▶ Classification is a regression problem with  $\{1, \dots, K\}$  substituted for  $\mathbb{R}$ .
- ▶ Don't get confused—the role of the hyperplane (for, say,  $d = 2$ ) is different:
  - ▶ Regression: Graph of regression function is hyperplane in  $\mathbb{R}^{d+1}$ .
  - ▶ Classification: Regression function is piece-wise constant. The classifier hyperplane lives in  $\mathbb{R}^d$  and marks where the regression function jumps.

# LEAST-SQUARES REGRESSION

## Squared-error loss

We use the **squared-error loss function**

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

Regression methods that determine  $f$  by minimizing  $L^{\text{se}}$  are called **least-squares regression** methods.

## Least-squares linear regression

For training data  $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \dots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$ , we have to find the parameter vector  $\boldsymbol{\beta} \in \mathbb{R}^{d+1}$  which solves

$$\hat{\boldsymbol{\beta}} := \arg \min_{\boldsymbol{\beta}} \sum_{i=1}^n L^{\text{se}}(\tilde{y}_i, f(\tilde{\mathbf{x}}_i; \boldsymbol{\beta}))$$


where

$$f(\mathbf{x}; \boldsymbol{\beta}) = \beta_0 + \sum_{j=1}^d \beta_j x_j = \langle \boldsymbol{\beta}, (1, \mathbf{x}) \rangle .$$

# MATRIX FORMULATION

## Data matrix

Since  $f(\mathbf{x}; \boldsymbol{\beta}) = \langle \boldsymbol{\beta}, (1, \mathbf{x}) \rangle$ , we write the data as a matrix:

$$\tilde{\mathbf{X}} := \begin{pmatrix} 1 & (\tilde{\mathbf{x}}_1)_1 & \dots & (\tilde{\mathbf{x}}_1)_j & \dots & (\tilde{\mathbf{x}}_1)_d \\ \vdots & & & \vdots & & \vdots \\ 1 & (\tilde{\mathbf{x}}_i)_1 & \dots & (\tilde{\mathbf{x}}_i)_j & \dots & (\tilde{\mathbf{x}}_i)_d \\ \vdots & & & \vdots & & \vdots \\ 1 & (\tilde{\mathbf{x}}_n)_1 & \dots & (\tilde{\mathbf{x}}_n)_j & \dots & (\tilde{\mathbf{x}}_n)_d \end{pmatrix}$$


We write  $\tilde{\mathbf{X}}_j^{\text{col}}$  for the column vectors with  $\tilde{\mathbf{X}}_0^{\text{col}} = (1, \dots, 1)$  and  $j = 1, \dots, d$ .

$$\tilde{\mathbf{X}}\boldsymbol{\beta} = \begin{pmatrix} f(\tilde{\mathbf{x}}_1; \boldsymbol{\beta}) \\ \vdots \\ f(\tilde{\mathbf{x}}_n; \boldsymbol{\beta}) \end{pmatrix}$$

## Least-squares linear regression: Matrix form

We have to minimize

$$\sum_{i=1}^n L^{\text{sc}}(\tilde{y}_i, f(\tilde{\mathbf{x}}_i; \boldsymbol{\beta})) = \sum_{i=1}^n (\tilde{y}_i - f(\tilde{\mathbf{x}}_i; \boldsymbol{\beta}))^2 = \|\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\boldsymbol{\beta}\|_2^2$$

The solution to the linear regression problem is now  $\hat{\boldsymbol{\beta}} = \arg \min_{\boldsymbol{\beta}} \|\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\boldsymbol{\beta}\|_2^2$ .

## Solving the minimization problem

Recall:

- ▶ We have to solve for a zero derivative,  $\frac{\partial L^{\text{sc}}}{\partial \boldsymbol{\beta}}(\hat{\boldsymbol{\beta}}) = 0$ .
- ▶ That means that  $\hat{\boldsymbol{\beta}}$  is an extremum.
- ▶ To ensure that the extremum is a minimum, we have to ensure the second derivative  $\frac{\partial^2 L^{\text{sc}}}{\partial \boldsymbol{\beta}^2}(\hat{\boldsymbol{\beta}})$  is positive. For matrices: Positive definite.

# LEAST-SQUARES SOLUTION

## Solution

$$\frac{\partial L^{\text{se}}}{\partial \boldsymbol{\beta}}(\hat{\boldsymbol{\beta}}) = -2\tilde{\mathbf{X}}^\top (\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\boldsymbol{\beta})$$

Equating to zero gives the **least-squares solution**:

$$\hat{\boldsymbol{\beta}} = (\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^\top \tilde{\mathbf{y}}$$

(Recall: The transpose  $\mathbf{X}^\top$  is the matrix with  $(\mathbf{X}^\top)_{ij} := \mathbf{X}_{ji}$ .)

## Second derivative

$$\frac{\partial^2 L^{\text{se}}}{\partial \boldsymbol{\beta}^2}(\hat{\boldsymbol{\beta}}) = 2\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$$

- ▶  $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$  is always positive semi-definite. If it is also invertible, it is positive definite.
- ▶ In other words: If  $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$  is invertible (which we also need to compute  $\hat{\boldsymbol{\beta}}$ ), then  $\hat{\boldsymbol{\beta}}$  is the unique global minimum of the squared-error loss.

# TOOLS: LINEAR ALGEBRA BASICS

# IMAGES OF LINEAR MAPPINGS (1)

## Linear mapping

A matrix  $\mathbf{X} \in \mathbb{R}^{n \times m}$  defines a linear mapping  $f_{\mathbf{X}} : \mathbb{R}^m \rightarrow \mathbb{R}^n$ .

## Image

Recall: The **image** of a mapping  $f$  is the set of all possible function values, here

$$\text{image}(f_{\mathbf{X}}) := \{\mathbf{y} \in \mathbb{R}^n \mid \mathbf{X}\mathbf{z} = \mathbf{y} \text{ for some } \mathbf{z} \in \mathbb{R}^m\}$$

## Image of a linear mapping

- ▶ The image of a linear mapping  $\mathbb{R}^m \rightarrow \mathbb{R}^n$  is a linear subspace of  $\mathbb{R}^n$ .
- ▶ The columns of  $\mathbf{X}$  form a basis of the image space:

$$\text{image}(\tilde{\mathbf{X}}) = \text{span}\{\mathbf{X}_1^{\text{col}}, \dots, \mathbf{X}_m^{\text{col}}\}$$

- ▶ This is one of most useful things to remember about matrices, so, again:

*The columns span the image.*



# IMAGES OF LINEAR MAPPINGS (2)

## Dimension of the image space

Clearly: The number of linearly independent column vectors. This number is called the **column rank** of  $\mathbf{X}$ .

## Invertible mappings

Recall: A mapping  $f$  is invertible if it is one-to-one, i.e. for each function value  $\tilde{\mathbf{y}}$  there is exactly one input value with  $f(\mathbf{z}) = \tilde{\mathbf{y}}$ .

## Invertible matrices

The matrix  $\tilde{\mathbf{X}}$  is called invertible if  $f_{\tilde{\mathbf{X}}}$  is invertible.

- ▶ Only square matrices can be invertible.
- ▶ For a *linear* mapping: If  $\tilde{\mathbf{X}}$  is a square matrix  $f_{\tilde{\mathbf{X}}}$  is invertible if the image has the same dimension as the input space.
- ▶ Even if  $\tilde{\mathbf{X}} \in \mathbb{R}^{n \times m}$ , the matrix  $\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}$  is in  $\mathbb{R}^{m \times m}$  (a square matrix).
- ▶ So:  $\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}$  is invertible if  $\tilde{\mathbf{X}}$  has full column rank.

# SYMMETRIC AND ORTHOGONAL MATRICES

## Recall: Transpose

The transpose  $A^\top$  of a matrix  $A \in \mathbb{R}^m$  is the matrix with entries

$$(A^\top)_{ij} := A_{ji}$$

## Orthogonal matrices

A matrix  $O \in \mathbb{R}^{m \times m}$  is called **orthogonal**

$$O^{-1} = O^\top$$

Orthogonal matrices describe two types of operations:

1. Rotations of the coordinate system.
2. Permutations of the coordinate axes.

## Symmetric matrices

A matrix  $A \in \mathbb{R}^{m \times m}$  is called **symmetric**

$$A = A^\top$$

**Note:** Symmetric and orthogonal matrices are very different objects. Only the identity is both.

## Recall: ONB

A basis  $\{v_1, \dots, v_m\}$  of  $\mathbb{R}^m$  is called an **orthonormal basis** if

$$\langle v_i, v_j \rangle = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

In other words, the  $v_i$  are pairwise orthogonal and each of length 1.

## Orthogonal matrices

A matrix is orthogonal precisely if its rows form an ONB. Any two ONBs can be transformed into each other by an orthogonal matrix.

TOOLS:  
EIGENVALUES AND GAUSSIAN  
DISTRIBUTIONS

# EIGENVALUES

We consider a square matrix  $A \in \mathbb{R}^{m \times m}$ .

## Definition

A vector  $\xi \in \mathbb{C}^m$  is called an **eigenvector** of  $A$  if the direction of  $\xi$  does not change under application of  $A$ . In other words, if there is a scalar  $\lambda$  such that

$$A\xi = \lambda\xi .$$

$\lambda$  is called an **eigenvalue** of  $A$  for the eigenvector  $\xi$ .

## Properties in general

- ▶ In general, eigenvalues are complex numbers  $\lambda \in \mathbb{C}$ .
- ▶ The class of matrices with the nicest eigen-structure are symmetric matrices, for which all eigenvectors are mutually orthogonal and real.

# EIGENSTRUCTURE OF SYMMETRIC MATRICES

If a matrix  $A$  is symmetric:

- ▶ There are  $\text{rank}(A)$  distinct eigendirections.
- ▶ The eigenvectors are pair-wise orthogonal.
- ▶ If  $\text{rank}(A) = m$ , there is an ONB of  $\mathbb{R}^m$  consisting of eigenvectors of  $A$ .

Definiteness

type	if ...
positive definite	all eigenvalues $> 0$
positive semi-definite	all eigenvalues $\geq 0$
negative semi-definite	all eigenvalues $\leq 0$
negative definite	all eigenvalues $< 0$
indefinite	none of the above

# EIGENVECTOR ONB

## Setting

- ▶ Suppose  $A$  symmetric,  $\xi_1, \dots, \xi_m$  are eigenvectors and form an ONB.
- ▶  $\lambda_1, \dots, \lambda_m$  are the corresponding eigenvalues.

How does  $A$  act on a vector  $v \in \mathbb{R}^m$ ?

1. Represent  $v$  in basis  $\xi_1, \dots, \xi_m$ :

$$v = \sum_{j=1}^m v_j^A \xi_j \quad \text{where } v_j^A = \langle v, \xi_j \rangle \in \mathbb{R}$$

2. Multiply by  $A$ : Eigenvector definition (recall:  $A\xi_j = \lambda_j\xi_j$ ) yields

$$Av = A\left(\sum_{j=1}^m v_j^A \xi_j\right) = \sum_{j=1}^m v_j^A A\xi_j = \sum_{j=1}^m v_j^A \lambda_j \xi_j$$

## Conclusion

A symmetric matrix acts by scaling the directions  $\xi_j$ .

# ILLUSTRATION

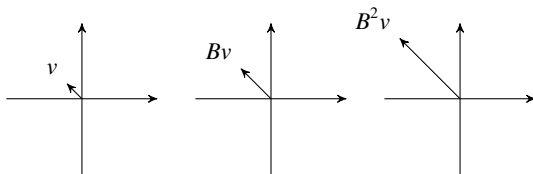
## Setting

We repeatedly apply a symmetric matrix  $B$  to some vector  $v \in \mathbb{R}^m$ , i.e. we compute

$$Bv, \quad B(Bv) = B^2v, \quad B(B(Bv)) = B^3v, \quad \dots$$

How does  $v$  change?

**Example 1:  $v$  is an eigenvector with eigenvalue 2**

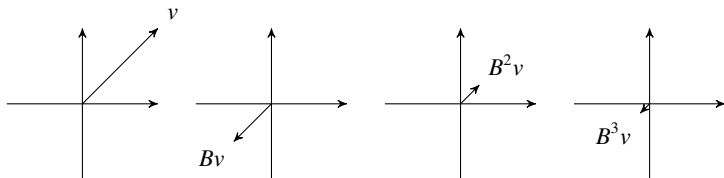


The direction of  $v$  does not change, but its length doubles with each application of  $B$ .



# ILLUSTRATION

Example 2:  $v$  is an eigenvector with eigenvalue  $-\frac{1}{2}$



For an arbitrary vector  $v$

$$B^n v = \sum_{j=1}^m v_j^B \lambda_j^n \xi_j$$

- ▶ The weight  $\lambda_j^n$  grows most rapidly for eigenvalue with largest absolute value.
- ▶ Consequence:

The direction of  $B^n v$  converges to the direction of the eigenvector with largest eigenvalue as  $n$  grows large.

# QUADRATIC FORMS

In applications, symmetric matrices often occur in quadratic forms.

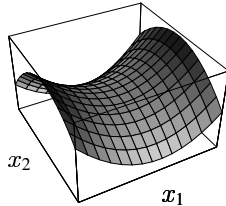
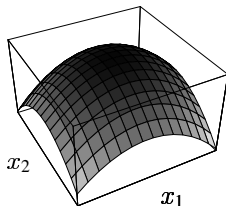
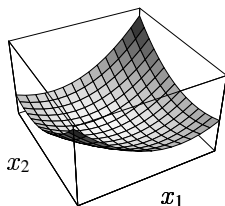
## Definition

The **quadratic form** defined by a matrix  $A$  is the function

$$q_A : \mathbb{R}^m \rightarrow \mathbb{R}$$
$$x \mapsto \langle x, Ax \rangle$$

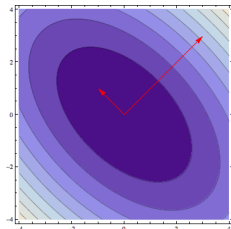
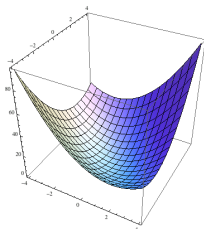
## Intuition

A quadratic form is the  $m$ -dimensional analogue of a quadratic function  $ax^2$ , with a vector substituted for the scalar  $x$  and the matrix  $A$  substituted for the scalar  $a \in \mathbb{R}$ .



# QUADRATIC FORMS

Here is the quadratic form for the matrix  $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ :

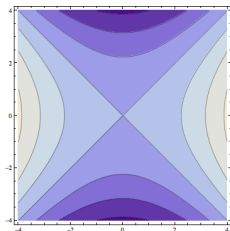
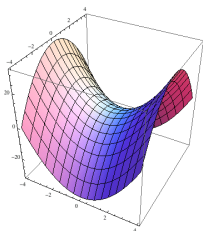


- ▶ Left: The function value  $q_A$  is graphed on the vertical axis.
- ▶ Right: Each line in  $\mathbb{R}^2$  corresponds to a constant function value of  $q_A$ .  
Dark color = small values.
- ▶ The red lines are eigenvector directions of  $A$ . Their lengths represent the (absolute) values of the eigenvalues.
- ▶ In this case, both eigenvalues are positive. If all eigenvalues are positive, the contours are ellipses. So:

positive definite matrices  $\leftrightarrow$  elliptic quadratic forms

# QUADRATIC FORMS

In this plot, the eigenvectors are axis-parallel, and one eigenvalue is negative:



The matrix here is  $A = \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}$ .

## Intuition

- ▶ If we change the sign of one of the eigenvalue, the quadratic function along the corresponding eigen-axis flips.
- ▶ There is a point which is a minimum of the function along one axis direction, and a maximum along the other. Such a point is called a *saddle point*.

# APPLICATION: COVARIANCE MATRIX

## Recall: Covariance

The covariance of two random variables  $X_1, X_2$  is

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

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

## Covariance matrix

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

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

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

## Notation

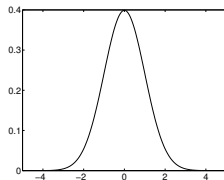
It is customary to denote the covariance matrix  $\text{Cov}[X]$  by  $\Sigma$ .

# GAUSSIAN DISTRIBUTION

## Gaussian density in one dimension

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

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



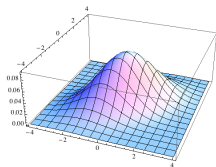
## Gaussian density in $m$ dimensions

The quadratic function

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

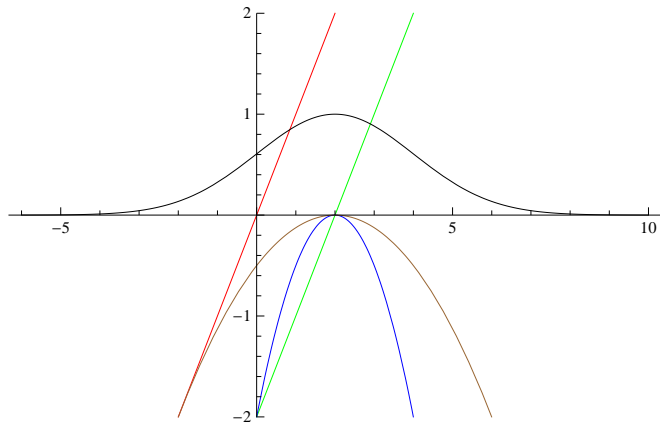
is replaced by a quadratic form:

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



# 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:  $x \mapsto \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$

## 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  $\text{Cov}[X] = \Sigma$ . In other words, a Gaussian is parameterized by its covariance.

## Observation

Since  $\text{Cov}[X_i, X_j] = \text{Cov}[X_j, X_i]$ , the covariance matrix is symmetric.

## What is the eigenstructure of $\Sigma$ ?

- ▶ We know:  $\Sigma$  symmetric  $\Rightarrow$  there is an eigenvector ONB
- ▶ Call the eigenvectors in this ONB  $\xi_1, \dots, \xi_m$  and their eigenvalues  $\lambda_1, \dots, \lambda_m$
- ▶ We can rotate the coordinate system to  $\xi_1, \dots, \xi_m$ . In the new coordinate system, letting the matrix  $Q = [\xi_1, \dots, \xi_m]$ , the covariance  $\Sigma$  transforms to

$$Q^T \Sigma Q = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_m \end{pmatrix} = \text{diag}(\lambda_1, \dots, \lambda_m)$$



# EXAMPLE

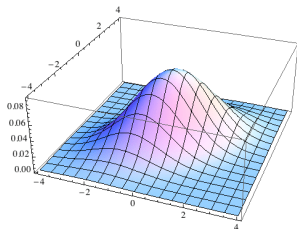
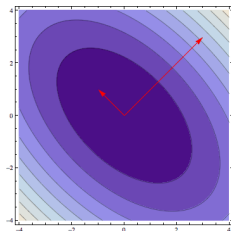
## Quadratic form

$$\langle \mathbf{x}, \Sigma \mathbf{x} \rangle \quad \text{with} \quad \Sigma = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

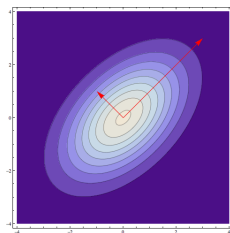
The eigenvectors are  $\frac{1}{\sqrt{2}}(1, 1)$  and  $\frac{1}{\sqrt{2}}(-1, 1)$  with eigenvalues 3 and 1.

## Gaussian density

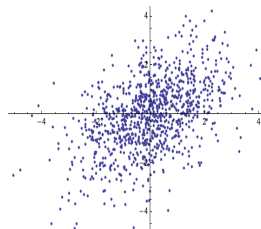
$p(\mathbf{x}; \boldsymbol{\mu}, \Sigma)$  with  $\boldsymbol{\mu} = (0, 0)$ .



Density graph



Density contour



1000 sample points

# INTERPRETATION

## Meaning of the random variables $\xi_i$

For any Gaussian  $p(\mathbf{x}; \boldsymbol{\mu}, \Sigma)$ , we can

1. shift the origin of the coordinate system into  $\boldsymbol{\mu}$
2. rotate the coordinate system to the eigen-ONB of  $\Sigma$ , namely  $Q = [\xi_1, \dots, \xi_m]$ .

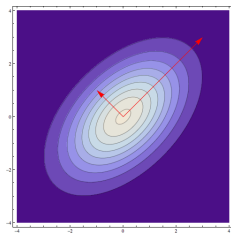
In this new coordinate system, the Gaussian has covariance matrix

$$Q^\top \Sigma Q = \text{diag}(\lambda_1, \dots, \lambda_m)$$

where  $\lambda_i$  are the eigenvalues of  $\Sigma$ .

## Gaussian in the new coordinates

A Gaussian vector  $X$  represented in the new coordinates consists of  $m$  *independent* 1D Gaussian variables  $X'_i$ . Each  $X'_i$  has mean 0 and variance  $\lambda_i$ .



# SHRINKAGE

## Robustness

- ▶ Least squares works only if  $\tilde{\mathbf{X}}$  has full column rank, i.e. if  $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$  is invertible.
- ▶ If  $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$  *almost* not invertible, least squares is numerically unstable.  
Statistical consequence: High variance of predictions.

## Not suited for high-dimensional data

- ▶ Modern problems: Many dimensions/features/predictors (possibly thousands)
- ▶ Only a few of these may be important  
→ need some form of feature selection
- ▶ Least squares:
  - ▶ Treats all dimensions equally
  - ▶ Relevant dimensions are averaged with irrelevant ones
  - ▶ Consequence: Signal loss

## Regularity

A matrix which is not invertible is also called a **singular** matrix. A matrix which is invertible (not singular) is called **regular**.

## In computations

Numerically, matrices can be "almost singular". Intuition:

- ▶ A singular matrix maps an entire linear subspace into a single point.
- ▶ If a matrix maps points far away from each other to points very close to each other, it almost behaves like a singular matrix.

# REGULARITY OF SYMMETRIC MATRICES

Recall: A positive semi-definite matrix  $A$  is singular  $\Leftrightarrow$  smallest eigenvalue is 0

## Illustration

If smallest eigenvalue  $\lambda_{\min} > 0$  but very small (say  $\lambda_{\min} \approx 10^{-10}$ ):

- ▶ Suppose  $x_1, x_2$  are two points in subspace spanned by  $\xi_{\min}$  with  $\|x_1 - x_2\| \approx 1000$ .
- ▶ Image under  $A$ :  $\|Ax_1 - Ax_2\| \approx 10^{-7}$

## In this case

- ▶  $A$  has an inverse, but  $A$  behaves almost like a singular matrix
- ▶ The inverse  $A^{-1}$  can map almost identical points to points with large distance, i.e.

small change in input  $\rightarrow$  large change in output

$\rightarrow$  unstable behavior

## Consequence for Statistics

If a statistical prediction involves the inverse of an almost-singular matrix, the predictions become unreliable (high variance).

## Recall: Prediction in linear regression

For a point  $\mathbf{x}_{\text{new}} \in \mathbb{R}^d$ , we predict the corresponding function value as

$$\hat{y}_{\text{new}} = \langle \hat{\beta}, (1, \mathbf{x}) \rangle = (\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^\top \mathbf{y}$$

## Effect of unstable inversion

- ▶ Suppose we choose an arbitrary training point  $\tilde{\mathbf{x}}_i$  and make a small change to its response value  $\tilde{y}_i$ .
- ▶ Intuitively, that should not have a big impact on  $\hat{\beta}$  or on prediction.
- ▶ If  $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$  is almost singular, a small change to  $\tilde{y}_i$  can prompt a huge change in  $\hat{\beta}$ , and hence in the predicted value  $\hat{y}_{\text{new}}$ .

# MEASURING REGULARITY (OF SYMMETRIC MATRICES)

## Symmetric matrices

Denote by  $\lambda_{\max}$  and  $\lambda_{\min}$  the eigenvalues of  $A$  with largest/smallest *absolute* value. If  $A$  is symmetric, then

$$A \text{ regular} \iff |\lambda_{\min}| > 0.$$

## Idea

- ▶ We can use  $|\lambda_{\min}|$  as a measure of regularity:

$$\text{larger value of } \lambda_{\min} \iff \text{"more regular" matrix } A$$

- ▶ We need a notion of scale to determine whether  $|\lambda_{\min}|$  is large.
- ▶ The relevant scale is how  $A$  scales a vector. Maximal scaling coefficient:  $\lambda_{\max}$ .

## Regularity measure

$$c(A) := \frac{|\lambda_{\min}|}{|\lambda_{\max}|}$$

The function  $c(\cdot)$  is called the **spectral condition** ("spectral" since the set of eigenvalues is also called the "spectrum").



## Objective

Ridge regression is a modification of least squares. We try to make least squares more robust if  $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$  is almost singular.

## Ridge regression solution

The ridge regression solution to a linear regression problem is defined as

$$\hat{\boldsymbol{\beta}}^{\text{ridge}} := (\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}} + \lambda \mathbf{I})^{-1} \tilde{\mathbf{X}}^\top \mathbf{y}$$

$\lambda$  is a tuning parameter.

## Recall

$\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}} \in \mathbb{R}^{(d+1) \times (d+1)}$  is positive definite.

## Spectral shift

Suppose  $\xi_1, \dots, \xi_{d+1}$  are EVectors of  $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$  with EValues  $\lambda_1, \dots, \lambda_{d+1}$ .

Then:

$$(\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}} + \lambda \mathbb{I})\xi_i = (\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}})\xi_i + \lambda \mathbb{I}\xi_i = (\lambda_i + \lambda)\xi_i$$

Hence:  $(\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}} + \lambda \mathbb{I})$  is positive definite with eigenvalues  $\lambda_1 + \lambda, \dots, \lambda_{d+1} + \lambda$ .

## Conclusion

$\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}} + \lambda \mathbb{I}$  is a *regularized* version of  $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$ .

## Effect of regularization

- ▶ We deliberately distort prediction:
  - ▶ If least squares ( $\lambda = 0$ ) predicts perfectly, the ridge regression prediction has an error that increases with  $\lambda$ .
  - ▶ Hence: Biased estimator, bias increases with  $\lambda$ .
- ▶ Spectral shift regularizes matrix  $\rightarrow$  decreases variance of predictions.

## Bias-variance trade-off

- ▶ We decrease the variance (improve robustness) at the price of incurring a bias.
- ▶  $\lambda$  controls the trade-off between bias and variance.

## Recall: Simple linear regression

- ▶ Linear regression solution was defined as minimizer of  $L(\boldsymbol{\beta}) := \|\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\boldsymbol{\beta}\|^2$
- ▶ We have so far defined ridge regression only directly in terms of the estimator  $\hat{\boldsymbol{\beta}}^{\text{ridge}} := (\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}} + \lambda \mathbb{I})^{-1} \tilde{\mathbf{X}}^\top \mathbf{y}$ .
- ▶ To analyze the method, it is helpful to understand it as an optimization problem.
- ▶ We ask: Which function  $L'$  does  $\hat{\boldsymbol{\beta}}^{\text{ridge}}$  minimize?

## Ridge regression as an optimization problem

$$\hat{\boldsymbol{\beta}}^{\text{ridge}} = \arg \min_{\boldsymbol{\beta}} \{ \|\mathbf{y} - \tilde{\mathbf{X}}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_2^2 \}$$

# REGRESSION WITH PENALTIES

## Penalty terms

Recall:  $\|\mathbf{y} - \tilde{\mathbf{X}}\boldsymbol{\beta}\|_2^2 = \sum_i L^{\text{se}}(y_i, f(\tilde{\mathbf{x}}_i; \boldsymbol{\beta}))$ , so ridge regression is of the form

$$L'(\boldsymbol{\beta}) = \sum_i L^{\text{se}}(y_i, f(\tilde{\mathbf{x}}_i; \boldsymbol{\beta})) + \lambda \|\boldsymbol{\beta}\|^2$$

The term  $\|\boldsymbol{\beta}\|^2$  is called a **penalty term**.

## Penalized fitting

The general structure of the optimization problem is

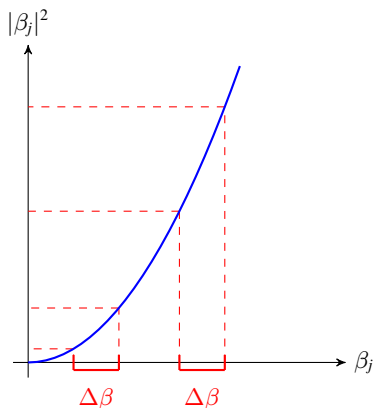
$$\text{total cost} = \text{goodness-of-fit term} + \text{penalty term}$$

Penalty terms make solutions we would like to discourage more expensive.

What kind of solutions does the choice  $\|\boldsymbol{\beta}\|^2$  favor or discourage?

# QUADRATIC PENALTIES

- ▶ A quadratic penalty implies that the reduction in cost we can achieve depends on the magnitude of  $\beta_j$ .
- ▶ Suppose we reduce  $\beta_j$  by a fixed amount  $\Delta\beta$ .
- ▶ Recall that the effect on the regression function is *linear*. The fitting cost (squared error) is quadratic, but in the *error*, not in  $\beta$ .
- ▶ Consequence: Optimization algorithm will favor vectors  $\beta$  whose *entries all have similar size*.



## Setting

- ▶ Regression problem with  $n$  data points  $\tilde{\mathbf{x}}_i$  in  $\mathbb{R}^D$ .
- ▶  $D$  may be very large (much larger than  $n$ ).
- ▶ Goal: Select a small subset of  $d \ll D$  dimensions and discard the rest.
- ▶ In machine learning lingo: Feature selection for regression.

## How do we switch off a dimension?

- ▶ In linear regression: Each entry of  $\boldsymbol{\beta}$  corresponds to a dimension in data space.
- ▶ If  $\beta_k = 0$ , the prediction is

$$f(\mathbf{x}, \boldsymbol{\beta}) = \beta_0 + \beta_1 x_1 + \dots + 0 \cdot x_k + \dots + \beta_D x_D ,$$

so the prediction does not depend on dimension  $k$ .

- ▶ Feature selection: Find a solution  $\boldsymbol{\beta}$  that (1) predicts well and (2) has only a small number of non-zero entries.
- ▶ A solution in which all but a few entries vanish is called a **sparse** solution.

## Penalization approach

Find a penalty term which discourages non-sparse solutions.

## Can quadratic penalty help?

- ▶ Suppose  $\beta_k$  is large, all other  $\beta_j$  are small but non-zero.
- ▶ Sparsity: Penalty should keep  $\beta_k$ , discard others (i.e. push other  $\beta_j$  to zero)
- ▶ Quadratic penalty: Will favor entries  $\beta_j$  which all have similar size  
→ pushes  $\beta_k$  towards small value.

Overall, a quadratic penalty favors many small, but non-zero values.

## Solution

Sparsity can be achieved using *linear* penalty terms.



## Sparse regression

$$\boldsymbol{\beta}^{\text{lasso}} := \arg \min_{\boldsymbol{\beta}} \{ \|\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1 \}$$

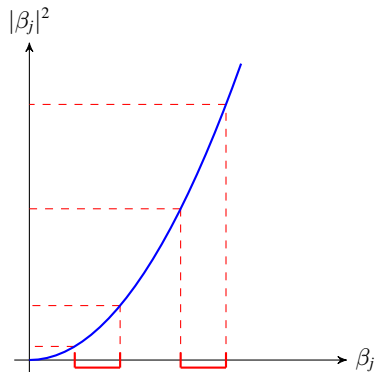
where

$$\|\boldsymbol{\beta}\|_1 := \sum_{j=1}^D |\beta_j|$$

The regression method which determines  $\boldsymbol{\beta}^{\text{lasso}}$  is also called the LASSO (for "Least Absolute Shrinkage and Selection Operator").

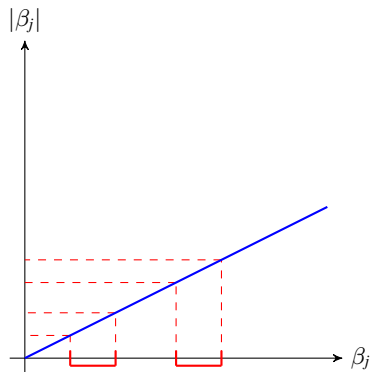
# QUADRATIC PENALTIES

## Quadratic penalty



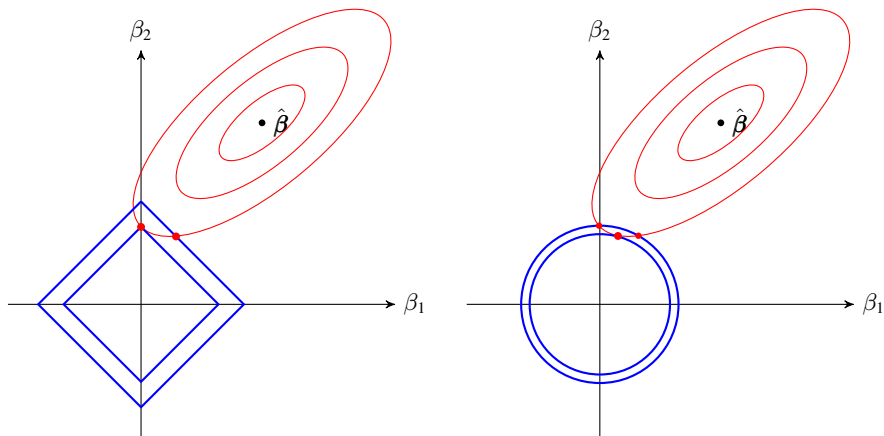
Reducing a large value  $\beta_j$  by a fixed amount achieves a large cost reduction.

## Linear penalty



Cost reduction does not depend on the magnitude of  $\beta_j$ .

# RIDGE REGRESSION VS LASSO



- ▶ Red: Contours of  $\|\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\boldsymbol{\beta}\|_2^2$
- ▶ Blue: Contours of  $\|\boldsymbol{\beta}\|_1$  (left) and  $\|\boldsymbol{\beta}\|_2$  (right)

## $\ell_p$ -norms

$$\|\boldsymbol{\beta}\|_p := \left( \sum_{j=1}^D |\beta_j|^p \right)^{\frac{1}{p}} \quad \text{for } 0 < p \leq \infty$$

is called the  $\ell_p$ -norm.

## $\ell_p$ -regression

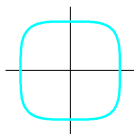
The penalized linear regression problem

$$\boldsymbol{\beta}^{\ell_p} := \arg \min_{\boldsymbol{\beta}} \{ \|\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_p^p \}$$

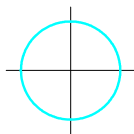
is also referred to as  $\ell_p$ -**regression**. We have seen:

- ▶  $\ell_1$ -regression = LASSO
- ▶  $\ell_2$ -regression = ridge regression

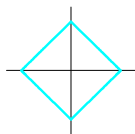
# $\ell_p$ PENALIZATION TERMS



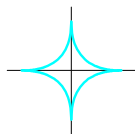
$p = 4$



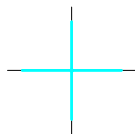
$p = 2$



$p = 1$



$p = 0.5$



$p = 0.1$

$p$	Behavior of $\ \cdot\ _p$
$p = \infty$	Norm measures largest absolute entry, $\ \boldsymbol{\beta}\ _\infty = \max_j \ \beta_j\ $
$p > 2$	Norm focusses on large entries
$p = 2$	Large entries are expensive; encourages similar-size entries.
$p = 1$	Encourages sparsity.
$p < 1$	Encourages sparsity as for $p = 1$ (note "pointy" behavior on the axes), but contour set not convex.
$p \rightarrow 0$	Simply records whether an entry is non-zero, i.e. $\ \boldsymbol{\beta}\ _0 = \sum_j \mathbb{I}\{\beta_j \neq 0\}$

## Ridge regression

Recall: Solution can be computed directly as  $\hat{\boldsymbol{\beta}}^{\text{ridge}} := (\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}} + \lambda \mathbb{I})^{-1} \tilde{\mathbf{X}}^\top \mathbf{y}$ . There is no similar formula for the  $\ell_1$  case.

## Solution of $\ell_1$ problem

By convex optimization.

# SUMMARY: REGRESSION

## Methods we have discussed

- ▶ Linear regression with least squares
- ▶ Ridge regression, Lasso, and other  $\ell_p$  penalties

Note: All of these are linear. The solutions are hyperplanes. The different methods differ only in how they *place* the hyperplane.

## Ridge regression

Suppose we obtain two training samples  $\mathcal{X}_1$  and  $\mathcal{X}_2$  from the same distribution.

- ▶ Ideally, the linear regression solutions on both should be (nearly) identical.
- ▶ With standard linear regression, the problem may not be solvable (if  $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$  not invertible).
- ▶ Even if it is solvable, if the matrices  $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$  are close to singular (small spectral condition  $c(\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}})$ ), then the two solutions can differ significantly.
- ▶ Ridge regression stabilizes the inversion of  $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$ . Consequences:
  - ▶ Regression solutions for  $\mathcal{X}_1$  and  $\mathcal{X}_2$  will be almost identical if  $\lambda$  sufficiently large.
  - ▶ The price we pay is a bias that grows with  $\lambda$ .

# SUMMARY: REGRESSION

## Lasso

- ▶ The  $\ell_1$ -constraint "switches off" dimensions; only some of the entries of the solution  $\beta^{\text{lasso}}$  are non-zero (sparse  $\beta^{\text{lasso}}$ ).
- ▶ This variable selection also stabilizes  $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$ , since we are effectively inverting only along those dimensions which provide sufficient information.
- ▶ No closed-form solution; use numerical optimization.

## Formulation as optimization problem

Method	$f(\beta)$	$g(\beta)$	Solution method
Least squares	$\ \tilde{\mathbf{y}} - \tilde{\mathbf{X}}\beta\ _2^2$	0	Analytic solution exists if $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$ invertible
Ridge regression	$\ \tilde{\mathbf{y}} - \tilde{\mathbf{X}}\beta\ _2^2$	$\ \beta\ _2^2 - t$	Analytic solution exists
Lasso	$\ \tilde{\mathbf{y}} - \tilde{\mathbf{X}}\beta\ _2^2$	$\ \beta\ _1 - t$	Numerical optimization



# MODEL BIAS AND VARIANCE

# OVERVIEW

- ▶ We have already encountered the fact that we can trade off model flexibility against stability of estimates (e.g. shrinkage).
- ▶ To make this effect a bit more precise, we have to discuss the type of errors that we encounter in estimation problems.
- ▶ In this context, it is useful to interpret models as sets of probability distributions.

# SPACE OF PROBABILITY DISTRIBUTIONS

## The space of probability measure

We denote the set of probability distributions on  $\mathbf{X}$  by  $\mathbf{M}(\mathbf{X})$ .

Example:  $\mathbf{X} = \{a, b, c\}$

- ▶ We write  $\delta_{\{a\}}$  for the distribution with

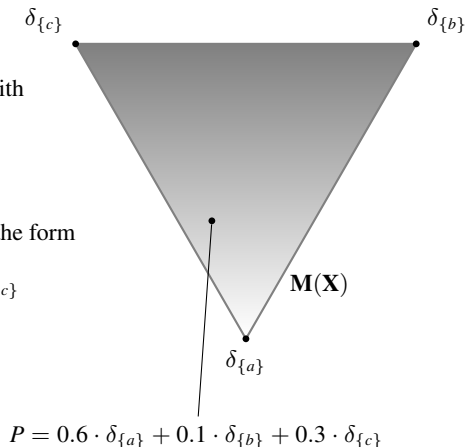
$$\Pr\{X = a\} = 1,$$

similarly for  $b$  and  $c$ .

- ▶ Every distribution  $P \in \mathbf{M}(\mathbf{X})$  is of the form

$$P = c_a \delta_{\{a\}} + c_b \delta_{\{b\}} + c_c \delta_{\{c\}}$$

with  $c_a + c_b + c_c = 1$ .



## Dirac distributions

A **Dirac distribution**  $\delta_x$  is a probability distribution which concentrates all its mass at a single point  $x$ . A Dirac  $\delta_x$  is also called a **point mass**.

**Note:** This means that there is no uncertainty in a random variable  $X$  with distribution  $\delta_x$ : We know before we even sample that  $X = x$  with probability 1.

## Working with a Dirac

The defining property of a Dirac is that

$$\int_{\mathbf{X}} f(x) \delta_{x_0}(dx) = f(x_0)$$

for every (integrable) function  $f$ .

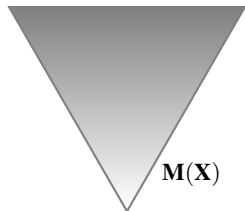
# VISUALIZATION OF $\mathbf{M}(\mathbf{X})$

## $\mathbf{M}(\mathbf{X})$ for an infinite set $\mathbf{X}$

- ▶ If  $\mathbf{X}$  is infinite (e.g.  $\mathbf{X} = \mathbb{R}^d$ ), the distributions  $\delta_{\{a\}}, \delta_{\{b\}}, \delta_{\{c\}}$  above are replaced by Diracs  $\delta_{\mathbf{x}}$  (one for each  $\mathbf{x} \in \mathbf{X}$ ).
- ▶ The distributions  $\delta_{\mathbf{x}}$  still have the property that they cannot be represented as convex combinations.
- ▶ Hence: Each  $\delta_{\mathbf{x}}$  is an extreme point of  $\mathbf{M}(\mathbf{X})$ .
- ▶ We need one additional dimension for each point  $\mathbf{x} \in \mathbf{X}$ .
- ▶ Roughly speaking,  $\mathbf{M}(\mathbf{X})$  is the infinite-dimensional analogue of a triangle or tetraeder, with its extreme points labelled by the points in  $\mathbf{X}$ .

## Visualization

In the following, we will still visualize  $\mathbf{M}(\mathbf{X})$  as a triangle, but keep in mind that *this is a cartoon*.



# THE EMPIRICAL DISTRIBUTION

## The empirical distribution

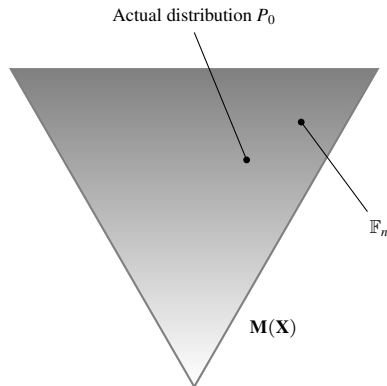
If  $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$  is a sample, its empirical distribution is

$$\mathbb{F}_n := \sum_{i=1}^n \frac{1}{n} \delta_{\mathbf{x}_i} .$$

## The sample as a distribution

Using  $\mathbb{F}_n$ , we can regard the sample as an element of the space  $\mathbf{M}(\mathbf{X})$ .

For i.i.d. samples, the law of large numbers says that  $\mathbb{F}_n$  converges to the true distribution as  $n \rightarrow \infty$ .



# EXAMPLE: PLUG-IN ESTIMATORS

A simple application of the empirical distribution are plug-in estimators.

## Integral statistics

Many of the most common statistics can be written in the form

$$S[p] = \int_{\mathbf{x}} f(\mathbf{x})p(\mathbf{x})d\mathbf{x} .$$

Examples: Expectation of  $p$  (where  $f(\mathbf{x}) = \mathbf{x}$ ), variance of  $p$  (where  $f(\mathbf{x}) = (\mathbf{x} - \mu)^2$ ), etc.

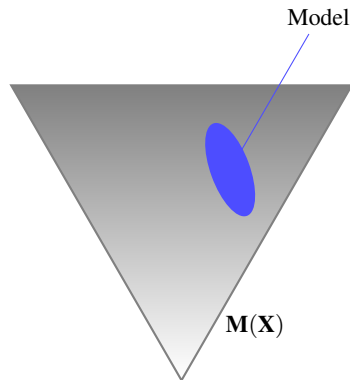
## Plug-in estimator

One way to estimate  $S$  from a sample  $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$  is to "plug in" the empirical distribution  $\mathbb{F}_n$  for the true distribution  $p$ :

$$\hat{S} := \int_{\mathbf{x}} f(\mathbf{x})\mathbb{F}_n(d\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n f(\mathbf{x}_i)$$

This estimator is called the **plug-in** estimator of  $S$ .

# STATISTICAL MODELS



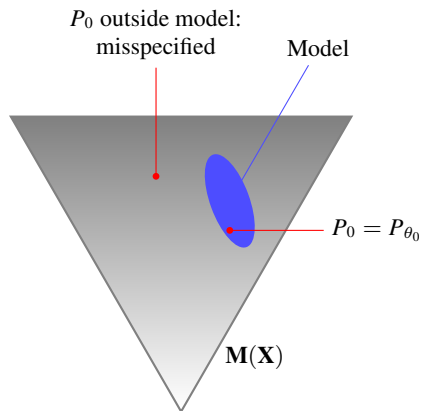
Recall that a statistical model with parameter space  $\mathcal{T}$  is a set

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

of distributions. In particular, a model is a subset of  $\mathbf{M}(\mathbf{X})$ .



# MODEL MISSPECIFICATION

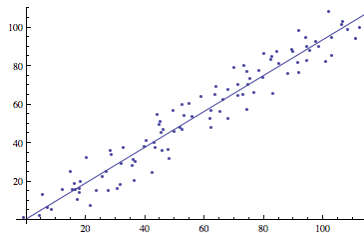


Suppose the observed data is generated by a "true" distribution  $P_0$ .

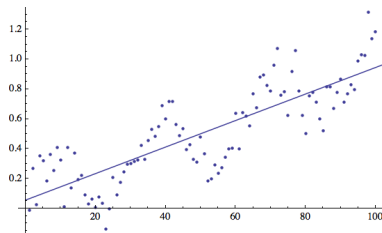
- ▶ We say that the model is **correctly specified** if  $P_0 \in \mathcal{P}$ .
- ▶ If  $P_0 \notin \mathcal{P}$ , the model is **misspecified**.

# MODEL MISSPECIFICATION

## Example: Regression



Correctly specified



Misspecified

## Implications

- ▶ If the model is correctly specified, we can in principle find a parameter value  $\theta \in \mathcal{T}$  which fully explains the data.
- ▶ Finding  $\theta$  still requires a valid estimation procedure.
- ▶ In most cases, we can live with misspecification, provided that the approximation error can be controlled.

## Model complexity

If our only objective is to avoid misspecification, we should make the model (the subset  $\mathcal{P}$  of  $\mathbf{M}(\mathbf{X})$ ) as large as possible. A larger set  $\mathcal{P}$  corresponds to a model that is more flexible.

## Bias vs. Variance

- ▶ Misspecification means that, no matter how much data we observe, our estimated model never completely explains the data. This can be interpreted as a form of bias.
- ▶ To avoid misspecification, we can make the model more flexible.
- ▶ We have already seen how estimates in more flexible models tend to vary more between sample sets (higher variance).

Thus, we can decrease bias at the expense of increasing variance, and vice versa. This phenomenon is called the **bias-variance trade-off**.

# MEASURING MODEL COMPLEXITY

## In parametric models

- ▶ A fundamental measure of model complexity is the number of **degrees of freedom** (d.o.f.).
- ▶ This is roughly the dimension of the parameter space (= the number of independent scalar parameters), provided the parametrization is chosen such that the entries of the parameter vector are reasonably independent of each other.
- ▶ For example, a Gaussian scale model on  $\mathbb{R}^d$  (unknown mean, fixed variance) has  $d$  degrees of freedom, a Gaussian model with unknown mean and variance has  $d + d(d - 1)/2$ .

## Remark: Nonparametric models

- ▶ In nonparametric models (= infinite-dimensional parameter space), measuring model complexity is much harder.
- ▶ Tools to solve this problem are developed in two closely related research fields called Statistical Learning Theory (in Machine Learning) and Empirical Process Theory (in Statistics).

# EXAMPLE: REGRESSION WITH ADDITIVE NOISE

## Model

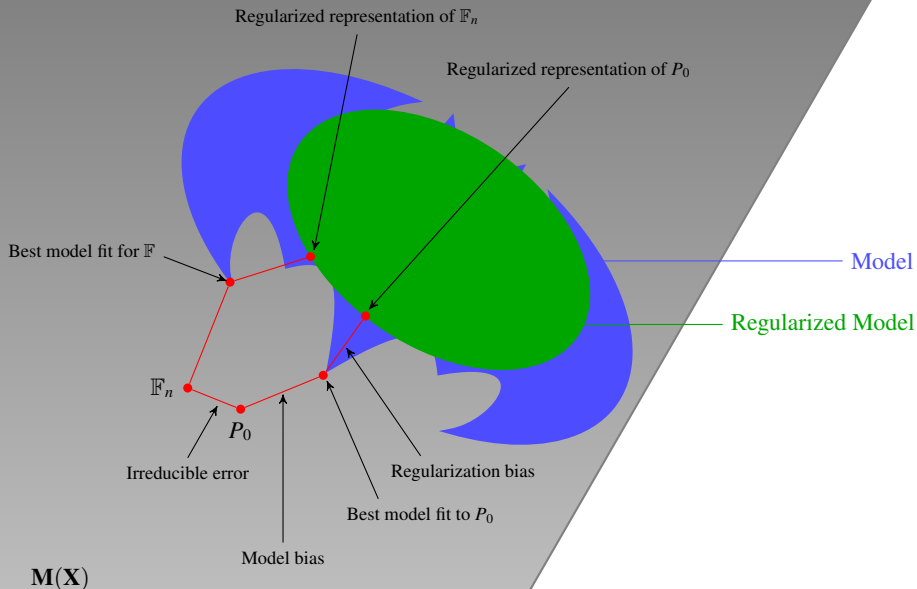
$$Y = f(X) + \varepsilon \quad \text{with} \quad \mathbb{E}[\varepsilon] = 0 \quad \text{and} \quad \text{Var}[\varepsilon] = \sigma^2 .$$

We assume that  $f \in \mathcal{F}$ , where  $\mathcal{F}$  is some class of functions. Linear regression is the special case where  $\mathcal{F}$  is the set of affine functions.

## Bias and Variance

$$\begin{aligned} \text{Prediction error}(x_0) &= \mathbb{E}[(Y - \hat{f}(x_0))^2 | X = x_0] \\ &= \sigma^2 + (\mathbb{E}[\hat{f}(x_0)] - f(x_0))^2 + \mathbb{E}[\hat{f}(x_0) - \mathbb{E}[\hat{f}(x_0)]]^2 \\ &= \sigma^2 + \text{Bias}(\hat{f}(x_0))^2 + \text{Var}[\hat{f}(x_0)] \\ &= \underbrace{\text{Irreducible error}}_{\text{This is due to } \mathbb{E}_n \neq P_0.} + \underbrace{\text{Bias}^2}_{\substack{\uparrow \\ \text{Decreases with model flexibility.}}} + \underbrace{\text{Variance}}_{\substack{\uparrow \\ \text{Increases with model flexibility.}}} \end{aligned}$$

# TYPES OF ERRORS



# SPECIFICALLY: LINEAR REGRESSION

## Unregularized case

In linear least-squares regression, the variance term is

$$\text{Var}[\hat{f}(x_0)] = \|(\tilde{\mathbf{X}}^t \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^t x_0\| \sigma_\varepsilon^2$$

## Ridge regression

In ridge, the variance term is

$$\text{Var}[\hat{f}(x_0)] = \|(\tilde{\mathbf{X}}^t \tilde{\mathbf{X}} + \lambda \mathbb{I})^{-1} \tilde{\mathbf{X}}^t x_0\| \sigma_\varepsilon^2$$

This term is generally smaller than in the unregularized case, but the corresponding bias term is larger.

## Model complexity

- ▶ Model choice has to trade off stability (low variance) vs flexibility (low bias).
- ▶ It can be beneficial (in terms of prediction error) to permit a bias if this decreases the variance.
- ▶ Bias and variance terms combine to form prediction error.

## How does cross validation fit in?

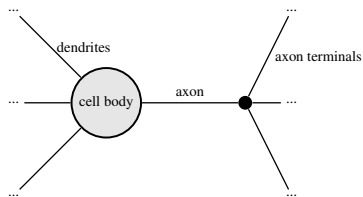
- ▶ Cross validation estimates the prediction error.
- ▶ A high variance of estimates will typically be reflected in a high variance between estimates on different blocks.



# NEURAL NETWORKS

# OVERVIEW

Your brain is made up of many interconnected neurons.



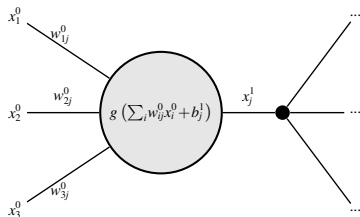
Highly schematic information flow:

- ▶ Signals from other neurons enter into the neuron along the dendrites.
- ▶ These inputs are combined in some way in the cell body.
- ▶ An outgoing signal is sent out the axon.
- ▶ The axon terminals transmit that signal to the dendrites of other neurons.

In machine learning, ‘neural networks’ are a fairly general family of methods based on this basic computational schematic.

# FROM NEURON TO NEURAL UNIT

Basic computational building block (think perceptron):

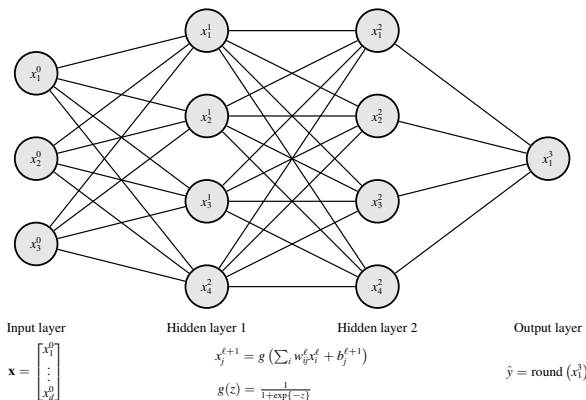


Data flow:

- ▶ Inputs  $x_i^0$  enter into neuron  $j$  along weighted edges  $w_{ij}^0$ .
- ▶ These weighted inputs are summed with some bias  $b_j^1$ .
- ▶ ‘Activation’  $g$  transforms the sum. Typical choice (of many):  $g(z) = \frac{1}{1+\exp(-z)}$ .
- ▶ The result  $x_j^1$  is transmitted to layer 2, the next layer of neurons.

With enough layers and enough neurons per layer, the network is a *universal function approximator*: any function can be fit (given enough data, of course).

# FROM NEURAL UNIT TO NEURAL NETWORK



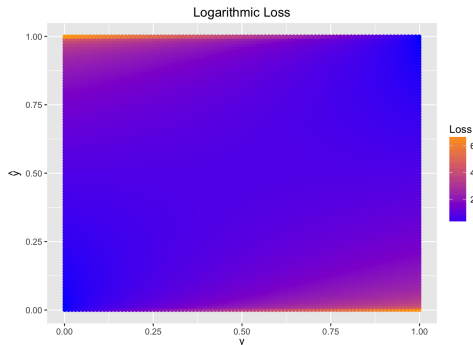
- ▶ Given weights  $W = \{w_{ij}^{\ell}, b_j^{\ell}\}_{i,j,\ell}$ , this is just a classifier  $f_W : \mathbb{R}^d \rightarrow \{0, 1\}$ .
- ▶  $\hat{y} = x_1^3 \rightarrow$  regression network (or  $\hat{y} = [x_1^L \dots x_D^L]^{\top}$ ).
- ▶ As with every other method, the work is to optimize the  $w_{ij}^{\ell}$  and  $b_j^{\ell}$ .

# LOSS FUNCTION

First, we must specify an objective to be optimized. In neural networks the typical choice is the *cross-entropy* or *log loss*:

$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

where  $\hat{y} = x_1^3$  in our previous example, and  $y \in \{0, 1\}$  is the training label.



As usual, we minimize empirical risk:  $\arg \min_W C(W) = \frac{1}{n} \sum_{i=1}^n L(y_i, \hat{y}_i)$ .

# BACKPROPAGATION

As usual we hope to perform gradient descent:  $\nabla_W C(W) = \left[ \frac{\partial}{\partial w_{ij}^\ell} C(W) \right]_{ij\ell}$ .

Backpropagation is clever use of the chain rule to reduce huge computational cost.

- ▶ Start at the output:  $x_1^L = g\left(\sum_i w_{i1}^{L-1} x_i^{L-1}\right) \triangleq g(z_1^L)$ .
- ▶ Differentiate:  $\delta_1^L \triangleq \frac{\partial C}{\partial z_1^L} = \frac{\partial C}{\partial x_1^L} g'(z_1^L)$ .
- ▶ Previous layer:  $\forall i, \delta_i^{L-1} = \frac{\partial C}{\partial z_i^{L-1}} \frac{\partial z_1^L}{\partial z_i^{L-1}} = \delta_1^L \frac{\partial z_1^L}{\partial z_i^{L-1}} = \delta_1^L w_{i1}^{L-1} g'(z_1^L)$ .
- ▶ Propagate *back* through all layers:

$$\delta_i^\ell = \frac{\partial C}{\partial z_i^\ell} = \sum_{j=1}^{|\ell+1|} \frac{\partial C}{\partial z_j^{\ell+1}} \frac{\partial z_j^{\ell+1}}{\partial z_i^\ell} = \sum_{j=1}^{|\ell+1|} \delta_j^{\ell+1} w_{ij}^{\ell+1} g'(z_j^{\ell+1}).$$

- ▶ Finally:

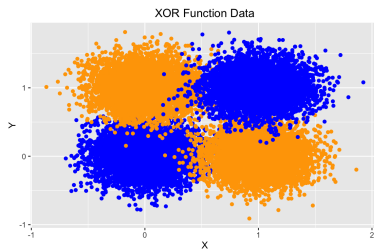
$$\frac{\partial C}{\partial w_{ij}^\ell} = \frac{\partial C}{\partial z_j^{\ell+1}} \frac{\partial z_j^{\ell+1}}{\partial w_{ij}^\ell} = \delta_j^{\ell+1} x_i^\ell.$$

- ▶ Backpropagation is simply an efficient way to calculate the gradient.

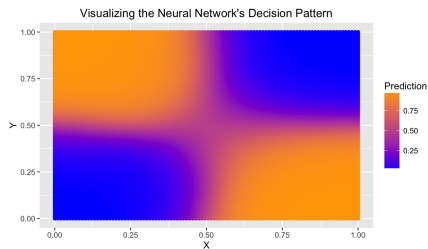
# EXAMPLE: THE XOR PROBLEM

- ▶ Late 1960s: proof that perceptron can not learn an XOR function, which led to the misconception that multilayer neural networks were not generally useful.
- ▶ Late 1980s: neural network popularity again, followed by another decline.
- ▶ Early 2010s: third and largest rise to prominence.

Neural network with 1 hidden layer and 20 units (trained with `neuralnet` in R):



Training data.



Output node value.

# EXAMPLE: CLASSIFYING CANCER

The *Wisconsin Breast Cancer Dataset* is a common benchmark for classification algorithms. The goal is to predict the presence or absence of breast cancer based on images of biopsied tissues.

[https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+\(Diagnostic\)](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic))

- ▶ SVM, Naive Bayes, and other algorithms we have studied perform in the 93-97% accuracy range on test data.
- ▶ Train a basic neural network with the same choices as in the XOR example.

Neural network performance (classification rate on test data)			
Units per hidden layer	5	10	15
1 hidden layer	0.965	0.939	0.947
2 hidden layers	0.956	0.895	0.912

- ▶ Performance is top notch but unpredictable (more units/layers  $\neq$  better).
- ▶ Neural networks are general and powerful but not as well understood as many other methods we have studied. Training them well is often considered ‘an art’.



# FROM NEURAL NETWORKS TO DEEP LEARNING

Deep learning = extensions of this idea + massive data + cloud/GPU computing:

- ▶ {Overfitting, runtime, architecture, theory} are all rapidly evolving stories.
- ▶ Convolutional NN: state-of-the-art image classification (ImageNet)
- ▶ Recursive and Long short-term memory NN: speech, text, word embeddings...
- ▶ Deep Q-learning: reinforcement learning NN (best in world(?) at Atari & Go)
- ▶ High-quality Software: [www.tensorflow.org/](http://www.tensorflow.org/) , [deeplearning.net/software/theano/](http://deeplearning.net/software/theano/) , [caffe.berkeleyvision.org/](http://caffe.berkeleyvision.org/) , [torch.ch/](http://torch.ch/) , (today's R) [www.rpubs.com/ashutoshnanda/neuralnets](http://www.rpubs.com/ashutoshnanda/neuralnets)
- ▶ Other reading: [neuralnetworksanddeeplearning.com/](http://neuralnetworksanddeeplearning.com/) , [deeplearning.net/tutorial/](http://deeplearning.net/tutorial/) , [colah.github.io/](http://colah.github.io/) , [karpathy.github.io/2015/05/21/rnn-effectiveness/](http://karpathy.github.io/2015/05/21/rnn-effectiveness/) , ...
- ▶ A few (of many) state-of-the-art examples:

Generating Shakespeare (cf.  $n$ -grams).

KING LEAR:

O, if you were a feeble sight, the courtesy of your law,  
Your sight and several breath, will wear the gods  
With his heads, and my hands are wonder'd at the deeds,  
So drop upon your lordship's head, and your opinion  
Shall be against your honour.

Blending style (Picasso) with content (Gandalf).



# UNSUPERVISED LEARNING

# UNSUPERVISED LEARNING

## In short

- ▶ Label information available → supervised learning (classification, regression)
- ▶ No label information available → **unsupervised learning**

## Problem

- ▶ Try to find structure or patterns in data without knowing a correct solution.
- ▶ By choosing a model, we specify what kind of patterns we are looking for.

## Examples of unsupervised learning problems

- ▶ Dimension reduction
- ▶ Clustering

# DIMENSION REDUCTION

# DIMENSION REDUCTION PROBLEMS

## Setting

- ▶ Given: High-dimensional data  $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^D$
- ▶ Look for: Low-dimensional projection of the data such that important structure in data is preserved

## More precisely

- ▶ Find suitable linear subspace  $V \subset \mathbb{R}^D$  with  $\dim(V) =: d$  small.
- ▶ Compute projection  $\mathbf{x}_j^v$  of each  $\mathbf{x}_j$  onto  $V$

Most common cases:  $d \in \{2, 3\}$  for visualization.

## Assumptions

1. Directions along which uncertainty in data is maximal are most interesting.
2. Uncertainty is measured by variance.

## Method

- ▶ Compute empirical covariance matrix of the data.
- ▶ Compute its EValues  $\lambda_1, \dots, \lambda_D$  and EVectors  $\xi_1, \dots, \xi_D$ .
- ▶ Choose the  $d$  largest EValues, say,  $\lambda_{j_1}, \dots, \lambda_{j_d}$ .
- ▶ Define subspace as  $V := \text{span}\{\xi_{j_1}, \dots, \xi_{j_d}\}$
- ▶ Project data onto  $V$ : For each  $\mathbf{x}_i$ , compute  $\mathbf{x}_i^v := \sum_{j=1}^d \langle \mathbf{x}_i, \xi_j \rangle \xi_j$

This algorithm is called **Principal Component Analysis (PCA)**.

## Notation

- ▶ Empirical mean of data:  $\hat{\mu}_n := \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i$
- ▶ Empirical variance of data (1 dimension):  $\hat{\sigma}_n^2 := \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \hat{\mu}_n)^2$
- ▶ Empirical covariance of data ( $D$  dimensions):  $\hat{\Sigma}_n := \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \hat{\mu}_n)(\mathbf{x}_i - \hat{\mu}_n)^t$

Recall outer product of vectors: Matrix  $(\mathbf{x}\mathbf{x}^t)_{ij} := x_i x_j$

## PCA Idea

Project data onto a direction  $v \in \mathbb{R}^D$  such that the variance of the projected data is maximized.

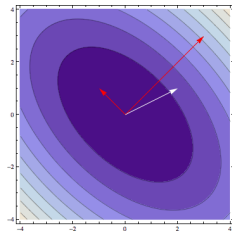
## Claim

The variance of the projected data is given by  $\langle \mathbf{v}, \hat{\Sigma}_n \mathbf{v} \rangle$ .

## Explanation

The projection of  $\mathbf{x}_i$  onto  $\mathbf{v}$  is  $\langle \mathbf{x}_i, \mathbf{v} \rangle$ . Substitute into empirical variance:

$$\begin{aligned} \frac{1}{n} \sum_{i=1}^n (\langle \mathbf{x}_i, \mathbf{v} \rangle - \langle \hat{\mu}_n, \mathbf{v} \rangle)^2 &= \frac{1}{n} \sum_{i=1}^n \langle (\mathbf{x}_i - \hat{\mu}_n), \mathbf{v} \rangle^2 \\ &= \left\langle \underbrace{\left( \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \hat{\mu}_n)(\mathbf{x}_i - \hat{\mu}_n)^t \right)}_{=\hat{\Sigma}_n} \mathbf{v}, \mathbf{v} \right\rangle \end{aligned}$$



Red: Eigenvectors. White:  $\mathbf{v}$ .

## Recall: quadratic forms

The variance along  $\mathbf{v}$  is the value of the quadratic form defined by  $\hat{\Sigma}_n$ , evaluated at  $\mathbf{v}$ .



## PCA as optimization problem

$$\begin{aligned} \max_{\mathbf{v}} \quad & \langle \mathbf{v}, \hat{\Sigma}_n \mathbf{v} \rangle \\ \text{s.t.} \quad & \|\mathbf{v}\|_2 = 1 \end{aligned}$$

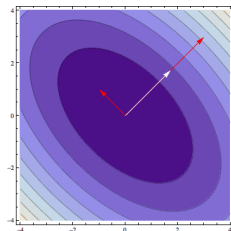
The constraint  $\|\mathbf{v}\|_2 = 1$  ensures that we maximize by adjusting the direction of  $\mathbf{v}$ ; otherwise, we could make  $\langle \mathbf{v}, \hat{\Sigma}_n \mathbf{v} \rangle$  arbitrarily large by scaling  $\mathbf{v}$ .

## Optimization problem: Solution

We know from our discussion of quadratic forms:

$$\langle \mathbf{v}, \hat{\Sigma}_n \mathbf{v} \rangle \text{ maximal} \Leftrightarrow \mathbf{v} \text{ points in direction of } \xi_{\max}$$

where  $\xi_{\max}$  is the EVector associated with the largest EValue.



## Projecting onto 2 dimensions

1. Project onto 1 dimension.
2. Remove that dimension from data, i.e. restrict data to the space orthogonal to  $\mathbf{v}$ .
3. Apply PCA on restricted space.

It is not hard to show that the result is the direction of the EVector associated with the second-largest eigenvalue.

## Projecting onto $d$ dimensions

By iterating the procedure above, we find that the optimal projection onto  $d$  dimensions corresponds to the  $d$  largest EValues.

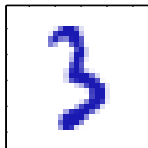
## Summary

The PCA algorithm (=project on the  $d$  "largest" EVectors) can be justified as the projection which maximizes the variance of the projected data.

# PCA: EXAMPLE

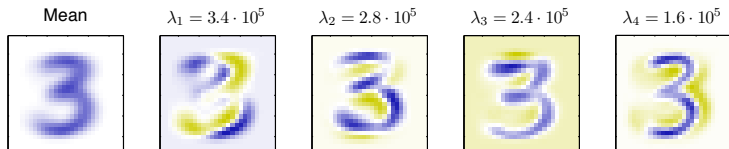
## Again: Digit data

- ▶ Recall:  $\mathbf{x}_i \in \mathbb{R}^{256}$
- ▶ Here: Images representing the number 3.



## Eigenvectors

The mean  $\hat{\mu}_n$  and the EVectors are also elements of  $\mathbb{R}^{256}$ , so we can plot them as images as well.



These are the EVectors for the four largest EValues.

## Principal components

The first few eigenvectors are called **principal components**. They can be regarded as a summary of the main features of the data.

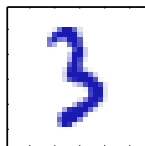
## Using PCA as a compressor

- ▶ To store a digit, we have to store 256 floating point (FP) numbers.
- ▶ If we store its projection onto  $d$  eigenvectors, we have to store:
  1. The  $d$  complete eigenvectors =  $d \cdot 256$  FP numbers.
  2.  $d$  FP numbers per image.
- ▶ For  $n$  large enough, i.e. if  $n \cdot d + d \cdot 256 < n \cdot 256$ , this results in compression of the data.

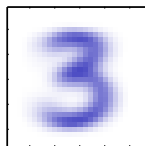
## Lossy data compression

- ▶ From the compressed data, we cannot restore the data completely. Such compression methods are called **lossy compression**.
- ▶ Other examples: JPEG, MP3, etc.
- ▶ Compression methods which completely preserve the data are called **lossless**. (Example: ZIP compression for digital files.)

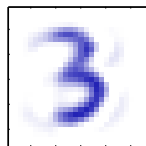
# COMPRESSING DIGITS WITH PCA



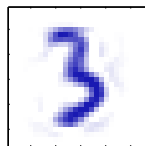
Input



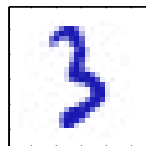
$d = 1$



$d = 10$



$d = 50$



$d = 200$

- ▶ The input image  $\mathbf{x}$  is projected onto each eigenvector  $\xi_i$  to obtain a coefficient  $c_i$ .
- ▶ Then  $\mathbf{x}$  can be represented as

$$\mathbf{x} = \sum_{j=1}^D c_j \xi_j$$

- ▶ A compressed version using  $d$  components is obtained as

$$\mathbf{x}^{(d)} = \sum_{j=1}^d c_j \xi_j$$

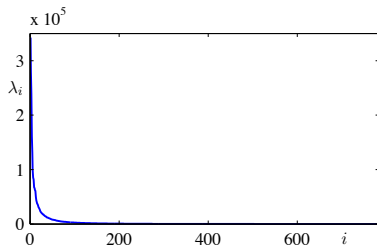
Since  $\mathbf{x}^{(d)} \in \mathbb{R}^{256}$ , we can plot it as an image. These are the images above.

## How many eigenvectors should we use?

- ▶ For visualization: Usually 2 or 3.
- ▶ For approximation or compression: We would like to minimize the approximation error, so we should try to keep all large EValues.

## Eigenvalues in the digit problem

- ▶ Ideally, the curve of the size-ordered EValues shows a clear jump or bent at which we can truncate.
- ▶ Such a jump is called a **spectral gap**.

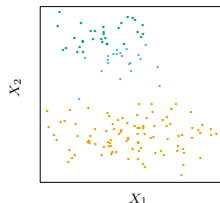


# DATA CLUSTERING

# CLUSTERING

## Problem

- ▶ Given: Data  $x_1, \dots, x_n$ .
- ▶ Assumption: Each data point belongs to exactly one group or class. These groups are called **clusters**.
- ▶ Our task is to find the clusters, given only the data.



## Representation

For  $K$  clusters, we encode assignments to clusters as a vector  $\mathbf{m} \in \{1, \dots, K\}^n$  as

$$m_i = k \quad \Leftrightarrow \quad x_i \text{ assigned to cluster } k$$

## Clustering and classification

Clustering is the "unsupervised" counterpart to classification. There is no training data and no labels, only one, unlabeled data set.

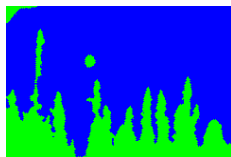
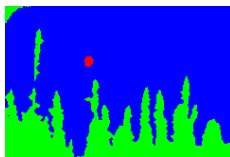
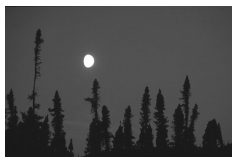


# EXAMPLE: IMAGE SEGMENTATION

## Segmentation

**Image segmentation** is the problem of partitioning an image into "coherent" regions. The problem is not well-posed: Its solution depends on the meaning of "coherent".

## Example



## Segmentation as a clustering problem

- ▶ For each pixel, place a small window around the pixel. Extract features (measurements) from this window. For the  $i$ -th pixel, represent measurements by a vector  $x_i$ .
- ▶ Compute a clustering of the data  $x_1, \dots, x_n$  with  $K$  clusters.
- ▶ Each cluster represents one segment. In the images above, one cluster is colored blue, one green, one red.

# A VERY SIMPLE CLUSTERING ALGORITHM: $K$ -MEANS

## $K$ -means algorithm

- ▶ Randomly choose  $K$  "cluster centers" (the "means")  $\mu_1^{(0)}, \dots, \mu_K^{(0)} \in \mathbb{R}^d$
- ▶ Iterate until convergence ( $j$  = iteration number):
  1. Assign each  $x_i$  to the closest (in Euclidean distance) mean:

$$m_i^{(j+1)} := \arg \min_{k \in \{1, \dots, K\}} \|x_i - \mu_k^{(j)}\|$$

2. Recompute each  $\mu_k^{(j)}$  as the mean of all points assigned to it:

$$\mu_k^{(j+1)} := \frac{1}{|\{i | m_i^{(j+1)} = k\}|} \sum_{i | m_i^{(j+1)} = k} x_i$$

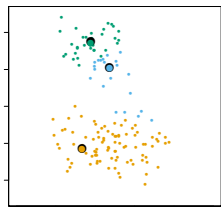
## Convergence Criterion

For example: Terminate when a the total change of the means satisfies

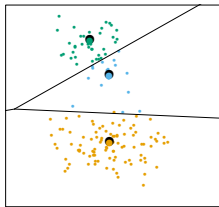
$$\sum_{k=1}^K \|\mu_k^{(j+1)} - \mu_k^{(j)}\| < \tau .$$

The threshold value  $\tau$  is set by the user.

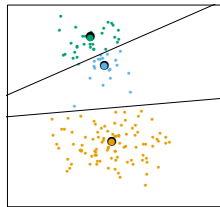
# $K$ -MEANS: ILLUSTRATION



$j = 1$



$j = 2$

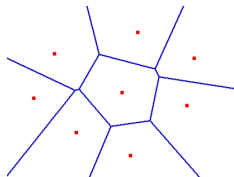


$j = 20$

## Voronoi decomposition

The means  $\mu_k$  partition the space (here  $\mathbb{R}^2$ ) into  $K$  regions. The regions corresponding to  $\mu_k$  is the set of all points closer to  $\mu_k$  than to any other  $\mu_l$ . Such a partition is called a **Voronoi decomposition**.

The  $K$ -means assignment step assigns all data points in the Voronoi region of  $\mu_k$  to cluster  $k$ . The lines in the  $k$ -means example are the boundaries of Voronoi regions.



# K-MEANS: GAUSSIAN INTERPRETATION

## K Gaussians

Consider the following algorithm:

- ▶ Suppose each  $\mu_k$  is the expected value of a Gaussian density  $p(x|\mu_k, \mathbb{I})$  with unit covariance.
- ▶ Start with  $K$  randomly chose means and iterate:
  1. Assign each  $x_i$  to the Gaussian under which it has the highest probability of occurrence (more precisely: highest density value).
  2. Given the assignments, fit  $p(x|\mu_k, \mathbb{I})$  by maximum likelihood estimation of  $\mu_k$  from all points assigned to cluster  $k$ .

## Comparison to K-means

- ▶ Since the Gaussians are spherical with identical covariance, the density  $p(x_i|\mu_k, \mathbb{I})$  is largest for the mean  $\mu_k$  which is closest to  $x_i$  in Euclidean distance.
- ▶ The maximum likelihood estimator of  $\mu_k$  is

$$\hat{\mu}_k := \frac{1}{|\{i|m_i = k\}|} \sum_{i|m_i=k} x_i$$

This is precisely the  $k$ -means algorithm!

# WHAT NEXT

- ▶ We will discuss a more sophisticated version of  $K$ -means called the *Expectation-Maximization (EM) algorithm*.
- ▶ EM gives
  1. A better statistical explanation of what is going on.
  2. A direct generalization to other distributions. We will consider (1) Gaussians with general covariance structure and (2) multinomial distributions.

## Mixture

For a parametric model  $p(x|\theta)$  and a probability density  $q$ , a distribution of the form

$$\pi(x) = \int_{\mathcal{T}} p(x|\theta)q(\theta)d\theta$$

is called a **mixture model**. The distribution given by  $q$  is called the **mixing distribution**.

## Interpretation

Mixtures describe two-stage sampling procedures. We can generate samples from  $\pi$  as follows:

1. Sample  $\theta_i \sim q$ .
2. Sample  $X_i \sim p(\cdot | \theta_i)$ .

The distribution of a sample  $x_1, \dots, x_n$  generated in this manner has density  $\pi$ .

# EXAMPLE: CONTINUOUS MIXTURE

## Example

We are mostly interested *discrete* mixing distributions, but  $\theta$  can be continuous variable, as in the following example.

## Mixture components

1. Sample  $\theta \sim \text{Gamma}(\alpha, \beta)$ .
2. Regard  $\theta$  as an inverse variance  $\frac{1}{\sigma^2} := \theta$  and sample

$$X \sim \text{Normal}(0, \sigma)$$

## Mixture distribution

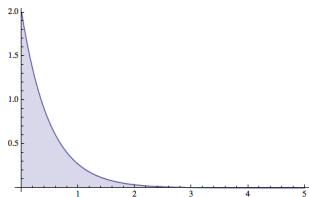
The distribution of  $X$  is the mixture with density

$$\pi(x|0, \nu := \frac{\alpha}{\beta}, \tau := 2\alpha) = \int_{\mathbb{R}_+} p_{\text{Normal}}(x|0, 1/\theta) q_{\text{Gamma}}(\theta|\alpha, \beta) d\theta$$

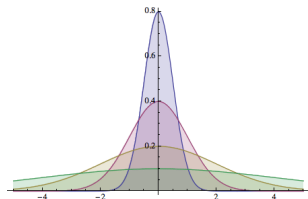
This is **Student's  $t$ -distribution** with parameters 0 (the mean of the normal),  $\nu$ ,  $\tau$ .

# EXAMPLE: CONTINUOUS MIXTURE

## Mixture components

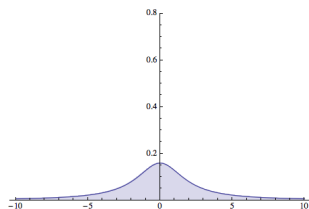


Gamma distribution

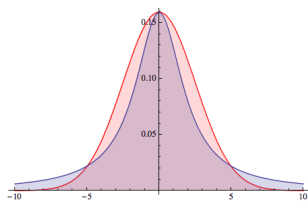


Normal distribution, different variances

## Mixture distribution



The mixture is a Student distribution. Mixing over different variances results in "heavy tails".



Comparison: Normal distribution (red) vs Student distribution (blue)



# FINITE MIXTURES

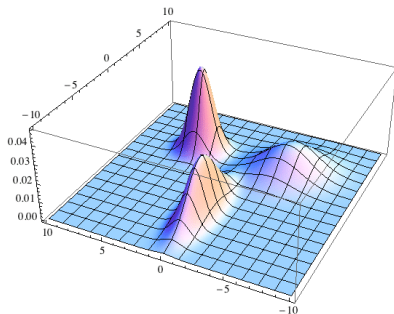
## Finite Mixture Model

A **finite mixture model** is a distribution with density of the form

$$\pi(x) = \sum_{k=1}^K c_k p(x|\theta_k) ,$$

where  $\sum_k c_k = 1$  and  $c_k \geq 0$ .

**Example: Finite mixture of Gaussians**



## Interpretation as mixture

A mixture is of the form

$$\pi(x) = \int_{\mathcal{T}} p(x|\theta)q(\theta)d\theta .$$

We choose  $q$  as

$$q = \sum_{k=1}^K c_k \delta_{\theta_k}$$

for  $K$  fixed values  $\theta_k \in \mathcal{T}$ . Recall that integration against the Dirac distribution  $\delta_{\theta}$  "picks out" the function value at  $\theta$ .

The mixture with mixing distribution  $q$  is therefore

$$\begin{aligned} \pi(x) &= \int_{\mathcal{T}} p(x|\theta) \left( \sum_{k=1}^K c_k \delta_{\theta_k} \right) d\theta = \sum_{k=1}^K c_k \int_{\mathcal{T}} p(x|\theta) \delta_{\theta_k} d\theta \\ &= \sum_{k=1}^K c_k p(x|\theta_k) . \end{aligned}$$

# EXAMPLE: GAUSSIAN MIXTURE

## Specifying component parameters

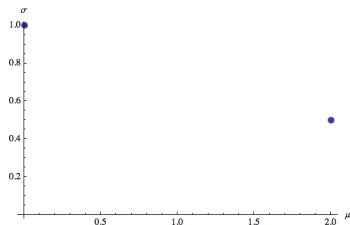
To obtain mixture components with

$$(\mu_1, \sigma_1) = (0, 1) \quad \text{and} \quad (\mu_2, \sigma_2) = (2, 0.5),$$

we define Dirac distributions

$$\delta_{(0,1)} \quad \text{and} \quad \delta_{(2,0.5)}.$$

Right: Dirac locations on parameter space  $\mathcal{T}$ .



## Resulting mixture model

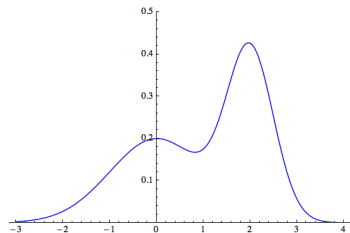
Convolution of

$$q(\mu, \sigma) = c_1 \delta_{(0,1)}(\mu, \sigma) + c_2 \delta_{(2,0.5)}(\mu, \sigma)$$

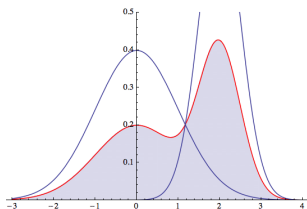
with a Gaussian density  $g(x|\mu, \sigma)$  results in

$$\pi(x) = c_1 g(x|0, 1) + c_2 g(x|2, 0.5).$$

Right:  $\pi(x)$  plotted for  $c_1 = c_2 = \frac{1}{2}$ .



## Mixture of two Gaussians

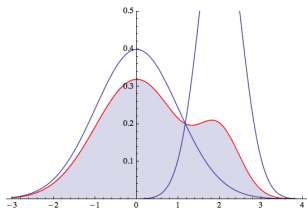


The curve outlined in red is the mixture

$$\pi(x) = 0.5g(x|0, 1) + 0.5g(x|2, 0.5) ,$$

where  $g$  is the Gaussian density. The blue curves are the component densities.

## Influence of the weights



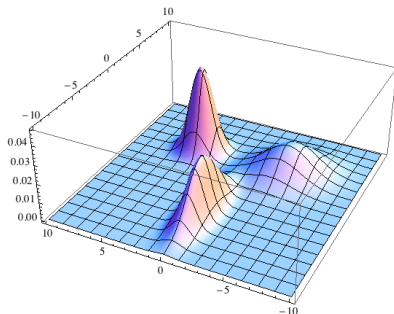
Here, the weights  $c_1 = c_2 = 0.5$  above have been changed to  $c_1 = 0.8$  and  $c_2 = 0.2$ . The component distributions are the same as above.

## Sampling from a finite mixture

For a finite mixture with fixed parameters  $c_k$  and  $\theta_k$ , the two-step sampling procedure is:

1. Choose a mixture component at random. Each component  $k$  is selected with probability  $c_k$ .
2. Sample  $x_i$  from  $p(x|\theta_k)$ .

**Note:** We always repeat both steps, i.e. for  $x_{i+1}$ , we choose again choose a (possibly different) component at random.



## Clustering with finite mixtures

For a clustering problem with  $K$  clusters,

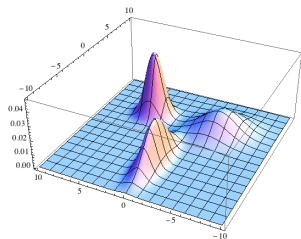
$$p(x|\theta_k) = \text{model of cluster } k$$

The weight  $c_k$  is the relative cluster size.

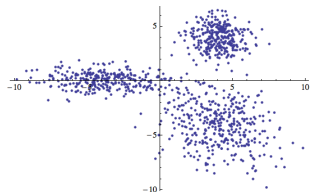
## Estimation problem

If  $K$  is fixed and given, the unknown parameters of a mixture model are the weights  $c_k$  and the cluster parameters  $\theta_k$ . The parameters of finite mixtures are estimated using a method known as the *EM algorithm*.

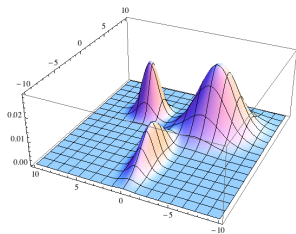
# ILLUSTRATION: MIXTURE OF GAUSSIAN IN 2D



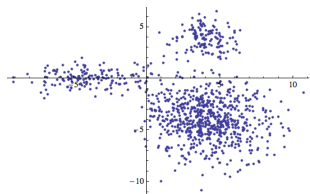
Plot of the mixture density.



A sample of size 1000.



Same components as above, with weight of one component increased.



A sample of 1000 points. Note how the relative size of one cluster has increased.

# MIXTURE ESTIMATION

## Maximum likelihood for finite mixtures

Writing down the maximum likelihood problem is straightforward:

$$(\hat{\mathbf{c}}, \hat{\boldsymbol{\theta}}) := (\hat{c}_1, \dots, \hat{c}_K, \hat{\theta}_1, \dots, \hat{\theta}_K) = \arg \max_{\mathbf{c}, \boldsymbol{\theta}} \prod_{i=1}^n \left( \sum_{k=1}^K c_k p(x_i | \theta_k) \right)$$

The maximality equation for the logarithmic likelihood is

$$\frac{\partial}{\partial(\mathbf{c}, \boldsymbol{\theta})} \sum_{i=1}^n \log \left( \sum_{k=1}^K c_k p(x_i | \theta_k) \right) = 0$$

The component equation for each  $\theta_k$  is:

$$\sum_{i=1}^n \frac{c_k \frac{\partial}{\partial \theta_k} p(x_i | \theta_k)}{\sum_{k=1}^K c_k p(x_i | \theta_k)} = 0$$

Solving this problem is analytically infeasible (note that we cannot multiply out the denominator, because of the sum over  $i$ ). Even numerical solution is often difficult.



## Problems with ML estimation

- ▶ Solving the ML problem is difficult.
- ▶ For clustering, the maximum likelihood solution does not tell us *which* cluster generated each  $x_i$ .

## Cluster assignments

- ▶ The mixture assumption implies that each  $x_i$  was generated from one component.
- ▶ For each  $x_i$ , we again use an **assignment variable**  $m_i \in \{1, \dots, K\}$  which encodes which cluster  $x_i$  was sampled from.

## Latent Variables

Since we do not know which component each  $x_i$  was generated by, the values of the assignment variables are *unobserved*. Such variables whose values are not observed are called **latent variables** or **hidden variables**.

## Latent variables as auxiliary information

If we knew the correct assignments  $m_i$ , we could:

- ▶ Estimate each component distribution  $p(x|\theta_k)$  separately, using only the data assigned to cluster  $k$ .
- ▶ Estimate the cluster proportions  $c_k$  as  $\hat{c}_k := \frac{\text{\#points in cluster } k}{n}$ .

## EM algorithm: Idea

The EM algorithm estimates values of the latent variables to simplify the estimation problem. EM alternates between two steps:

1. Estimate assignments  $m_i$  given current estimates of the parameters  $c_k$  and  $\theta_k$  ("E-step").
2. Estimate parameters  $c_i$  and  $\theta_k$  given current estimates of the assignments ("M-step").

These two steps are iterated repeatedly.

# REPRESENTATION OF ASSIGNMENTS

We re-write the assignments as vectors of length  $K$ :

$$\mathbf{x}_i \text{ in cluster } k \quad \text{as} \quad M_i := \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \leftarrow k\text{th entry}$$

so  $M_{ik} = 1$  if  $x_i$  in cluster  $k$ , and  $M_{ik} = 0$  otherwise.

We collect the vectors into a matrix

$$\mathbf{M} = \begin{pmatrix} M_{11} & \dots & M_{1K} \\ \vdots & & \vdots \\ M_{n1} & \dots & M_{nK} \end{pmatrix}$$

Note: Rows = observations, columns = clusters

Row sums = 1, column sums = cluster sizes.

## Hard vs soft assignments

- ▶ The vectors  $M_i$  are "hard assignments" with values in  $\{0, 1\}$  (as in  $k$ -means).
- ▶ EM computes "soft assignments"  $a_{ik}$  with values in  $[0, 1]$ .
- ▶ Once the algorithm terminates, each point is assigned to a cluster by setting

$$m_i := \arg \max_k a_{ik}$$

The vectors  $M_i$  are the latent variables in the EM algorithm. The  $a_{ik}$  are there current estimates.

## Assignment probabilities

The soft assignments are computed as

$$a_{ik} := \frac{c_k p(x_i | \theta_k)}{\sum_{l=1}^K c_l p(x_i | \theta_l)} .$$

They can be interpreted as

$$a_{ik} := \mathbb{E}[M_{ik} | x_i, \mathbf{c}, \boldsymbol{\theta}] = \Pr\{x_i \text{ generated by component } k \mid \mathbf{c}, \boldsymbol{\theta}\}$$

# M-STEP (1)

## Objective

The M-Step re-estimates  $\mathbf{c}$  and  $\boldsymbol{\theta}$ . In principle, we use maximum likelihood within each cluster, but we have to combine it with the use of weights  $a_{ik}$  instead of "switch variables"  $M_{ik}$ .

## Cluster sizes

If we know which points belong to which cluster, we can estimate the cluster proportions  $c_k$  by counting point:

$$\hat{c}_k = \frac{\# \text{ points in cluster } k}{n} = \frac{\sum_{i=1}^n M_{ik}}{n}$$

Since we do not know  $M_{ik}$ , we substitute our current best guess, which are the expectations  $a_{ik}$ :

$$\hat{c}_k := \frac{\sum_{i=1}^n a_{ik}}{n}$$

# M-STEP (2)

## Gaussian special case

The estimation of the component parameters  $\theta$  depends on which distribution we choose for  $p$ . For now, we assume a Gaussian.

## Component parameters

We use maximum likelihood to estimate  $\theta = (\mu, \Sigma)$ . We can write the MLE of  $\mu_k$  as

$$\hat{\mu}_k := \frac{1}{\# \text{ points in cluster } k} \sum_{i|x_i \text{ in } k} x_i = \frac{\sum_{i=1}^n M_{ik} x_i}{\sum_{i=1}^n M_{ik}}$$

By substituting current best guesses ( $=a_{ik}$ ) again, we get:

$$\hat{\mu}_k := \frac{\sum_{i=1}^n a_{ik} x_i}{\sum_{i=1}^n a_{ik}}$$

For the covariance matrices:

$$\hat{\Sigma}_k := \frac{\sum_{i=1}^n a_{ik} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^t}{\sum_{i=1}^n a_{ik}}$$

# NOTATION SUMMARY

## Assignment probabilities

$$\mathbf{a} = \begin{pmatrix} a_{11} & \dots & a_{1K} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nK} \end{pmatrix} = \mathbb{E} \left[ \begin{pmatrix} M_{11} & \dots & M_{1K} \\ \vdots & & \vdots \\ M_{n1} & \dots & M_{nK} \end{pmatrix} \right] = \begin{pmatrix} \mathbb{E}[M_{11}] & \dots & \mathbb{E}[M_{1K}] \\ \vdots & & \vdots \\ \mathbb{E}[M_{n1}] & \dots & \mathbb{E}[M_{nK}] \end{pmatrix}$$

Rows = observations, columns = clusters.

## Mixture parameters

$$\boldsymbol{\tau} = (\mathbf{c}, \boldsymbol{\theta}) \quad \mathbf{c} = \text{cluster proportions} \quad \boldsymbol{\theta} = \text{component parameters}$$

## Iterations

$\theta^{(j)}$ ,  $\mathbf{a}^{(j)}$  etc = values in  $j$ th iteration

# SUMMARY: EM FOR GAUSSIAN MIXTURE

## Gaussian special case

$\theta = (\mu, \Sigma)$  (mean & covariance)       $p(x|\theta) = g(x|\mu, \Sigma)$  (Gaussian density)

## Algorithm

The EM algorithm for a finite mixture of Gaussians looks like this:

- ▶ **Initialize:** Choose random values  $c_k^{(0)}$  and  $\theta_k^{(0)}$ .
- ▶ **E-Step:** Recompute the assignment weight matrix as

$$a_{ik}^{(j+1)} := \frac{c_k^{(j)} g(x_i|\theta_k^{(j)})}{\sum_{l=1}^K c_l^{(j)} g(x_i|\theta_l^{(j)})}.$$

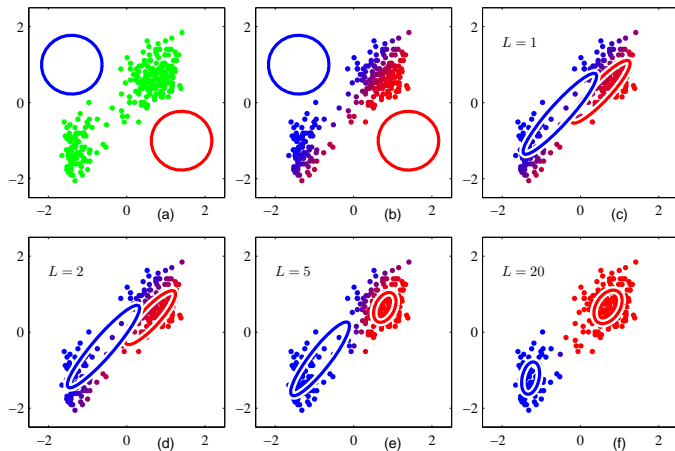
- ▶ **M-Step:** Recompute the proportions  $c_k$  and parameters  $\theta_k = (\mu_k, \Sigma_k)$  as

$$\mu_k^{(j+1)} := \frac{\sum_{i=1}^n a_{ik}^{(j+1)} x_i}{\sum_{i=1}^n a_{ik}^{(j+1)}} \quad \text{and} \quad \Sigma_k^{(j+1)} := \frac{\sum_{i=1}^n a_{ik}^{(j+1)} (x_i - \mu_k^{(j+1)})(x_i - \mu_k^{(j+1)})^t}{\sum_{i=1}^n a_{ik}^{(j+1)}}$$

The E-Step and M-Step are repeated alternately until convergence criterion (e.g. threshold) satisfied.



## EM for a mixture of two Gaussians



The algorithm fits both the mean and the covariance parameter.

## Reminder: Objective

Estimate  $\theta$  and  $c$  by (approximate) Maximum Likelihood for

$$\pi(x) = \sum_{k=1}^K c_k p(x|\theta_k) =: \pi(x|\mathbf{c}, \boldsymbol{\theta}) .$$

The components  $p(x|\theta_k)$  need not be Gaussian.

# STEP 1 (OF 4)

## Including the latent variables

Recall that we can integrate out a variable  $y$  from a joint density  $p(x, y)$ :

$$p(x) = \int p(x, y) dy$$

We can apply this idea backwards and write the likelihood  $\pi(x|c, \theta)$  as

$$\pi(x|c, \theta) = \sum_{\mathbf{M}} \pi(x, \mathbf{M}|c, \theta)$$

Since  $\mathbf{M}$  is discrete, the integral is a sum (over all possible assignment vectors  $\mathbf{M}$ ).

## Application to the log-likelihood

$$\sum_{i=1}^n \log \pi(x_i|c, \theta) = \sum_{i=1}^n \log \left( \sum_{\mathbf{M}_i} \pi(x, \mathbf{M}_i|c, \theta) \right)$$

## STEP 2

### Approximation of the log-likelihood

We replace the log-likelihood

$$\sum_{i=1}^n \log \left( \sum_{\mathbf{M}_i} \pi(x, \mathbf{M}_i | c, \theta) \right) \quad \text{by} \quad \sum_{i=1}^n \sum_{\mathbf{M}_i} \log \pi(x, \mathbf{M}_i | c, \theta)$$

This is an *approximation*, the two terms are not identical.

### Justification

It can be shown that always

$$\sum_{i=1}^n \sum_{\mathbf{M}_i} \log \pi(x, \mathbf{M}_i | c, \theta) \leq \sum_{i=1}^n \log \left( \sum_{\mathbf{M}_i} \pi(x, \mathbf{M}_i | c, \theta) \right) .$$

That means we substitute the log-likelihood by a *lower bound*, and maximize the lower bound.

# STEP 3

## Current form of our problem

We have to solve the problem

$$(c^*, \theta^*) = \arg \max_{c, \theta} \sum_{i=1}^n \sum_{\mathbf{M}_i} \log \pi(x_i, \mathbf{M}_i | c, \theta)$$

but we only know the data  $x_i$ , not the  $\mathbf{M}_i$ .

## Taking expectations

If we knew at least the distribution  $q(\mathbf{M}_i)$  of  $\mathbf{M}_i$ , we could maximize the expected value:

$$(c^*, \theta^*) := \arg \max_{c, \theta} \sum_{i=1}^n \sum_{\mathbf{M}_i} q(\mathbf{M}_i) \log \pi(x_i, \mathbf{M}_i | c, \theta)$$

## STEP 4 (OF 4)

### Making the steps iterative

In step  $(j + 1)$  of the algorithm:

- ▶ We want to compute  $c^{(j+1)}, \theta^{(j+1)}$ .
- ▶ We *know* the previous estimates  $c^{(j)}, \theta^{(j)}$ .

Strategy: For quantities we do not know in *current* step, we substitute estimates based on *previous* step.

### Substituting previous estimates

As distribution  $q(\mathbf{M}_i)$ , we use

$$\Pr\{\mathbf{M}_i = k | c^{(j)}, \theta^{(j)}\} = \Pr\{x_i \text{ generated by component } k \text{ in mixture } \pi(x_i | \mathbf{c}^{(j)}, \boldsymbol{\theta}^{(j)})\}$$

which is precisely

$$\Pr\{\mathbf{M}_i = k | c^{(j)}, \theta^{(j)}\} = a_{ik}^{(j)} .$$

## Algorithm

- ▶ **E-Step:** Recompute assignment matrix  $a_{ik}^{(j)}$  as

$$a_{ik}^{(j+1)} := \frac{c_k^{(j)} p(x_i | \theta_k^{(j)})}{\sum_{l=1}^K c_l^{(j)} p(x_i | \theta_l^{(j)})}.$$

- ▶ **M-Step:** Recompute  $(c, \theta)$  as

$$(c^{(j+1)}, \theta^{(j+1)}) := \arg \max_{c, \theta} \left\{ \sum_{ik} a_{ik}^{(j+1)} \log(c_k p(x_i | \theta_k)) \right\}$$

## Convenient special case

If the MLE of  $p(x|\theta)$  is of the form  $\hat{\theta}_{\text{ML}} = \frac{1}{n} \sum_i f(x_i)$  for some function  $f$ , the M-step computes the "weighted maximum likelihood estimate":

$$c_k^{(j+1)} := \frac{\sum_{i=1}^n a_{ik}^{(j+1)}}{n} \quad \text{and} \quad \theta_k^{(j+1)} := \frac{\sum_{i=1}^n a_{ik}^{(j+1)} f(x_i)}{\sum_{i=1}^n a_{ik}^{(j+1)}}$$

This is, for example, the case for the Gaussian and the multinomial distribution.

## Approximations Used in EM

The derivation makes two approximations:

1. The log-likelihood is substituted by a lower bound.
2. The unknown assignments  $M_i$  are substituted by their expectations  $a_{ik}^{(j)}$  under the current model.



# CONVERGENCE PROPERTIES

## Log-likelihood

- ▶ It can be shown that the likelihood

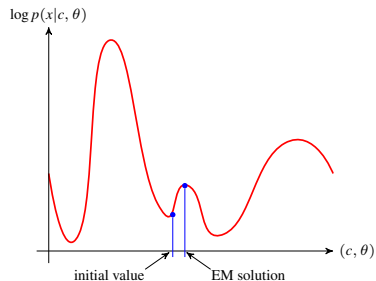
$$\prod_{i=1}^n \pi(x_i | \mathbf{c}, \boldsymbol{\theta})$$

always increases from each step to the next, unless  $(\mathbf{c}, \boldsymbol{\theta})$  is already a stationary point.

- ▶ The theory guarantees only that the algorithm terminates at a stationary point. That point can be a saddle point rather than a maximum (very rare problem).

## The real problem: Local maxima

- ▶ EM is effectively a gradient method.
- ▶ The maxima it finds are **local maxima of the log-likelihood**.
- ▶ There are no guarantees on the global quality of the solution: The global maximum may differ arbitrarily from the one we find.



## Comparing solutions

- ▶ If  $(\mathbf{c}, \boldsymbol{\theta})$  and  $(\mathbf{c}', \boldsymbol{\theta}')$  are two different EM solutions, we can always compute the log-likelihoods

$$\sum_i \log \pi(x_i | \mathbf{c}, \boldsymbol{\theta}) \quad \text{and} \quad \sum_i \log \pi(x_i | \mathbf{c}', \boldsymbol{\theta}')$$

(no approximations or complications!).

- ▶ The solution with the higher likelihood is better.
- ▶ This is a very convenient feature of EM: Different solutions are comparable.

## Random restarts

In practice, the best way to use EM is often:

- ▶ Restart EM repeatedly with randomly chosen initial values.
- ▶ Compute the log-likelihoods of all solutions and compare them.
- ▶ Choose the solution achieving maximal log-likelihood.

# MODEL ORDER SELECTION

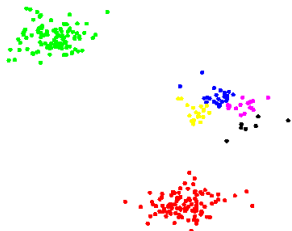
# MODEL SELECTION FOR CLUSTERING

## The model selection problem

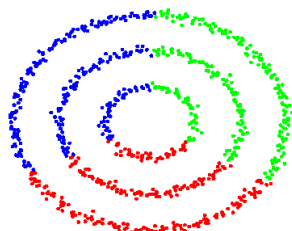
For mixture models  $\pi(x) = \sum_{k=1}^K c_k p(x|\theta_k)$ , we have so far assumed that the number  $K$  of clusters is known.

## Model Order

Methods which automatically determine the complexity of a model are called **model selection** methods. The number of clusters in a mixture model is also called the **order** of the mixture model, and determining it is called **model order selection**.



(a) Inappropriate model order.



(b) Inappropriate model type.

# MODEL SELECTION FOR CLUSTERING

## Notation

We write  $\mathcal{L}$  for the log-likelihood of a parameter under a model  $p(x|\theta)$ :

$$\mathcal{L}(\mathbf{x}^n; \theta) := \log \prod_{i=1}^n p(x_i|\theta)$$

In particular, for a mixture model:

$$\mathcal{L}(\mathbf{x}^n; \mathbf{c}, \boldsymbol{\theta}) := \log \prod_{i=1}^n \left( \sum_{k=1}^K c_k p(x_i|\theta_k) \right)$$

## Number of clusters: Naive solution (wrong!)

We could treat  $K$  as a parameter and use maximum likelihood, i.e. try to solve:

$$(K, c_1, \dots, c_K, \theta_1, \dots, \theta_K) := \arg \max_{K, \mathbf{c}', \boldsymbol{\theta}'} \mathcal{L}(\mathbf{x}^n; K, \mathbf{c}', \boldsymbol{\theta}')$$

## Problem with naive solution: Example

Suppose we use a Gaussian mixture model.

- ▶ The optimization procedure can add additional components arbitrarily.
- ▶ It can achieve minimal fitting error by using a separate mixture component for each data point (ie  $\mu_k = x_i$ ).
- ▶ By reducing the variance of each component, it can additionally increase the density value at  $\mu_k = x_i$ . That means we can achieve arbitrarily high log-likelihood.
- ▶ Note that such a model (with very high, narrow component densities at the data points) would achieve *low* log-likelihood on a new sample from the same source. In other words, it does not generalize well.

In short: The model overfits.

## The general problem

- ▶ Recall our discussion of model complexity: Models with more degrees of freedom are more prone to overfitting.
- ▶ The number of degrees of freedom is roughly the number of scalar parameters.
- ▶ By increasing  $K$ , the clustering model can *add more degrees of freedom*.

## Most common solutions

- ▶ **Penalization approaches:** A penalty term makes adding parameters expensive. Similar to shrinkage in regression.
- ▶ **Stability:** Perturb the distribution using resampling or subsampling. Idea: A choice of  $K$  for which solutions are stable under perturbation is a good explanation of the data.
- ▶ **Bayesian methods:** Each possible value of  $K$  is assigned a probability, which is combined with the likelihood given  $K$  to evaluate the plausibility of the solution. Somewhat related to penalization.

# PENALIZATION STRATEGIES

## General form

Penalization approaches define a *penalty function*  $\phi$ , which is an increasing function of the number  $m$  of model parameters.

Instead of *maximizing* the log-likelihood, we *minimize* the *negative* log-likelihood and add  $\phi$ :

$$(m, \theta_1, \dots, \theta_m) = \arg \min_{m, \theta_1, \dots, \theta_m} -\mathcal{L}(\mathbf{x}^n; \theta_1, \dots, \theta_m) + \phi(m)$$

## The most popular choices

The penalty function

$$\phi_{\text{AIC}}(m) := m$$

is called the **Akaike information criterion (AIC)**.

$$\phi_{\text{BIC}}(m) := \frac{1}{2} m \log n$$

is called the **Bayesian information criterion (BIC)**.



## Clustering with penalization

For clustering, AIC means:

$$(K, \mathbf{c}, \boldsymbol{\theta}) = \arg \min_{K, \mathbf{c}', \boldsymbol{\theta}'} -\mathcal{L}(\mathbf{x}^n; K, \mathbf{c}', \boldsymbol{\theta}') + K$$

Similarly, BIC solves:

$$(K, \mathbf{c}, \boldsymbol{\theta}) = \arg \min_{K, \mathbf{c}', \boldsymbol{\theta}'} -\mathcal{L}(\mathbf{x}^n; K, \mathbf{c}', \boldsymbol{\theta}') + \frac{1}{2}K \log n$$

## Which criterion should we use?

- ▶ BIC penalizes additional parameters more heavily than AIC (ie tends to select fewer components).
- ▶ Various theoretical results provide conditions under which one of the criteria succeeds or fails, depending on:
  - ▶ Whether the sample is small or large.
  - ▶ Whether the individual components are misspecified or not.
- ▶ BIC is more common choice in practice.

## Assumption

A value of  $K$  is plausible if it results in similar solutions on separate samples.

## Strategy

As in cross validation and bootstrap methods, we "simulate" different sample sets by perturbation or random splits of the input data.

## Recall: Assignment in mixtures

Recall that, under a mixture model  $\pi = \sum_{k=1}^K c_k p(x|\theta_k)$ , we compute a "hard" assignment for a data point  $x_i$  as

$$m_i := \arg \max_k c_k p(x_i|\theta_k)$$

## Computing the stability score for fixed $K$

1. Randomly split the data into two sets  $\mathcal{X}'$  and  $\mathcal{X}''$  of equal size.
2. Separately estimate mixture models  $\pi'$  on  $\mathcal{X}'$  and  $\pi''$  on  $\mathcal{X}''$ , using EM.
3. For each data point  $x_i \in \mathcal{X}''$ , compute assignments  $m'_i$  under  $\pi'$  and  $m''_i$  under  $\pi''$ . (That is:  $\pi'$  is now used for prediction on  $\mathcal{X}''$ .)
4. Compute the score

$$\psi(K) := \min_{\sigma} \sum_{i=1}^n \mathbb{I}\{m'_i \neq \sigma(m''_i)\}$$

where the minimum is over all permutations  $\sigma$  which permute  $\{1, \dots, K\}$ .

## Explanation

- ▶  $\psi(K)$  measures: How many points are assigned to a different cluster under  $\pi'$  than under  $\pi''$ ?
- ▶ The minimum over permutations is necessary because the numbering of clusters is not unique. (Cluster 1 in  $\pi'$  might correspond to cluster 5 in  $\pi''$ , etc.)

## Selecting the number of clusters

1. Compute  $\psi(K)$  for a range of values of  $K$ .
2. Select  $K$  for which  $\psi(K)$  is minimal.

## Improving the estimate of $\psi(K)$

For each  $K$ , we can perform multiple random splits and estimate  $\psi(K)$  by averaging over these.

## Performance

- ▶ Empirical studies show good results on a range of problems.
- ▶ Some basic theoretical results available, but not as detailed as for AIC or BIC.

# EXPONENTIAL FAMILY MODELS

# EXPONENTIAL FAMILY DISTRIBUTIONS

## Definition

We consider a model  $\mathcal{P}$  for data in a sample space  $\mathbf{X}$  with parameter space  $\mathcal{T} \subset \mathbb{R}^m$ . Each distribution in  $\mathcal{P}$  has density  $p(x|\theta)$  for some  $\theta \in \mathcal{T}$ .

The model is called an **exponential family model** (EFM) if  $p$  can be written as

$$p(x|\theta) = \frac{h(x)}{Z(\theta)} e^{\langle S(x), \theta \rangle}$$

where:

- ▶  $S$  is a function  $S : \mathbf{X} \rightarrow \mathbb{R}^m$ . This function is called the **sufficient statistic** of  $\mathcal{P}$ .
- ▶  $h$  is a function  $h : \mathbf{X} \rightarrow \mathbb{R}_+$ .
- ▶  $Z$  is a function  $Z : \mathcal{T} \rightarrow \mathbb{R}_+$ , called the **partition function**.

Exponential families are important because:

1. The special form of  $p$  gives them many nice properties.
2. Most important parametric models (e.g. Gaussians) are EFMs.
3. Many algorithms and methods can be formulated generically for all EFMs.

# ALTERNATIVE FORM

The choice of  $p$  looks perhaps less arbitrary if we write

$$p(x|\theta) = \exp\left(\langle S(x), \theta \rangle - \phi(x) - \psi(\theta)\right)$$

which is obtained by defining

$$\phi(x) := -\log(h(x)) \quad \text{and} \quad \psi(\theta) := \log(Z(\theta))$$

## A first interpretation

Exponential family models are models in which:

- ▶ The data and the parameter interact only through the linear term  $\langle S(x), \theta \rangle$  in the exponent.
- ▶ The logarithm of  $p$  can be non-linear in both  $S(x)$  and  $\theta$ , but there is no *joint* nonlinear function of  $(S(x), \theta)$ .

# THE PARTITION FUNCTION

## Normalization constraint

Since  $p$  is a probability density, we know

$$\int_{\mathbf{X}} \frac{h(x)}{Z(\theta)} e^{\langle S(x), \theta \rangle} dx = 1 .$$

## Partition function

The only term we can pull out of the integral is the partition function  $Z(\theta)$ , hence

$$Z(\theta) = \int_{\mathbf{X}} h(x) e^{\langle S(x), \theta \rangle} dx$$

**Note:** This implies that an exponential family is completely determined by choice of the spaces  $\mathbf{X}$  and  $\mathcal{T}$  and of the functions  $S$  and  $h$ .



# EXAMPLE: GAUSSIAN

## In 1 dimension

We can rewrite the exponent of the Gaussian as

$$\begin{aligned}\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2} \frac{(x - \mu)^2}{\sigma^2}\right) &= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2} \frac{x^2}{\sigma^2} + \frac{2x\mu}{2\sigma^2}\right) \exp\left(-\frac{1}{2} \frac{\mu^2}{\sigma^2}\right) \\ &= \underbrace{c(\mu, \sigma)}_{\text{some function of } \mu \text{ and } \sigma} \exp\left(x^2 \cdot \frac{-1}{2\sigma^2} + x \cdot \frac{\mu}{\sigma^2}\right)\end{aligned}$$

This shows the Gaussian is an exponential family, since we can choose:

$$S(x) := (x^2, x) \quad \text{and} \quad \theta := \left(\frac{-1}{2\sigma^2}, \frac{\mu}{\sigma^2}\right) \quad \text{and} \quad h(x) = 1 \quad \text{and} \quad Z(\theta) = c(\mu, \sigma)^{-1}.$$

## In $d$ dimensions

$$S(\mathbf{x}) = (\mathbf{x}\mathbf{x}^t, \mathbf{x}) \quad \text{and} \quad \theta := \left(-\frac{1}{2}\Sigma^{-1}, \Sigma^{-1}\mu\right)$$

# MORE EXAMPLES OF EXPONENTIAL FAMILIES

Model	Sample space	Sufficient statistic
Gaussian	$\mathbb{R}^d$	$S(\mathbf{x}) = (\mathbf{x}\mathbf{x}^t, \mathbf{x})$
Gamma	$\mathbb{R}_+$	$S(x) = (\ln(x), x)$
Poisson	$\mathbb{N}_0$	$S(x) = x$
Multinomial	$\{1, \dots, K\}$	$S(x) = x$
Wishart	Positive definite matrices	(requires more details)
Mallows	Rankings (permutations)	(requires more details)
Beta	$[0, 1]$	$S(x) = (\ln(x), \ln(1 - x))$
Dirichlet	Probability distributions on $d$ events	$S(\mathbf{x}) = (\ln x_1, \dots, \ln x_d)$
Bernoulli	$\{0, 1\}$	$S(x) = x$
...	...	...

## Roughly speaking

On every sample space, there is a "natural" statistic of interest. On a space with Euclidean distance, for example, it is natural to measure both location *and* correlation; on categories (which have no "distance" from each other), it is more natural to measure only expected numbers of counts.

On most types of sample spaces, the exponential family model with  $S$  chosen as this natural statistic is the prototypical distribution.

## Log-likelihood for $n$ samples

$$\log \prod_{i=1}^n p(x_i|\theta) = \sum_{i=1}^n \left( \log(h(x_i)) - \log(Z(\theta)) + \langle S(x_i), \theta \rangle \right)$$

## MLE equation

$$0 = \frac{\partial}{\partial \theta} \sum_{i=1}^n \left( \log(h(x_i)) - \log(Z(\theta)) + \langle S(x_i), \theta \rangle \right) = -n \frac{\partial}{\partial \theta} \log(Z(\theta)) + \sum_{i=1}^n S(x_i)$$

Hence, the MLE is the parameter value  $\hat{\theta}$  which satisfies the equation

$$\frac{\partial}{\partial \theta} \log(Z(\hat{\theta})) = \frac{1}{n} \sum_{i=1}^n S(x_i)$$

# MOMENT MATCHING

## Further simplification

We know that  $Z(\theta) = \int h(x) \exp \langle S(x), \theta \rangle dx$ , so

$$\frac{\partial}{\partial \theta} \log(Z(\theta)) = \frac{\frac{\partial}{\partial \theta} Z(\theta)}{Z(\theta)} = \frac{\int h(x) \frac{\partial}{\partial \theta} e^{\langle S(x), \theta \rangle} dx}{Z(\theta)} = \frac{\int S(x) h(x) e^{\langle S(x), \theta \rangle} dx}{Z(\theta)} = \mathbb{E}_{p(x|\theta)}[S(x)]$$

## MLE equation

Substitution into the MLE equation shows that  $\hat{\theta}$  is given by

$$\mathbb{E}_{p(x|\hat{\theta})}[S(x)] = \frac{1}{n} \sum_{i=1}^n S(x_i)$$

Using the empirical distribution  $\mathbb{F}_n$ , the right-hand side can be expressed as

$$\mathbb{E}_{p(x|\hat{\theta})}[S(x)] = \mathbb{E}_{\mathbb{F}_n}[S(x)]$$

This is called a **moment matching equation**. Hence, MLEs of exponential family models can be obtained by moment matching.

# SUMMARY: MLE FOR EFMS

## The MLE

If  $p(x|\theta)$  is an exponential family model with sufficient statistic  $S$ , the maximum likelihood estimator  $\hat{\theta}$  of  $\theta$  given data  $x_1, \dots, x_n$  is given by the equation

$$\mathbb{E}_{p(x|\hat{\theta})}[S(x)] = \frac{1}{n} \sum_{i=1}^n S(x_i)$$

## Note

We had already noticed that the MLE (for some parameter  $\tau$ ) is often of the form

$$\hat{\tau} = \frac{1}{n} \sum_{i=1}^n f(x_i) .$$

Models are often defined so that the parameters can be interpreted as expectations of some useful statistic (e.g., a mean or variance). If  $\theta$  in an exponential family is chosen as  $\theta = \mathbb{E}_{p(x|\theta)}[S(x)]$ , then we have indeed

$$\hat{\theta} = \frac{1}{n} \sum_{i=1}^n S(x_i) .$$

# EM FOR EXPONENTIAL FAMILY MIXTURE

## Finite mixture model

$$\pi(x) = \sum_{k=1}^K c_k p(x|\theta_k),$$

where  $p$  is an exponential family with sufficient statistic  $S$ .

## EM Algorithm

- ▶ **E-Step:** Recompute the assignment weight matrix as

$$a_{ik}^{(j+1)} := \frac{c_k^{(j)} p(x_i|\theta_k^{(j)})}{\sum_{l=1}^K c_l^{(j)} p(x_i|\theta_l^{(j)})}.$$

- ▶ **M-Step:** Recompute the proportions  $c_k$  and parameters  $\theta_k$  by solving

$$c_k^{(j+1)} := \frac{\sum_{i=1}^n a_{ik}^{(j+1)}}{n} \quad \text{and} \quad \mathbb{E}_{p(x|\theta_k^{(j+1)})}[S(x)] = \frac{\sum_{i=1}^n a_{ik}^{(j+1)} S(x_i)}{\sum_{i=1}^n a_{ik}^{(j+1)}}$$

# EM FOR EXPONENTIAL FAMILY MIXTURE

If in particular the model is parameterized such that

$$\mathbb{E}_{p(x|\theta)}[S(x)] = \theta$$

the algorithm becomes very simple:

- ▶ **E-Step:** Recompute the assignment weight matrix as

$$a_{ik}^{(j+1)} := \frac{c_k^{(j)} p(x_i | \theta_k^{(j)})}{\sum_{l=1}^K c_l^{(j)} p(x_i | \theta_l^{(j)})} .$$

- ▶ **M-Step:** Recompute the proportions  $c_k$  and parameters  $\theta_k$  as

$$c_k^{(j+1)} := \frac{\sum_{i=1}^n a_{ik}^{(j+1)}}{n} \quad \text{and} \quad \theta_k^{(j+1)} := \frac{\sum_{i=1}^n a_{ik}^{(j+1)} S(x_i)}{\sum_{i=1}^n a_{ik}^{(j+1)}}$$

# THE MULTINOMIAL DISTRIBUTION



## Categorical random variable

We call a random variable  $\xi$  **categorical** if it takes values in a finite set, i.e. if  $\xi \in \{1, \dots, d\}$  for some  $d \in \mathbb{N}$ . We interpret the  $d$  different outcomes as  $d$  separate *categories* or classes.

## Category probabilities

Suppose we know the probability  $t_j = \Pr\{\xi = j\}$  for each category  $j$ . Then

$$t_j \geq 0 \quad \text{and} \quad \sum_{j=1}^d t_j = 1$$

We can represent the distribution of  $\xi$  by the vector  $\mathbf{t} = (t_1, \dots, t_d) \in \mathbb{R}^d$ . In other words, we can parameterize distributions of categorical variables by vectors  $\mathbf{t}$ .

# SAMPLES OF SIZE $n$

## A single sample

We can represent a single sample as a vector, e.g.

$$(0, 1, 0, 0, 0) \quad \text{if } d = 5 \quad \text{and} \quad \xi = 2 .$$

(Recall the assignments in EM.)

## $n$ samples

A sample of size  $n$  is a vector of counts, e.g.

$$(2, 5, 1, 3, 0)$$

We denote the counts by  $H_j$  and write

$$\mathbf{H} := (H_1, \dots, H_d) \quad \text{with} \quad \sum_{j=1}^d H_j = n .$$

# MULTINOMIAL DISTRIBUTION

## Modeling assumption

The  $n$  observations of  $\xi$  are independent, and the probability for  $\xi = j$  in each draw is  $t_j$ . What is the probability of observing the sample  $H = (H_1, \dots, H_d)$ ?

## Multinomial distribution

Answer: The probability is

$$P(\mathbf{H}|\mathbf{t}) = \frac{n!}{H_1! \dots H_d!} \prod_{j=1}^d t_j^{H_j} = \frac{n!}{H_1! \dots H_d!} \exp\left(\sum_{j=1}^d H_j \log(t_j)\right)$$

Recall:  $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$

**Note:** The assignment variables  $M_i$  in a finite mixture model are multinomially distributed with  $n = 1$  and  $\theta = (c_1, \dots, c_k)$ .

## As an exponential family

The form of  $P$  above shows that the multinomial is an EFM with

$$S(\mathbf{H}) := \mathbf{H} \quad h(\mathbf{H}) := \frac{n!}{H_1! \dots H_d!} \quad \theta_j := \log t_j \quad Z(\theta) := 1 .$$

# EXPLANATION

- ▶ In one draw, the probability of observing  $\xi = j$  is  $t_j$ .
- ▶ In  $n$  draws, the probability of  $n$  times observing  $\xi = j$  is  $t_j^n$ .

Suppose we have  $n = 3$  observation in two categories. How many ways are there to observe exactly two observations in category 1? Three:

	[1, 2] [3]	[1, 3] [2]	[2, 3] [1]
Probability:	$t_1^2 \cdot t_2$	also $t_1^2 \cdot t_2$	again $t_1^2 \cdot t_2$

The total probability of  $H_1 = 2$  and  $H_2 = 1$  is  $3 \cdot t_1^2 \cdot t_2$ .

- ▶ The number of ways that  $n$  elements can be subdivided into  $d$  classes with,  $H_j$  elements falling into class  $j$ , is precisely

$$\frac{n!}{H_1! \cdots H_d!}$$

In the multinomial formula:

$$P(\mathbf{H}|\mathbf{t}) = \underbrace{\frac{n!}{H_1! \cdots H_d!}}_{\text{\# combinations}} \underbrace{\prod_{j=1}^d t_j^{H_j}}_{\text{probability of one combination}}$$

## MLE

The maximum likelihood estimator of  $\mathbf{t}$  is

$$\hat{\mathbf{t}} = (\hat{t}_1, \dots, \hat{t}_d) := \frac{1}{n} (H_1, \dots, H_d) .$$

# MULTINOMIAL PARAMETERS AND SIMPLICES

## The simplex

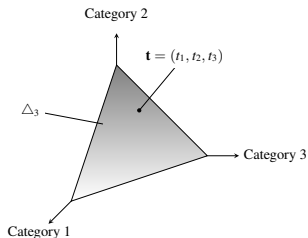
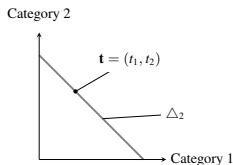
The set of possible parameters of a multinomial distribution is

$$\Delta_d := \{ \mathbf{t} \in \mathbb{R}^d \mid t_j \geq 0 \text{ and } \sum_j t_j = 1 \}$$

$\Delta_d$  is a subset of  $\mathbb{R}^d$  and is called the  $d$ -**simplex**, or the **standard simplex in  $\mathbb{R}^d$** .

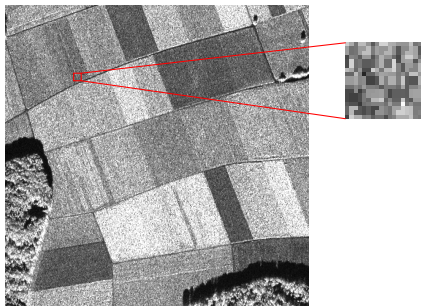
## Interpretation

- ▶ Each point in e.g.  $\Delta_3$  is a distribution on 3 events.
- ▶ Each extreme point (corner) correspond to one category  $j$  and is the distribution with  $t_j = 1$ .
- ▶ The edges of  $\Delta_3$  are the distributions under which only 2 events can occur. (The category corresponding to the opposite corner has zero probability.)
- ▶ The inner points are distributions under which all categories can occur.



# EXAMPLE 1: LOCAL IMAGE HISTOGRAMS

## Extracting local image statistics



1. Place a small window (size  $l \times l$ ) around location in image.
2. Extract the pixel values inside the image. If the grayscale values are e.g.  $\{0, \dots, 255\}$ , we obtain a histogram with 256 categories.
3. Decrease resolution by binning; in Homework 4, we decrease from 256 to 16 categories.

## Resulting data

$$\mathbf{H} = (H_1, \dots, H_{16}) \quad \text{where} \quad H_j = \# \text{ pixel values in bin } j .$$

Since  $256/16 = 16$ , bin  $j$  represents the event

$$\text{pixel value} \in \{(j-1) \cdot 16, \dots, j \cdot 16 - 1\} .$$

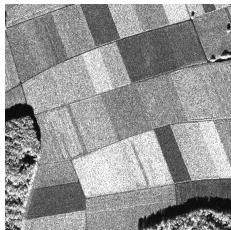
# EXAMPLE 1: LOCAL IMAGE HISTOGRAMS

## Multinomial model

We can model the data by a multinomial distribution  $P(\mathbf{H}|\mathbf{t}, n = l^2)$ . Then

$$t_j = \Pr\{\xi = j\} = \Pr\{\text{grayscale value falls in bin } j\} .$$

## Homework: Multinomial clustering



- ▶ The probability of e.g. bin 1 (dark pixels) clearly varies between locations in the image.
- ▶ Consequence: A single multinomial distribution is not a good representation of this image.
- ▶ In HW 4, the image is represented by a mixture of multinomials which is estimated using EM.



# MULTINOMIAL CLUSTERING AND TEXT MODELS

## Setting

Data set: A huge set of text documents (e.g. all books in a library). The entire set of texts is called a **corpus**.

Can we learn models from text which describe natural language?

## Terminology

We have to distinguish occurrences of words in a document and *distinct* words in the dictionary. We refer to words regarded as entries of the dictionary as **terms**.

# EXAMPLE 2: SIMPLE TEXT MODEL

## Data

Suppose our data is a text document. We are given a dictionary which contains all terms occurring in the document.

## Documents as vectors of counts

We represent the document as

$$\mathbf{H} = (H_1, \dots, H_d) \quad \text{where } H_j = \# \text{ occurrences of term } j \text{ in document.}$$

Note:

- ▶  $d$  is the number of all terms (distinct words) in the dictionary i.e.  $d$  is identical for all documents.
- ▶  $n = \sum_j H_j$  can change from document to document.

# EXAMPLE 2: SIMPLE TEXT MODEL

## Multinomial model

To define a simple probabilistic model of document generation, we can use a multinomial distribution  $P(\mathbf{H}|\mathbf{t}, n)$ . That means:

- ▶ Each word in the document is sampled independently of the other words.
- ▶ The probabilities of occurrence are

$$\Pr\{\text{word} = \text{term } j\} = t_j .$$

## Implicit assumption

The assumption implicit in this model is that the probability of observing a document is completely determined by how often each term occurs; the order of words does not matter. This is called the **bag-of-words assumption**.

## Task

Can we predict the next word in a text?

## Context

In language, the co-occurrence and order of words is highly informative. This information is called the **context** of a word.

**Example:** The English language has over 200,000 words.

- ▶ If we choose any word at random, there are over 200,000 possibilities.
- ▶ If we want to choose the next word in

There is an airplane in the \_\_

the number of possibilities is *much* smaller.

## Significance for statistical methods

Context information is well-suited for machine learning: By parsing lots of text, we can record which words occur together and which do not.

The standard models based on this idea are called *n-gram models*.

## Bigram model

A bigram model represents the conditional distribution

$$\Pr(\text{word}|\text{previous word}) =: \Pr(w_l|w_{l-1}) ,$$

where  $w_l$  is the  $l$ th word in a text.

## Representation by multinomial distributions

A bigram model is a *family* of  $d$  multinomial distributions, one for each possible previous word.

## Estimation

For each term  $k$ , find all terms in the corpus which are preceded by  $k$  and record their number of occurrences in a vector

$$\mathbf{H}_k = (H_{k1}, \dots, H_{kd}) \quad \text{where } H_{kj} = \text{number of times term } j \text{ follows on term } k$$

Then compute the maximum likelihood estimate  $\hat{\mathbf{t}}_k$  from the sample  $\mathbf{H}_k$ .

**Note:** Both  $j$  and  $k$  run through  $\{1, \dots, d\}$ .

## Multinomial representation of bigram

The distributions in the bigram model are:

$$\Pr(\text{word} = j | \text{previous word} = k) = P(H_j = 1 | \hat{\mathbf{t}}_k, n = 1)$$

where  $P$  is the multinomial distribution. The entire bigram model is the set

$$\{P(\cdot | \hat{\mathbf{t}}_k, n = 1) | k = 1, \dots, d\}$$

## N-gram models

More generally, a model conditional on the  $(N - 1)$  previous words

$$\Pr(w_l | w_{l-1}, \dots, w_{l-(N-1)})$$

is called an **N-gram model** (with the predicted word, there are  $N$  words in total).

## Unigrams

The special case  $N = 1$  (no context information) is the simple multinomial word probability model which we discussed first. This model is also called a **unigram model**.

# LEARNING SHAKESPEARE (1)

## Unigram Model

To him swallowed confess hear both.  
Which. Of save on trail for are ay device  
and rote life have

Every enter now severally so, let

Hill he late speaks; or! a more to leg less  
first you enter

Are where exeunt and sighs have rise  
excellency took of.. Sleep knave we. near;  
vile like

## Bigram Model

What means, sir. I confess she? then all  
sorts, he is trim, captain.

Why dost stand forth thy canopy, forsooth;  
he is this palpable hit the King Henry.  
Live king. Follow.

What we, hath got so she that I rest and  
sent to scold and nature bankrupt, nor the  
first gentleman?

Enter Menenius, if it so many good  
direction found'st thou art a strong upon  
command of fear not a liberal largess  
given away, Falstaff! Exeunt



# LEARNING SHAKESPEARE (2)

## Trigram Model

Sweet prince, Falstaff shall die. Harry of Monmouth's grave.

This shall forbid it should be branded, if renown made it empty.

Indeed the duke; and had a very good friend.

Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.

## Quadrigram Model

King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;

Will you not tell me who I am?

It cannot be but so.

Indeed the short and the long. Marry, 'tis a noble Lepidus.

# COMPLEXITY OF $N$ -GRAM MODELS

## Enumerating contexts

An  $N$ -gram model considers ordered combinations of  $N$  terms (=distinct words). Say a corpus contains 100,000 words. Then there are

$$100000^N = 10^{5N}$$

possible combinations.

## Naive estimate

If we require on average  $n$  observations per combination to get a reliable estimate, we would need a corpus containing  $n \cdot 10^{5N}$  words.

## Consequence

In practice, you typically encounter bigrams or trigrams. Research labs at some internet companies have reported results for higher orders.

## Task

Suppose we have a corpus consisting of two types of text, (1) cheap romantic novels and (2) books on theoretical physics. Can a clustering algorithm with two clusters automatically sort the books according to the two types?

(We will see that there is more to this than solving artificial sorting problems.)

## Clustering model

We assume the corpus is generated by a multinomial mixture model of the form

$$\pi(\mathbf{H}) = \sum_{k=1}^K c_k P(\mathbf{H}|\mathbf{t}_k) ,$$

i.e. each component  $P(\mathbf{H}|\mathbf{t}_k)$  is multinomial.

**However:** We are now considering **documents** rather than individual words.

## Estimation

Apply EM algorithm for multinomial mixture models.

## Thought experiment

Say we run a mixture of two multinomial distributions on the cheap romantic novels and theoretical physics textbooks.

Outcome:

- ▶ Each cluster will roughly represent one of the two topics.
- ▶ The two parameter vectors  $\mathbf{t}_1$  and  $\mathbf{t}_2$  represent distributions of words in *texts of the respective topic*.

## Word distributions as topics

This motivates the interpretation of clusters as topics.

$\mathbf{t}_k =$  distribution of words that characterizes topic  $k$

Language models derived from this idea are called **topic models**.

# TOOLS: INFORMATION THEORY

## Information content of a random variable

We consider a random variable  $X$  with distribution  $P$ .

- ▶  $P$  expresses what we know *before* we observe  $X$ .
- ▶ How much information do we *gain* by observing  $X$ ?

That is: By information content of  $X$ , we mean the difference in information between knowing  $P$  and knowing both  $P$  and  $X = x$ .

## To reiterate

For the definition of information, it is useful to think of...

- ▶ ...the distribution  $P$  as what we *expect to happen*.
- ▶ ...the sample outcome  $X = x$  as what *actually happens*.

## Heuristic motivation

Suppose we sample  $X = x$  from a distribution  $P$ .

- ▶ If  $P(x)$  is large: Small surprise; we have not gained much additional information.
- ▶ If  $P(x)$  is small: We have gained more information.

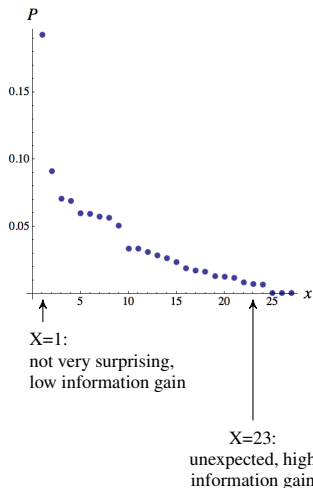
## Conclusions

- ▶ The information in  $X = x$  increases with  $\frac{1}{P(x)}$ .
- ▶ Intuitively, the information gain in two unrelated observations should be additive, so  $\frac{1}{P(x)}$  itself is not a useful measure of information.

## Definition

The **information** in observing  $X = x$  under  $P$  is

$$J_P(x) := \log \frac{1}{P(x)} = -\log P(x) .$$



# SHANNON'S ENTROPY

## Discrete random variables

In information theory, we have to distinguish between discrete and continuous random variables. If  $X$  is a RV with values in a space  $\mathbf{X}$ , we call  $X$  **discrete** if  $\mathbf{X}$  has a finite or at most countably infinite number of elements.

## Definition

Let  $X$  be a discrete random variable with distribution  $P$ . The *expected* information in a draw from  $P$ ,

$$\mathbb{H}[X] := \mathbb{E}_P[J_P(X)]$$

is called the **Shannon entropy** of  $X$ , or the **entropy** for short.

## Remarks

- ▶ Note that

$$\mathbb{E}[J_P(X)] = -\mathbb{E}_P[\log P(X)] = -\sum_{x \in \mathbf{X}} P(x) \log P(x)$$

- ▶ The entropy measures the information gained when sampling from  $P$ .
- ▶ We can interchangeably regard  $\mathbb{H}$  as a property of  $X$  or of  $P$ , and we equivalently write  $\mathbb{H}(P)$  for  $\mathbb{H}[X]$ .



# BASIC PROPERTIES

1. The entropy is non-negative:

$$\mathbb{H}[X] \geq 0$$

2.  $\mathbb{H}(P) = 0$  means there is no uncertainty in  $P$ :

$$\mathbb{H}(P) = 0 \quad \Leftrightarrow \quad P(x_0) = 1 \text{ for some } x_0 \in \mathbf{X} .$$

3. If  $\mathbf{X}$  is finite with  $d$  elements, the distribution with the largest entropy is the uniform distribution  $U_d$ , with

$$\mathbb{H}(U_d) = \log d$$

## Axiomatic description

Suppose we define *some* measure  $\mathcal{H}[X]$  of information in  $X$ . Regardless of the definition, we can postulate a number of properties (axioms) that a meaningful measure should satisfy.

## Additivity

- ▶ If two RVs  $X$  and  $Y$  are independent, their information content should be disjoint.
- ▶ Hence,  $\mathcal{H}$  should be additive:

$$X \perp\!\!\!\perp Y \quad \Rightarrow \quad \mathcal{H}[X, Y] = \mathcal{H}[X] + \mathcal{H}[Y]$$

- ▶ More generally: We should be able to "remove the joint information" in  $X$  and  $Y$  from  $Y$  by conditioning.
- ▶ This is what we require as our first axiom:

$$\text{(Axiom I)} \quad \mathcal{H}[X, Y] = \mathcal{H}[X] + \mathcal{H}[Y|X]$$

## Continuity

- ▶ We can alternatively regard  $\mathcal{H}[X]$  as a function  $\mathcal{H}(P)$  of the distribution of  $X$ .
- ▶ If we make a small change to  $P$ , then  $\mathcal{H}(P)$  should not "jump". That is:

(**Axiom II**)             $\mathcal{H}(P)$  should be continuous as a function of  $P$ .

## Monotonicity

- ▶ Suppose we consider in particular the uniform distribution  $P = U_d$  on  $d$  outcomes.
- ▶ If we increase  $d$ , the uncertainty in  $U_d$  increases; hence, the information gained by sampling should be higher for  $d + 1$  than for  $d$ :

(**Axiom III**)             $\mathcal{H}(U_d) < \mathcal{H}(U_{d+1})$

## Theorem

If a real-valued function  $\mathcal{H}$  on  $\mathbf{X}$  satisfies Axioms I–III, then

$$\mathcal{H}(P) = c \cdot \mathbb{H}(P) \quad \text{for all } P,$$

for some constant  $c \in \mathbb{R}_+$ . (The constant is the same for all  $P$ .)

## In other words

If any information measure satisfies our requirements, it is precisely the entropy, up to a choice of scale.

## How meaningful are the axioms?

- ▶ Over the years, about a dozen different axioms for information measures have been proposed.
- ▶ It can be shown that basically any meaningful combination of two or three of these axioms leads to the same result (i.e. determines the entropy up to scaling).

One might argue that this makes the entropy a much more fundamental quantity than most quantities used in statistics (variance etc).

## Historical note

- ▶ The notion of entropy was first conceived in physics. The first precise definition was given by Boltzmann in the 1870s.
- ▶ The information-theoretic entropy was introduced in the paper

Claude Shannon: "A mathematical theory of communication", 1948.

This paper introduced most of the quantities we discuss here, created the field of information theory, and proved almost all of its fundamental results.

# EXAMPLE: CODING

Suppose we would like to compress a text document (lossless compression).

## Huffman Coding

Here is a simple but efficient coding scheme:

1. Given a text, determine the frequency with which each word occurs.
2. Assign short code words to words that occur often, long code words to words that are rare.

This idea (with a specific algorithm for finding determining the code words) is called **Huffman coding**. If all we are allowed to do is to replace text words by code words, this compression method is optimal.

## Information-theoretic problems

Suppose we know the distribution  $P$  of words in texts. Then we can ask:

1. What is the *expected* compression rate for a random document?
2. Does our encoder achieve the optimal expected rate for  $P$ ?

## The Source Coding Theorem (Shannon)

Suppose we are given a distribution  $P$  on words or symbols and sample a string  $X^n = (X_1, \dots, X_n)$  iid from  $P$ . Then for every  $\varepsilon > 0$ , there is a lossless encoder for which

$$H(P) \leq \mathbb{E} \left[ \frac{1}{n} \cdot \text{length}(\text{encoding}(X^n)) \right] < H(P) + \varepsilon$$

for sufficiently large  $n$ .

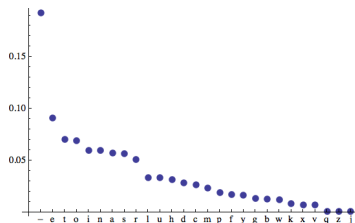
## Remarks

- ▶ In other words: We can encode the sequence  $X^n$  without loss of information using  $nH(P)$  bits on average.
- ▶ The entropy  $H(P)$  is a lower bound for lossless compression: If an encoder achieves a better (=smaller) expectation than above, the probability that it will result in information loss approaches 1 for  $n \rightarrow \infty$ .

# HOW WELL CAN WE COMPRESS ENGLISH TEXT?

## Character-by-character compression

- ▶ We can compress text by splitting the text into characters and assigning a code to each character.
- ▶ An empirical estimate of the distribution of characters is shown on the right. The entropy is 4.11 bit/character.
- ▶ This compression is not very effective: There are 27 characters and  $2^4 < 27 \leq 2^5$ , hence we can trivially encode with 5 bits/character.

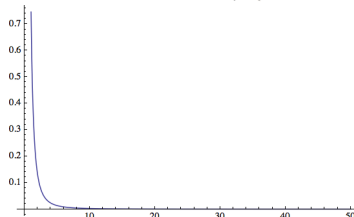
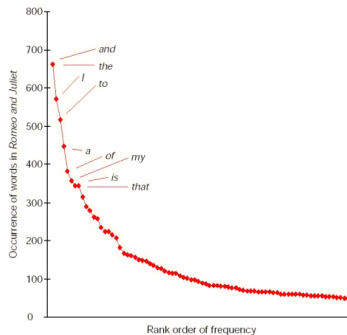




# HOW WELL CAN WE COMPRESS ENGLISH TEXT?

## Word-by-word compression

- ▶ The distribution of words in languages is highly concentrated on a few common words. (Upper plot: Ranked word occurrences in *Romeo and Juliet*.)
- ▶ If we rank words in English by frequency of occurrence, the occurrence distribution is well-approximated by a Zipf distribution with parameter between 1.5 and 2 (lower plot).
- ▶ Due to concentration, these distributions have relatively low entropy.
- ▶ Consequence: If we split into words instead or characters, we can achieve much better compression rates.
- ▶ Common compression algorithms (e.g. Lempel-Ziv) split into substrings which are not necessarily words.



# KULLBACK-LEIBLER DIVERGENCE

## Comparing distributions

We can use the notion of information to compare one distribution to another.

## Heuristic motivation

Suppose we wish to compare two distributions  $P$  and  $Q$  on  $\mathbf{X}$ .

- ▶ The entropy  $\mathbb{H}[Q] = \mathbb{E}_Q[J_Q(X)]$  measures how much information gain (in terms of  $Q$ ) we can *expect* from a random sample from  $Q$ .
- ▶ Now ask instead: How much information gain in terms of  $Q$  can we expect from a random sample drawn from  $P$ ? We compute:  $\mathbb{E}_P[J_Q(X)]$ .
- ▶ A measure of difference between  $P$  and  $Q$  should vanish if  $Q = P$ . Since  $P = Q$  means  $\mathbb{E}_P[J_Q(X)] = \mathbb{H}(P)$ , which is usually not 0, we have to normalize by subtracting  $\mathbb{H}(P)$ .

## Definition

The function

$$D_{\text{KL}}(P||Q) := \mathbb{E}_P[J_Q(X)] - \mathbb{H}(P)$$

is called the **Kullback-Leibler divergence** or the **relative entropy** of  $P$  and  $Q$ .

## Equivalent forms

$$D_{\text{KL}}[P\|Q] = \mathbb{E}_P[J_Q(X) - J_P(X)] = \sum_{x \in \mathbf{X}} P(x) \log \frac{P(x)}{Q(x)}$$

## Positive definiteness

$$D_{\text{KL}}[P\|Q] \geq 0 \quad \text{and} \quad D_{\text{KL}}[P\|Q] = 0 \Leftrightarrow P = Q.$$

## The KL divergence is not a metric

Intuitively,  $D_{\text{KL}}$  can be used like a distance measure between distributions, however:

- ▶ It is *not* symmetric:  $D_{\text{KL}}[P\|Q] \neq D_{\text{KL}}[Q\|P]$  in general.
- ▶ It does *not* satisfy a triangle inequality.

## Convexity

A very useful property of  $\mathbb{H}$  and  $D_{\text{KL}}$  is convexity:

- ▶  $\mathbb{H}(P)$  is concave as a function of  $P$ .
- ▶  $D_{\text{KL}}[P\|Q]$  is convex in the pair  $(P, Q)$ .

- ▶ How can we compute the entropy of  $Y$  conditional on  $X$ ?
- ▶ For a fixed value  $X = x$ , we can simply compute  $\mathbb{H}$  from the conditional probability  $P(Y|X = x)$  as

$$\mathbb{H}[Y|X = x] = - \sum_{y \in \mathbf{X}} P(y|x) \log P(y|x) .$$

- ▶ To make the definition independent of  $x$ , we take the expectation

$$\mathbb{H}[Y|X] := \mathbb{E}_{P(x)}[\mathbb{H}[Y|X = x]] .$$

This is called the **conditional entropy** of  $Y$  given  $X$ .

- ▶ A few lines of arithmetic show:

$$\mathbb{H}[Y|X] = - \sum_{x,y \in \mathbf{X}} P(x,y) \log P(y|x)$$

## Heuristic Motivation

- ▶ Another question we can ask about a pair  $X, Y$  of random variables is: How much information do they share?
- ▶ In other words: How much does observing  $X$  tell us about  $Y$ ?
- ▶ If  $X$  and  $Y$  contain no shared information, they are independent, and their joint distribution is  $P(x, y) = P(x)P(y)$ .
- ▶ Idea: Compare the actual joint distribution to the independent case using KL divergence.

We first define the mutual information in a different way, but will then see that the idea above indeed applies.

## Definition

The function

$$I[X, Y] := \mathbb{H}[X] - \mathbb{H}[X|Y] = \mathbb{H}[Y] - \mathbb{H}[Y|X]$$

is called the **mutual information** of  $X$  and  $Y$ .

# USEFUL RELATIONSHIPS

## Conditioning reduces entropy

$$\mathbb{H}[X, Y] = \mathbb{H}[Y|X] + \mathbb{H}[X]$$

## Mutual information as a Kullback-Leibler divergence

$$I[X, Y] = D_{\text{KL}}[P(x, y) \| P(x)P(y)] = \sum_{x, y} P(x, y) \log \frac{P(x, y)}{P(x)P(y)}$$

Note: This compares  $P(x, y)$  to the case where  $X, Y$  are independent (which means  $P(x, y) = P(x)P(y)$ ).

## Mutual information characterizes independence

$$I[X, Y] = 0 \quad \Leftrightarrow \quad X \perp\!\!\!\perp Y$$

# THE CONTINUOUS CASE

If the sample space  $\mathbf{X}$  is uncountable (e.g.  $\mathbf{X} = \mathbb{R}$ ), instead of  $P$  and  $Q$  we consider densities  $p$  and  $q$ , we have to substitute integrals for sums.

## Differential entropy

$$\mathbb{H}[X] := - \int_{\mathbf{X}} p(x) \log p(x) dx$$

Since  $p$  is a density, we can have  $\log p(x) > 0$ , and  $\mathbb{H}[X]$  can be negative. To distinguish it from the entropy,  $\mathbb{H}[X]$  is called the **differential entropy**.

## KL divergence and mutual information

$D_{\text{KL}}$  and  $I$  are defined analogously to the discrete case:

$$D_{\text{KL}}(p||q) := \int_{\mathbf{X}} p(x) \log \frac{p(x)}{q(x)} dx$$
$$I[X, Y] := \int_{\mathbf{X}} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} dx$$

## Differential entropy

- ▶ Since  $p$  is a density, we can have  $\log p(x) > 0$ , and  $\mathbb{H}[X]$  can be negative.
- ▶ The term differential entropy is used to distinguish it from the entropy.

## KL divergence

The KL divergence for densities still satisfies

$$D_{\text{KL}}(p||q) \geq 0 \quad \text{and} \quad D_{\text{KL}}(p||q) = 0 \Leftrightarrow p = q .$$

As a consequence, the mutual information still satisfies

$$I[X, Y] \geq 0 \quad \text{and} \quad I[X, Y] = 0 \Leftrightarrow X \perp\!\!\!\perp Y .$$



# KL DIVERGENCE AND MAXIMUM LIKELIHOOD

## Idea

Suppose we observe data  $x_1, \dots, x_n$  and assume a model  $\mathcal{P} = \{p(x|\theta) | \theta \in \mathcal{T}\}$ . We could fit the model using the KL divergence as a cost measure:

$$\hat{\theta} := \arg \min_{\theta \in \mathcal{T}} D_{\text{KL}}(\mathbb{F}_n | p(x|\theta))$$

## Computation

$$\begin{aligned} \hat{\theta} &= \arg \min_{\theta \in \mathcal{T}} D_{\text{KL}}(\mathbb{F}_n | p(x|\theta)) = \arg \min_{\theta \in \mathcal{T}} \left( \int_{\mathbf{X}} \mathbb{F}_n(x) \log \frac{\mathbb{F}_n(x)}{p(x|\theta)} dx \right) \\ &= \arg \min_{\theta \in \mathcal{T}} \left( \int_{\mathbf{X}} \mathbb{F}_n(x) \log \mathbb{F}_n(x) dx - \int_{\mathbf{X}} \mathbb{F}_n(x) \log p(x|\theta) dx \right) \\ &= \arg \max_{\theta \in \mathcal{T}} \left( \int_{\mathbf{X}} \mathbb{F}_n(x) \log p(x|\theta) dx \right) = \arg \max_{\theta \in \mathcal{T}} \left( \int_{\mathbf{X}} \frac{1}{n} \sum_{i=1}^n \delta_{x_i}(x) \log p(x|\theta) dx \right) \\ &= \arg \max_{\theta \in \mathcal{T}} \left( \frac{1}{n} \sum_{i=1}^n \log p(x_i|\theta) \right) = \hat{\theta}_{\text{MLE}} \end{aligned}$$

Minimizing KL divergence between  $\mathbb{F}_n$  and the model is equivalent to maximum likelihood estimation!

## The maximum entropy principle

Suppose we have to choose a model distribution from a given set  $\mathcal{P}$  of admissible distributions. The **maximum entropy principle** says: Always choose the distribution

$$P = \arg \max_{Q \in \mathcal{P}} \mathbb{H}(Q)$$

with the highest entropy in  $\mathcal{P}$ .  $P$  is called the **maximum entropy distribution**, which is sometimes abbreviated to ‘MaxEnt distribution’.

## Rationale

- ▶ When choosing a model distribution, we should try to avoid illicit assumptions.
- ▶ Higher entropy  $\leftrightarrow$  higher uncertainty  $\leftrightarrow$  fewer assumptions.

This idea was introduced by the physicist E. T. Jaynes, who championed it as a general modeling approach.

# MAXIMUM ENTROPY UNDER CONSTRAINTS

## Maximum entropy under constraints

Suppose the set  $\mathcal{P}$  of distributions is defined by a constraint. For example:

$$\mathcal{P} = \text{all distributions on } \mathbb{R} \text{ with variance } \sigma_0^2 .$$

### Example 1: Trivial constraint

Suppose the only constraint is that the choice of sample space, e.g.  $\mathbf{X} = [0, 1]$ . Then the maximum entropy distribution is the uniform distribution on  $[0, 1]$ .

### Example 2: Given variance

If  $\mathcal{P} = \{ \text{distributions on } \mathbb{R} \text{ with } \text{Var}[X] = \sigma_0^2 \}$ , then  $P$  is Gaussian with variance  $\sigma_0^2$ .

# THE EXPONENTIAL FAMILY AGAIN

## Expectations as constraints

Suppose  $\mathbf{X} = \mathbb{R}^d$ , and we formulate constraints by choosing functions  $S_1, \dots, S_m : \mathbf{X} \rightarrow \mathbb{R}$  and positing their expected values.

That is, the constrained set is

$$\mathcal{P} := \{Q \mid \mathbb{E}_Q[S_1(X)] = s_1, \dots, \mathbb{E}_Q[S_m(X)] = s_m\}.$$

## Constrained optimization problem (for the discrete case)

We add the constraints to the objective function  $\mathbb{H}(Q)$  using Lagrange multipliers  $\theta_1, \dots, \theta_n$ . We also include a normalization constraint with Lagrange multiplier  $\theta_0$ .

$$P = \arg \max_Q \mathbb{H}(Q) + \theta_0 \left(1 - \sum_{x \in \mathbf{X}} Q(x)\right) \\ + \theta_1 \left(s_1 - \sum_{x \in \mathbf{X}} S_1(x) Q(x)\right) + \dots + \theta_m \left(s_m - \sum_{x \in \mathbf{X}} S_m(x) Q(x)\right)$$

## Maximum entropy solution

The solution of the constrained optimization problem is

$$P(x) = \frac{1}{Z(\theta)} e^{\langle S(x), \theta \rangle},$$

where  $\theta = (\theta_1, \dots, \theta_m)$ .

## Continuous distributions

Exponential family densities  $p(x|\theta)$  for continuous random variables can similarly be obtained as maximum entropy models given constraints of the form  $\mathbb{E}_p[S_j(x)] = s_j$ . This case requires more technicalities, due to the properties of the differential entropy.

## Statistical physics

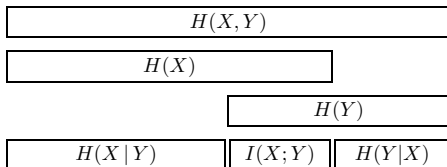
In physics, the maximum entropy distribution under given constraints is called the **Gibbs distribution**.

# SUMMARY: INFORMATION THEORY AND STATISTICS

- ▶ Maximum likelihood minimizes  $D_{\text{KL}}$  between empirical distribution and model.
- ▶ Variance, covariance and the  $\chi^2$ -statistic can be regarded as first-order approximations to entropy, mutual information and KL divergence.
- ▶ Various methods can be derived by substituting information-theoretic for traditional statistical quantities.
- ▶ Example: A dimension-reduction technique called *independent component analysis* can be motivated as (roughly speaking) a PCA-like method which measures independence in terms of mutual information rather than covariance.

# SUMMARY

The various additive relationships can be summarized as follows:



## Further reading

David J. C. MacKay: *Information Theory, Inference, and Learning Algorithms*.  
Cambridge University Press, 2003.

Online version: See link on course homepage.

# SEQUENTIAL DATA AND MARKOV MODELS



# MOTIVATION: PAGERANK

## Simple random walk

Start with a graph  $G$ . Define a random sequence of vertices as follows:

- ▶ Choose a vertex  $X_1$  uniformly at random.
- ▶ Choose a vertex  $X_2$  uniformly at random from the neighbors of  $X_1$ . Move to  $X_2$ .
- ▶ Iterate: At step  $n$ , uniformly sample a neighbor  $X_n$  of  $X_{n-1}$ , and move to  $X_n$ .

This is called *simple random walk* on  $G$ .

## Google's PageRank Algorithm

To sort the web pages matching a search query by importance, PageRank:

1. Defines a graph  $G$  whose vertices are web pages and whose edges are web links.
2. Computes the probability distribution on vertices  $x$  in  $G$  given by

$$P_n(x) = \Pr\{X_n = x\} \quad \text{where} \quad X_1, \dots, X_n \text{ is a simple random walk on } G$$

and  $n$  is very large.

We will try to understand (a) why and (b) how  $P_n$  can be computed.

## So far: I.i.d. sequences

We have assumed that samples are of the form

$$X_1 = x_1, X_2 = x_2, \dots \quad \text{where} \quad X_1, X_2, \dots \sim_{\text{iid}} P$$

for some distribution  $P$ . In particular, the order of observations does not matter.

## Now: Dependence on the past

We now consider sequences in which the value  $X_n$  can be stochastically dependent on  $X_1, \dots, X_{n-1}$ , so we have to consider conditional probabilities of the form

$$P(X_n | X_1, \dots, X_{n-1}) .$$

## Application examples

- ▶ Speech and handwriting recognition.
- ▶ Time series, e.g. in finance. (These often assume a *continuous* index. Our index  $n$  is discrete.)
- ▶ Simulation and estimation algorithms (Markov chain Monte Carlo).
- ▶ Random walk models (e.g. web search).

# MARKOV MODELS

## Markov models

The sequence  $(X_n)_n$  is called a **Markov chain of order**  $r$  if  $X_n$  depends only on a fixed number  $r$  of previous samples, i.e. if

$$P(X_n | X_{n-1}, \dots, X_1) = P(X_n | X_{n-1}, \dots, X_{n-r}).$$

If we simply call  $(X_n)_n$  a **Markov chain**, we imply  $r = 1$ .

## Initial state

The first state in the sequence is special because it does not have a "past", and is usually denoted  $X_0$ .

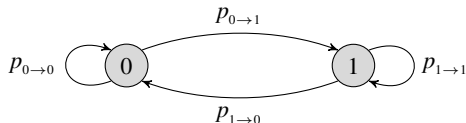
## Example: $r = 2$

$$\underbrace{X_0 = x_0, X_1 = x_1}_{X_4 \text{ is independent of these given } X_2, X_3}, \underbrace{X_2 = x_2, X_3 = x_3}_{X_4 \text{ may depend on these}}, X_4 = ?$$

# GRAPHICAL REPRESENTATION

## A simple binary chain

Suppose  $\mathbf{X} = \{0, 1\}$ .



- ▶ We regard 0 and 1 as possible "states" of  $X$ , represented as vertices in a graph.
- ▶ Each pair  $X_{n-1} = s, X_n = t$  in the sequence is regarded as a "transition" from  $s$  to  $t$  and represented as an edge in the graph.
- ▶ Each edge  $s \rightarrow t$  is weighted by the probability

$$p_{s \rightarrow t} := \Pr\{X_n = t | X_{n-1} = s\} .$$

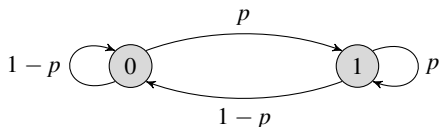
## State space

The elements of the sample space  $\mathbf{X}$  are called the **states** of the chain.  $\mathbf{X}$  is often called the **state space**. We generally assume that  $\mathbf{X}$  is finite, but Markov chains can be generalized to infinite and even uncountable state spaces.

# GRAPHICAL REPRESENTATION

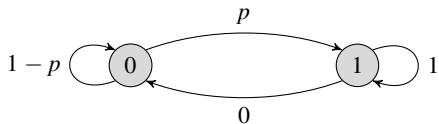
## First example: Independent coin flips

Suppose  $X$  is a biased coin with  $\Pr\{X_n = 1\} = p$  independently of  $X_{n-1}$ . In other words, the sequence  $(X_n)$  is iid Bernoulli with parameter  $p$ .



## Breaking independence

Here is a simple modification to the chain above; only  $p_{1 \rightarrow 0}$  and  $p_{1 \rightarrow 1}$  have changed:



This is still a valid Markov chain, but the elements of the sequence are no longer independent.

# GRAPHICAL REPRESENTATION

## Observation

The graph representation is only possible if  $p_{s \rightarrow t}$  is independent of  $n$ . Otherwise we would have to draw a different graph for each  $n$ .

If  $p_{s \rightarrow t}$  does not depend on  $n$ , the Markov chain is called **stationary**.

## Transition matrix

The probabilities  $p_{s \rightarrow t}$  are called the **transition probabilities** of the Markov chain. If  $|\mathbf{X}| = d$ , the  $d \times d$ -matrix

$$\mathbf{P} := (p_{i \rightarrow j})_{j,i \leq d} = \begin{pmatrix} p_{1 \rightarrow 1} & \cdots & p_{d \rightarrow 1} \\ \vdots & & \vdots \\ p_{1 \rightarrow d} & \cdots & p_{d \rightarrow d} \end{pmatrix}$$

is called the **transition matrix** of the chain (a left stochastic matrix). This is precisely the adjacency matrix of the graph representing the chain. Each column is a probability distribution on  $d$  events.

## Complete description of a Markov chain

The transition matrix does not completely determine the chain: It determines the probability of a state given a previous state, but not the probability of the starting state. We have to additionally specify the distribution of the first state.

## Initial distribution

The distribution of the first state, i.e. the vector

$$P_{\text{init}} := (\Pr\{X_0 = 1\}, \dots, \Pr\{X_0 = d\}) ,$$

is called the **initial distribution** of the Markov chain.

## Representing stationary Markov chains

Any stationary Markov chain with finite state space can be completely described by a transition matrix  $\mathbf{p}$  and an initial distribution  $P_{\text{init}}$ . That is, the pair  $(\mathbf{p}, P_{\text{init}})$  completely determines the joint distribution of the sequence  $(X_0, X_1, \dots)$ .

## Simple random walk

Suppose we are given a directed graph  $G$  (with unweighted edges). We had already mentioned that the **simple random walk** on  $G$  is the vertex-valued random sequence  $X_0, X_1, \dots$  defined as:

- ▶ We select a vertex  $X_0$  in  $G$  uniformly at random.
- ▶ For  $n = 1, 2, \dots$ , select  $X_n$  uniformly at random from the children of  $X_{n-1}$  in the graph.

## Markov chain representation

Clearly, the simple random walk on a graph with  $d$  vertices is a Markov chain with

$$P_{\text{init}} = \left( \frac{1}{d}, \dots, \frac{1}{d} \right) \quad \text{and} \quad p_{i \rightarrow j} = \frac{1}{\# \text{ edges out of } i}$$



## Generalizing simple random walk

We can generalize the idea of simple random walk by substituting the uniform distributions by other distributions. To this end, we can weight each edge in the graph by a probability of following that edge.

## Adjacency matrix

If the edge weights are proper probabilities, each column of the adjacency matrix must sum to one. In other words, the matrix is the transition matrix of a Markov chain.

## Random walks and Markov chains

If we also choose a general distribution for the initial state of the random walk, we obtain a completely determined Markov chain. Hence:

Any Markov chain on a finite state space is a random walk  
on a weighted graph, and vice versa.

## Queries

The first step in internet search is query matching:

1. The user enters a search query (a string of words).
2. The search engine determines all web pages indexed in its database which match the query.

This is typically a large set. For example, Google reports ca 83 million matches for the query "random walk".

## The ranking problem

- ▶ For the search result to be useful, the most useful link should with high probability be among the first few matches shown to the user.
- ▶ That requires the matching results to be *ranked*, i.e. sorted in order of decreasing "usefulness".

## Available data

Using a web crawler, we can (approximately) determine the link structure of the internet. That is, we can determine:

- ▶ Which pages there are.
- ▶ Which page links which.

A web crawler cannot determine:

- ▶ How often a link is followed.
- ▶ How often a page is visited.

## Web graph

The link structure can be represented as a graph with

vertices = web pages      and      edges = links.

# RANDOM WALK NETWORK MODELS

## Key idea

The popularity of a page  $x$  is proportional to the probability that a "random web surfer" ends up on page  $x$  after a  $n$  steps.

## Probabilistic model

The path of the surfer is modeled by a random walk on the web graph.

## Modeling assumptions

Two assumptions are implicit in this model:

1. Better pages are linked more often.
2. A link from a high-quality page is worth more than one from a low-quality page.

## Remarks

- ▶ We will find later that the choice of  $n$  does not matter.
- ▶ To compute the popularity score, we first have to understand Markov chains a bit better.

# STATE PROBABILITIES

## Probability after $n = 1$ steps

If we *know* the initial state, then

$$\Pr\{X_1 = s_1 \mid X_0 = s_0\} = p_{s_0 \rightarrow s_1} .$$

$P_1$  describes the probability of  $X_1$  if we do *not* know the starting state (i.e. the probability before we start the chain):

$$\begin{aligned} P_1(s_1) &= \Pr\{X_1 = s_1\} = \sum_{s_0 \in \mathbf{X}} \Pr\{X_1 = s_1 \mid X_0 = s_0\} P_{\text{init}}(s_0) \\ &= \sum_{s_0 \in \mathbf{X}} p_{s_0 \rightarrow s_1} P_{\text{init}}(s_0) . \end{aligned}$$

## Matrix representation

Recall that  $\mathbf{p}$  is a  $d \times d$ -matrix and  $P_{\text{init}}$  a vector of length  $d$ . The equation for  $P_1$  above is a matrix-vector product, so

$$P_1 = \mathbf{p} \cdot P_{\text{init}} .$$

## Probability after $n = 2$ steps

The same argument shows that  $P_2$  is given by

$$P_2(s_2) = \sum_{s_1 \in \mathbf{X}} p_{s_1 \rightarrow s_2} P_1(s_1) ,$$

hence

$$P_2 = \mathbf{p} \cdot P_1 = \mathbf{p} \cdot \mathbf{p} \cdot P_{\text{init}} .$$

## For arbitrary $n$

$$P_n = \mathbf{p}^n P_{\text{init}}$$

## Limiting distribution

Instead of considering  $P_n$  for a specific, large  $n$ , we take the limit

$$P_\infty := \lim_{n \rightarrow \infty} P_n = \lim_{n \rightarrow \infty} \mathbf{p}^n P_{\text{init}} ,$$

provided that the limit exists.

## Observation

If the limit  $P_\infty$  exists, then

$$\mathbf{p} \cdot P_\infty = \mathbf{p} \cdot \lim_{n \rightarrow \infty} \mathbf{p}^n P_{\text{init}} = \lim_{n \rightarrow \infty} \mathbf{p}^{n+1} P_{\text{init}} = P_\infty ,$$

which motivates the next definition.

## Equilibrium distribution

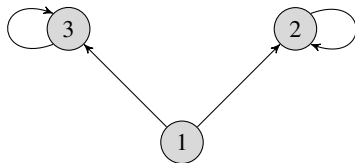
If  $\mathbf{p}$  is the transition matrix of a Markov chain, a distribution  $P$  on  $\mathbf{X}$  which is invariant under  $\mathbf{p}$  in the sense that

$$\mathbf{p} \cdot P = P$$

is called an **equilibrium distribution** or **invariant distribution** of the Markov chain.

# WHAT CAN GO WRONG?

## Problem 1: The equilibrium distribution may not be unique



For this chain, both  $P = (0, 1, 0)$  and  $P' = (0, 0, 1)$  are valid equilibria. Which one emerges depends on the initial state and (if we start in state 1) on the first transition.

### Remedy

Require that there is a path in the graph (with non-zero probability) from each state to every other state. A Markov chain satisfying this condition is called **irreducible**.



# WHAT CAN GO WRONG?

Recall that a sequence in  $\mathbb{R}$  does not have a limit if it "oscillates". For example,

$$\lim_n 1^n = 1 \quad \text{but} \quad \lim_n (-1)^n \text{ does not exist}$$

## Problem 2: The limit may not exist

- ▶ The chain on the right has no limit distribution.
- ▶ If we start e.g. in state 0, then:
  - ▶ 0 can only be reached in even steps.
  - ▶ 1 only in odd steps.
- ▶ The distribution  $P_n$  oscillates between

$$P_{\text{even}} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{and} \quad P_{\text{odd}} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} .$$



# WHAT CAN GO WRONG?

## Remedy

To prevent this (particular) problem, we can add two edges:



Now each state is reachable in every step.

The problem (at least this example) is that we have to leave the state before we can return to it. We prevent this, we introduce the following definition.

## Aperiodic chains

We call a stationary Markov chain **aperiodic** if, for every state  $s$ ,

$$\Pr\{X_n = s \mid X_{n-1} = s\} = p_{s \rightarrow s} > 0 .$$

In short, a stationary chain is aperiodic if the transition matrix has non-zero diagonal.

# EQUILIBRIUM DISTRIBUTIONS

We have introduced two definitions which prevent two rather obvious problems. Surprisingly, these definitions are all we need to guarantee limits.

## Theorem

Suppose a Markov chain  $(\mathbf{p}, P_{\text{init}})$  is stationary, and for each state  $s \in \mathbf{X}$ :

1. There is a path (with non-zero probability) from  $s$  to every other state (i.e. the chain is irreducible).
2.  $p_{s \rightarrow s} > 0$  (i.e. the chain is aperiodic).

Then:

- ▶ The limit distribution  $P_{\infty}$  exists.
- ▶ The limit distribution is also the equilibrium distribution.
- ▶ The equilibrium distribution is unique.

# COMPUTING THE EQUILIBRIUM

## Power method

If the the transition matrix  $\mathbf{p}$  makes the chain irreducible and aperiodic, we know that

equilibrium distribution = limit distribution .

This means we can compute the approximate the equilibrium  $P_\infty$  by  $P_n$ . In other words, we start with any distribution  $P_{\text{init}}$  (e.g. uniform) and repeatedly multiply by  $\mathbf{p}$ :

$$P_{n+1} = \mathbf{p} \cdot P_n$$

We can threshold the change between steps, e.g. by checking  $\|P_{n+1} - P_n\| < \tau$  for some small  $\tau$ .

## Remark: Eigenstructure

The power method can be regarded as an eigenvector computation. The definition

$$P = \mathbf{p} \cdot P$$

of the equilibrium means that  $P = P_\infty$  is an eigenvector of  $\mathbf{p}$  with eigenvalue 1. If  $\mathbf{p}$  is irreducible and aperiodic, it can be shown that 1 is the largest eigenvalue.

## Constructing the transition matrix

We start with the web graph and construct the transition matrix of simple random walk, i.e.

$$a_{ij} := \begin{cases} \frac{1}{\# \text{ edges out of } i} & \text{if } i \text{ links to } j \\ 0 & \text{otherwise} \end{cases}$$

A chain defined by  $A := (a_{ij})$  will almost certainly not be irreducible (think of web pages which do not link anywhere). We therefore regularize  $A$  by defining

$$\mathbf{p} := (1 - \alpha)A + \frac{\alpha}{d} \begin{pmatrix} 1 & \cdots & 1 \\ \vdots & & \vdots \\ 1 & \cdots & 1 \end{pmatrix}$$

for some small  $\alpha > 0$ .

Clearly, this makes  $\mathbf{p}$  both irreducible and aperiodic.

## Computing the equilibrium

Given  $\mathbf{p}$ , the equilibrium distribution is computed using the power method. Since the web changes, the power method can be re-run every few days with the previous equilibrium as initial distribution.

# THE RANDOM SURFER AGAIN

We can now take a more informed look at the idea of a random web surfer:

- ▶ Suppose the surfer is more likely to start on a popular page than on an unpopular one.
- ▶ In terms of the popularity model, this means

$$X_0 \sim P_{\text{equ}} ,$$

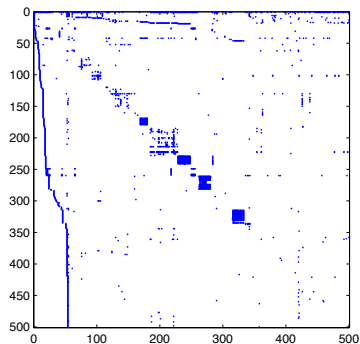
where  $P_{\text{equ}}$  is the equilibrium distribution of the chain.

- ▶ After following any number of links  $n$  (with probabilities given by the transition matrix  $\mathbf{p}$ ),

$$P_n = \mathbf{p}^n P_{\text{equ}} = P_{\text{equ}} .$$

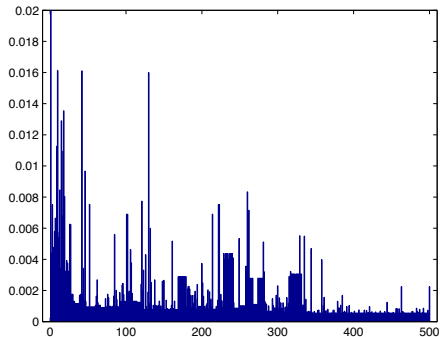
- ▶ In this sense,  $P_{\text{equ}}$  is really the consistent solution to our problem, even if we *compute* it by starting the random walk from e.g. a uniform initial distribution instead.
- ▶ In particular, it does not matter how we choose  $n$  in the model.

# EXAMPLE



Adjacency matrix of the web graph of 500 web pages.

The root (index 0) is [www.harvard.edu](http://www.harvard.edu).



Equilibrium distribution computed by PageRank.

# HIDDEN MARKOV MODELS



## Motivation

We have already used Markov models to model sequential data. Various important types of sequence data (speech etc) have long-range dependencies that a Markov model does not capture well.

## Hidden Markov model

- ▶ A hidden Markov model is a latent variable model in which a sequence of latent (or "hidden") variables is generated by a Markov chain.
- ▶ These models can generate sequences of *observations* with long-range dependencies, but the *explanatory* variables (the latent variables) are Markovian.
- ▶ It turns out that this is exactly the right way to model dependence for a variety of important problems, including speech recognition, handwriting recognition, and parsing problems in genetics.

# HIDDEN MARKOV MODELS

## Definition

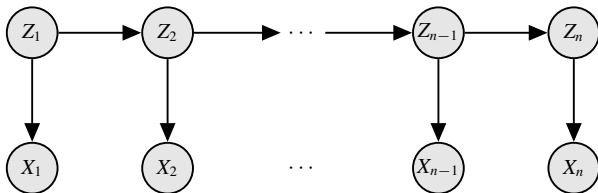
A **(discrete) hidden Markov model (HMM)** consists of:

- ▶ A stationary Markov chain  $(Q_{\text{init}}, \mathbf{q})$  with states  $\{1, \dots, K\}$ , initial distribution  $Q_{\text{init}}$  and transition matrix  $\mathbf{q}$ .
- ▶ A (discrete) **emission distribution**, given by a conditional probability  $P(x|z)$ .

The model generates a sequence  $X_1, X_2, \dots$  by:

1. Sampling a sequence  $Z_1, Z_2, \dots$  from the Markov chain  $(Q_{\text{init}}, \mathbf{q})$ .
2. Sampling a sequence  $X_1, X_2, \dots$  by independently sampling  $X_i \sim P(\cdot | Z_i)$ .

In a **continuous HMM**, the variables  $X_i$  have continuous distributions, and  $P(x|z)$  is substituted by a density  $p(x|z)$ . The Markov chain still has finite state space  $[K]$ .



We will see a lot of sequences, so we use the "programming" notation

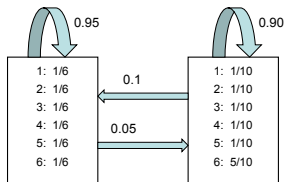
$$x_{1:n} := (x_1, \dots, x_n)$$

# EXAMPLE: DISHONEST CASINO

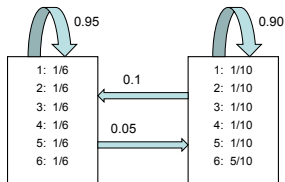
This example is used in most textbooks and is very simple, but it is useful to understand the conditional independence structure.

## Problem

- ▶ We consider two dice (one fair, one loaded).
- ▶ At each roll, we either keep the current die, or switch to the other one with a certain probability.
- ▶ A roll of the chosen die is then observed.



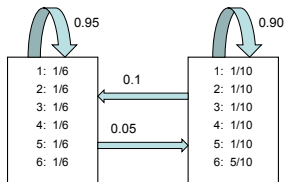
# EXAMPLE: DISHONEST CASINO



## HMM

- ▶ States:  $Z_n \in \{\text{fair}, \text{loaded}\}$ .
- ▶ Sample space:  $\mathbf{X} = \{1, \dots, 6\}$ .
- ▶ Transition matrix:  $\mathbf{q} = \begin{pmatrix} 0.95 & 0.05 \\ 0.10 & 0.90 \end{pmatrix}$
- ▶ Emission probabilities:  
 $P(x|z = \text{fair}) = (1/6, 1/6, 1/6, 1/6, 1/6, 1/6)$   
 $P(x|z = \text{loaded}) = (1/10, 1/10, 1/10, 1/10, 1/10, 5/10)$

# EXAMPLE: DISHONEST CASINO



## Conditional independence

- ▶ Given the state (=which dice), the outcomes are independent.
- ▶ If we do not know the current state, observations are dependent!
- ▶ For example: If we observe sequence of sixes, we are more likely to be in state "loaded" than "fair", which increases the probability of the next observation being a six.

# HMM: ESTIMATION PROBLEMS

## Filtering problem

- ▶ **Given:** Model and observations, i.e. :
  1. Transition matrix  $\mathbf{q}$  and emission distribution  $P(\cdot | z)$ .
  2. Observed sequence  $x_{1:N} = (x_1, \dots, x_N)$ .
- ▶ **Estimate:** Probability of each hidden variable, i.e.  $Q(Z_n = k | x_{1:n})$

Variant: **Smoothing problem**, in which we estimate  $Q(Z_n = k | x_{1:N})$  instead.

## Decoding problem

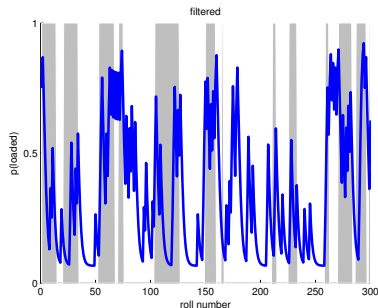
- ▶ **Given:** Model ( $\mathbf{q}$  and  $P(\cdot | z)$ ) and observed sequence  $x_{1:N}$ .
- ▶ **Estimate:** Maximum likelihood estimates  $\hat{z}_{1:N} = (\hat{z}_1, \dots, \hat{z}_N)$  of hidden states.

## Learning problem

- ▶ **Given:** Observed sequence  $x_{1:N}$ .
- ▶ **Estimate:** Model (i.e.  $\mathbf{q}$  and  $P(\cdot | z)$ ).

# EXAMPLES

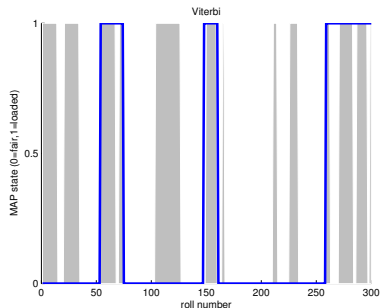
Before we look at the details, here are examples for the dishonest casino.



Filtering result.

Gray bars: Loaded dice used.

Blue: Probability  $P(Z_n = \text{loaded} | x_{1:N})$



Decoding result.

Gray bars: Loaded dice used.

Blue: Most probable state  $Z_n$ .



# PROBABILITIES OF HIDDEN STATES

The first estimation problem we consider is to estimate the probabilities  $Q(z_n|x_{1:n})$ .

## Idea

We could use Bayes' equation (recall:  $P(a|b) = \frac{P(b|a)P(a)}{P(b)}$ ) to write:

$$Q(k|x_n) = \frac{P(x_n|k)Q(Z_n = k)}{\sum_{j=1}^K P(x_n|k)Q(Z_n = k)} .$$

Since we know the Markov chain  $(Q_{\text{init}}, \mathbf{q})$ , we can compute  $Q$ , and the emission probabilities  $P(x_n|k)$  are given.

## Filtering

The drawback of the solution above is that it throws away all information about the past. We get a better estimate of  $Z_n$  by taking  $x_1, \dots, x_{n-1}$  into account. Reducing the uncertainty in  $Z_n$  using  $x_1, \dots, x_{n-1}$  is called **filtering**.

## Filtering problem

Our task is to estimate the probabilities  $Q(z_n | x_{1:n})$ . Since the sequence has length  $n$  and each  $Z_i$  can take  $K$  possible values, this is a  $N \times K$ -matrix  $\hat{Q}$ , with entries

$$\hat{Q}_{nk} := Q(Z_n = k | x_{1:n}) .$$

## Decomposition using Bayes' equation

We can use Bayes' equation (recall:  $P(a|b) = \frac{P(b|a)P(a)}{P(b)}$ ) to write:

$$Q(z_n | x_{1:n}) = Q(z_n | x_n, x_{1:(n-1)}) = \frac{\overbrace{P(x_n | z_n, x_{1:(n-1)})}^{\text{This is the emission probability } P(x_n | z_n) \text{ (conditional independence!)}} \underbrace{Q(z_n | x_{1:(n-1)})}_{\text{This is the crucial term}}}{\underbrace{\sum_{z_n=1}^K P(x_n | z_n, x_{1:(n-1)}) Q(z_n | x_{1:(n-1)})}_{\text{Normalization}}}$$

## Reduction to previous step

The crucial idea is that we can use the results computed for step  $n - 1$  to compute those for step  $n$ :

$$Q(Z_n = k | x_{1:(n-1)}) = \sum_{l=1}^K \underbrace{Q(Z_n = k | Z_{n-1} = l)}_{= q_{lk} \text{ (transition matrix)}} \underbrace{Q(Z_{n-1} = l | x_{1:(n-1)})}_{= \hat{Q}_{(n-1)l}}$$

## Summary

In short, we can compute the numerator in the Bayes equation as

$$a_{nk} := P(x_n | z_n) \sum_{l=1}^K q_{lk} \hat{Q}_{(n-1)l} .$$

The normalization term is

$$\sum_{z_n=1}^K \left( P(x_n | z_n) \sum_{l=1}^K q_{lk} \hat{Q}_{(n-1)l} \right) = \sum_{j=1}^K a_{nj} .$$

## Solution to the filtering problem: The forward algorithm

Given is a sequence  $(x_1, \dots, x_N)$ .

For  $n = 1, \dots, N$ , compute

$$a_{nk} := P(x_n | z_n) \sum_{l=1}^K q_{lk} \hat{Q}_{(n-1)l} ,$$

and

$$\hat{Q}_{nk} = \frac{a_{nk}}{\sum_{j=1}^K a_{nj}} .$$

This method is called the **forward algorithm**.

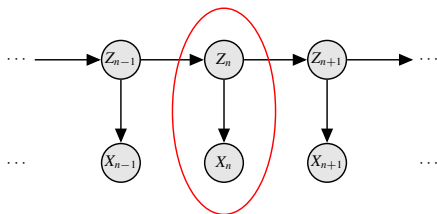
# HMMs AND MIXTURE MODELS

## Parametric emission model

We usually define the emission probabilities  $P(x_n|z_n)$  using a parametric model  $P(x|\theta)$  (e.g. a multinomial or Gaussian model). Then

$$P(x_n|Z_n = k) := P(x_n|\theta_k) ,$$

i.e. the emission distribution of each state  $k$  is defined by a parameter value  $\theta_k$ .



## Relation to mixture models

If we just consider a *single* pair  $(Z_n, X_n)$ , this defines a finite mixture with  $K$  clusters:

$$\pi(x_n) = \sum_{k=1}^K c_k P(x_n|\theta_k) = \sum_{k=1}^K Q(Z_n = k) P(x_n|\theta_k)$$

# EM FOR HMMS

## Recall: EM for mixtures

E-step	M-step
Soft assignments $\mathbb{E}[M_{ik}] = \Pr(m_i = k)$	cluster weights $c_k$ component parameters $\theta_k$

## HMM case

- ▶ For mixtures,  $\Pr\{m_i = k\} = c_k$ . In HMMs, the analogous probability  $\Pr\{Z_n = k\}$  is determined by the transition probabilities.
- ▶ The analogue of the soft assignments  $a_{ik}$  computed for mixtures are state probabilities

$$b_{nk} = Q(Z_n = k | \theta, x_{1:N}) .$$

- ▶ Additionally, we have to estimate the transition matrix  $\mathbf{q}$  of the Markov chain.

## EM for HMMs

E-step	M-step
Transition probabilities $q_{kj}$ State probabilities $b_{nk}$	component parameters $\theta_k$

## M-step

The M-step works exactly as for mixture models. E.g. for Gaussian emission distributions with parameters  $\mu_k$  and  $\sigma_k^2$ ,

State probabilities substituted  
for assignment probabilities

$$\mu_k = \frac{\sum_{n=1}^N b_{nk} x_n}{\sum_{n=1}^N b_{nk}}$$

and

$$\sigma_k^2 = \frac{\sum_{n=1}^N b_{nk} (x_n - \mu_k)^2}{\sum_{n=1}^N b_{nk}}$$

## E-step

- ▶ Computing the state probabilities is a filtering problem:

$$b_{nk}^{\text{new}} = Q(Z_n = k | \theta^{\text{old}}, x_{1:n}) .$$

The forward-backward algorithm assumes the emission probabilities are known, so we use the emission parameters  $\theta^{\text{old}}$  computed during the previous M-step.

- ▶ Estimating the transition probabilities is essentially a filtering-type problem for *pairs* of states and can also be solved recursively, but we will skip the details since the equations are quite lengthy.

# APPLICATION: SPEECH RECOGNITION

## Problem

Given speech in form of a sound signal, determine the words that have been spoken.

## Method

- ▶ Words are broken down into small sound units (called *phonemes*). The states in the HMM represent phonemes.
- ▶ The incoming sound signal is transformed into a sequence of vectors (feature extraction). Each vector  $x_n$  is indexed by a time step  $n$ .
- ▶ The sequence  $x_{1:N}$  of feature vectors is the observed data in the HMM.



# PHONEME MODELS

## Phoneme

A **phoneme** is defined as the smallest unit of sound in a language that distinguishes between distinct meanings. English uses about 50 phonemes.

## Example

---

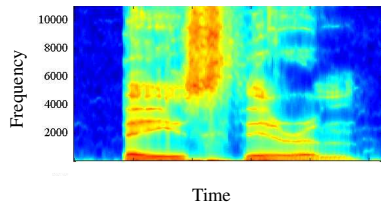
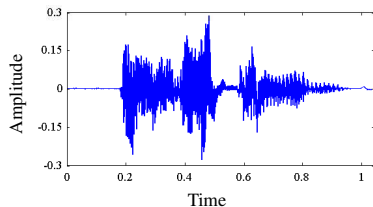
Zero	Z I H R O W	Six	S I H K S
One	W A H N	Seven	S E H V A X N
Two	T U W	Eight	E Y T
Three	T H R I Y	Nine	N A Y N
Four	F O W R	Oh	O W
Five	F A Y V		

---

## Subphonemes

Phonemes can be further broken down into subphonemes. The standard in speech processing is to represent a phoneme by three subphonemes ("triphons").

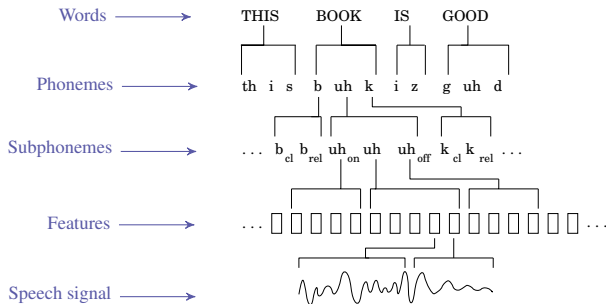
# PREPROCESSING SPEECH



## Feature extraction

- ▶ A speech signal is measured as amplitude over time.
- ▶ The signal is typically transformed into various types of features, including (windowed) Fourier- or cosine-transforms and so-called "cepstral features".
- ▶ Each of these transforms is a scalar function of time. All function values for the different transforms at time  $t$  are collected in a vector, which is the feature vector (at time  $t$ ).

# LAYERS IN PHONEME MODELS



## HMM speech recognition

- ▶ Training: The HMM parameters (emission parameters and transition probabilities) are estimated from data, often using both supervised and unsupervised techniques.
- ▶ Recognition: Given a language signal (= observation sequence  $x_{1:N}$ , estimate the corresponding sequence of subphonemes (= states  $z_{1:N}$ ). This is a decoding problem.

## Factory model

Training requires a lot of data; software is typically shipped with a model trained on a large corpus (i.e. the HMM parameters are set to "factory settings").

## The adaptation problem

- ▶ The factory model represents an average speaker. Recognition rates can be improved drastically by adapting to the specific speaker using the software.
- ▶ Before using the software, the user is presented with a few sentences and asked to read them out, which provides labelled training data.

## Speaker adaptation

- ▶ Transition probabilities are properties of the language. Differences between speakers (pronunciation) are reflected by the emission parameters  $\theta_k$ .
- ▶ Emission probabilities in speech are typically multi-dimensional Gaussians, so we have to adapt means and covariance matrices.
- ▶ The arguably most widely used method is **maximum likelihood linear regression (MLLR)**, which uses a regression technique to make small changes to the covariance matrices.

## More details on HMMs

If you feel enthusiastic, the following books provide more background:

- ▶ David Barber's "Bayesian reasoning and machine learning" (available online; see class homepage).
- ▶ Chris Bishop's "Pattern recognition and machine learning".
- ▶ Many books on speech, e.g. Rabiner's classic "Fundamentals of speech recognition".

## HTK

If you would like to try out speech recognition software, have a look at the HTK (**H**MM **T**oolkit) package, which is the de-facto standard in speech research. HTK implements both HMMs for recognition and routines for feature extraction.

# BAYESIAN MODELS

## Approach

The defining assumption of Bayesian statistics is that the distribution  $P$  which explains the data is *a random quantity* and itself has a distribution  $Q$ . The generative model for data  $X_1, X_2, \dots$  is

$$P \sim Q$$
$$X_1, X_2, \dots \sim_{\text{iid}} P$$

## Rationale

- ▶ In any statistical approach (Bayesian or classical), the distribution  $P$  is unknown.
- ▶ Bayesian statistics argues that any form of uncertainty should be expressed by probability distributions.
- ▶ We can think of the randomness in  $Q$  as a model of the statistician's lack of knowledge regarding  $P$ .

An older name for Bayesian statistics is *inverse probability*.

## Prior and Posterior

The distribution  $Q$  of  $P$  is called the **a priori distribution** (or the **prior** for short). Our objective is to determine the conditional probability of  $P$  given observed data,

$$\Pi[P|x_1, \dots, x_n] .$$

This distribution is called the **a posteriori distribution** or **posterior**.

## Parametric case

We can impose the modeling assumption that  $P$  is an element of a parametric model, e.g. that the density  $p$  of  $P$  is in a family  $\mathcal{P} = \{p(x|\theta) | \theta \in \mathcal{T}\}$ . If so, the prior and posterior can be expressed as distributions on  $\mathcal{T}$ . We write

$$q(\theta) \quad \text{and} \quad \Pi(\theta|x_1, \dots, x_n)$$

for the prior and posterior density, respectively.

## Remark

The posterior  $\Pi[P|x_1, \dots, x_n]$  is an abstract object, which can be rigorously defined using the tools of probability theory, but is in general (even theoretically) impossible to compute. However: In the parametric case, the posterior can be obtained using the Bayes equation.



# COMPUTING PARAMETRIC POSTERiors

## Parametric modeling assumption

Suppose  $\mathcal{P} = \{p(x|\theta)|\theta \in \mathcal{T}\}$  is a model and  $q$  a prior distribution on  $\mathcal{T}$ . Our sampling model then has the form:

$$\begin{aligned}\theta &\sim q \\ X_1, X_2, \dots &\sim_{\text{iid}} p(\cdot|\theta)\end{aligned}$$

Note that the data is *conditionally i.i.d.* given  $\Theta = \theta$ .

## Bayes' Theorem

If  $\mathcal{P}$  is a parametric Bayesian model and  $q$  a distribution on  $\mathcal{T}$ , the posterior under data  $X_1, \dots, X_n$  generated as above is

$$\Pi(\theta|x_1, \dots, x_n) = \frac{\left(\prod_{i=1}^n p(x_i|\theta)\right)q(\theta)}{p(x_1, \dots, x_n)} = \frac{\left(\prod_{i=1}^n p(x_i|\theta)\right)q(\theta)}{\int_{\mathcal{T}} \left(\prod_{i=1}^n p(x_i|\theta)\right)q(\theta)d\theta}$$

The individual terms have names:

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

# EXAMPLE: UNKNOWN GAUSSIAN MEAN

## Model

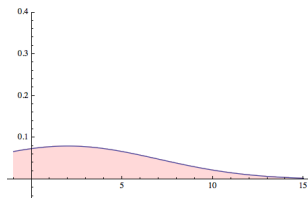
We assume that the data is generated from a Gaussian with fixed variance  $\sigma^2$ . The mean  $\mu$  is unknown. The model likelihood is  $p(x|\mu, \sigma) = g(x|\mu, \sigma)$  (where  $g$  is the Gaussian density on the line).

## Bayesian model

We choose a Gaussian prior on  $\mu$ ,

$$q(\mu) := g(\mu|\mu_0, \sigma_0) .$$

In the figure,  $\mu_0 = 2$  and  $\sigma_0 = 5$ . Hence, we assume that  $\mu_0 = 2$  is the most probable value of  $\mu$ , and that  $\mu \in [-3, 7]$  with a probability  $\sim 0.68$ .



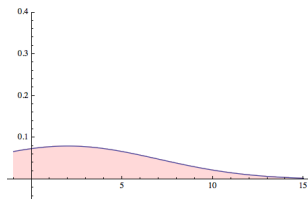
## Posterior

Application of Bayes' formula to the Gaussian-Gaussian model shows

$$\Pi(\mu|x_{1:n}) = g(\mu|\mu_n, \sigma_n) \quad \text{where} \quad \mu_n := \frac{\sigma^2 \mu_0 + \sigma_0^2 \sum_{i=1}^n x_i}{\sigma^2 + n\sigma_0^2} \quad \text{and} \quad \sigma_n := \frac{\sigma^2 \sigma_0^2}{\sigma^2 + n\sigma_0^2}$$

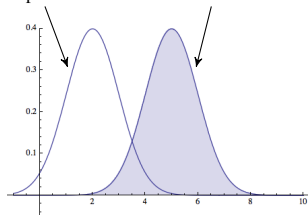
# EXAMPLE: UNKNOWN GAUSSIAN MEAN

## Model



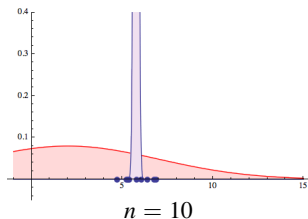
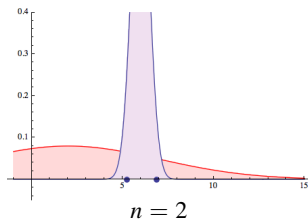
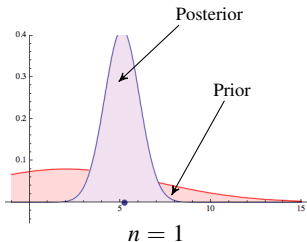
Prior

most probable model  
under the prior



Sampling distribution

## Posterior distributions

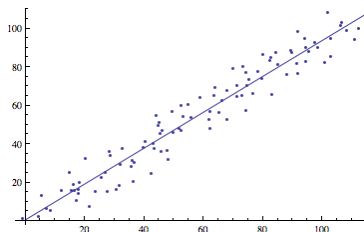


# A SLIGHTLY DIFFERENT PERSPECTIVE

## Parameters

Intuitively, we can think of  $\theta$  as the common pattern underlying the data:

$$P(X|\theta) = \text{Probability}[\text{data}|\text{pattern}]$$



## Inference idea

data = underlying pattern + independent randomness

Broadly speaking, the goal of statistics is to extract the pattern from the data.  
Bayesian statistics models the pattern as a random quantity.

# MAP ESTIMATION

## Definition

Suppose  $\Pi(\theta|x_{1:n})$  is the posterior of a Bayesian model. The estimator

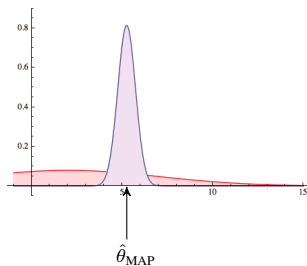
$$\hat{\theta}_{\text{MAP}} := \arg \max_{\theta \in \mathcal{T}} \Pi(\theta|x_{1:n})$$

is called the **maximum a posteriori** (or **MAP**) estimator for  $\theta$ .

## Point estimates

The goal of Bayesian inference is to compute the posterior distribution. Contrast this to classical statistics (e.g. maximum likelihood), where we typically estimate a single value for  $\theta$  (a so-called **point estimate**).

MAP estimation combines aspects of Bayesian methodology (use of a prior) with aspects of classical methodology (since  $\hat{\theta}_{\text{MAP}}$  is a point estimate).



# MAP AND REGULARIZATION

## Logarithmic view

Since the logarithm leaves the maximum invariant,

$$\hat{\theta}_{\text{MAP}} = \arg \max_{\theta \in \mathcal{T}} \Pi(\theta|x_{1:n}) = \arg \max_{\theta \in \mathcal{T}} \log \Pi(\theta|x_{1:n})$$

Substituting in the Bayes equation gives

$$\log \Pi(\theta|x_{1:n}) = \sum_{i=1}^n \log p(x_i|\theta) + \log q(\theta) - \log p(x_1, \dots, x_n) .$$

## MAP as regularized ML

Since log-evidence does not depend on  $\theta$ ,

$$\hat{\theta}_{\text{MAP}} = \arg \max_{\theta \in \mathcal{T}} \left\{ \sum_{i=1}^n \log p(x_i|\theta) + \log q(\theta) \right\}$$

Thus, the MAP estimate can be regarded as a regularized version of a maximum likelihood estimator. The regularization term  $\log q(\theta)$  favors values where  $q$  (and hence  $\log q$ ) is large.

## Families of priors

The prior has to be expressed by a specific distribution. In parametric Bayesian models, we typically choose  $q$  as an element of a standard parametric family (e.g. the Gaussian in the previous example).

## Hyperparameters

If we choose  $q$  as an element of a parametric family

$$\mathcal{Q} = \{q(\theta|\phi) | \phi \in \mathcal{H}\}$$

on  $\mathcal{T}$ , selecting the prior comes down to choosing  $\phi$ . Hence,  $\phi$  becomes a tuning parameter of the model.

Parameter of the prior family are called **hyperparameters** of the Bayesian model.

# NATURAL CONJUGATE PRIORS

## Exponential family likelihood

We now assume the parametric model  $\mathcal{P} = \{p(x|\theta)|\theta \in \mathcal{T}\}$  is an exponential family model, i.e.

$$p(x|\theta) = \frac{h(x)}{Z(\theta)} e^{\langle s(x), \theta \rangle} .$$

## Natural conjugate prior

We define a prior distribution using the density

$$q(\theta|\lambda, y) = \frac{1}{K(\lambda, y)} \exp\left(\langle \theta, y \rangle - \lambda \cdot \log Z(\theta)\right)$$

- ▶ Hyperparameters:  $\lambda \in \mathbb{R}_+$  and  $y \in \mathcal{T}$ .
- ▶ Note that the choice of  $\mathcal{P}$  enters through  $Z$ .
- ▶  $K$  is a normalization function.

Clearly, this is itself an exponential family (on  $\mathcal{T}$ ), with  $h \equiv Z^{-\lambda}$  and  $Z \equiv K$ .



# UGLY COMPUTATION

Substitution into Bayes' equation gives

$$\begin{aligned}\Pi(\theta|x_1, \dots, x_n) &= \frac{\prod_{i=1}^n p(x_i|\theta)}{p(x_1, \dots, x_n)} \cdot q(\theta) \\ &= \frac{\frac{\prod_{i=1}^n h(x_i)}{Z(\theta)^n} \exp\langle \sum_i S(x_i), \theta \rangle}{p(x_1, \dots, x_n)} \cdot \frac{\exp(\langle \theta, y \rangle - \lambda \log Z(\theta))}{K(\lambda, y)}\end{aligned}$$

If we neglect all terms which do not depend on  $\theta$ , we have

$$\Pi(\theta|x_1, \dots, x_n) \propto \frac{\exp\langle \sum_i S(x_i), \theta \rangle}{Z(\theta)^n} \exp(\langle \theta, y \rangle - \lambda \log Z(\theta)) = \frac{\exp(\langle y + \sum_i S(x_i), \theta \rangle)}{Z(\theta)^{\lambda+n}}$$

Up to normalization, this is precisely the form of an element of  $\mathcal{Q}$ :

$$\dots = \exp\left(\left\langle y + \sum_i S(x_i), \theta \right\rangle - (\lambda + n) \log Z(\theta)\right) \propto q(\theta|\lambda + n, y + \sum_{i=1}^n S(x_i))$$

# POSTERIOR OF CONJUGATE PRIORS

## Conclusion

If  $\mathcal{P}$  is an exponential family model with sufficient statistic  $S$ , and if  $q(\theta|\lambda, y)$  is a natural conjugate prior for  $\mathcal{P}$ , the posterior under observations  $x_1, \dots, x_n$  is

$$\Pi(\theta|x_1, \dots, x_n) = q(\theta|\lambda + n, y + \sum_{i=1}^n S(x_i))$$

## Remark

The form of the posterior above means that we can *compute the posterior by updating the hyperparameters*. This property motivates the next definition.

## Definition

Assume that  $\mathcal{P}$  is a parametric family and  $\mathcal{Q}$  a family of priors. Suppose, for each sample size  $n \in \mathbb{N}$ , there is a function  $T_n : \mathbf{X}^n \times \mathcal{H} \rightarrow \mathcal{H}$  such that

$$\Pi(\theta|x_1, \dots, x_n) = q(\theta|\hat{\phi}) \quad \text{with} \quad \hat{\phi} := T_n(x_1, \dots, x_n, \phi) .$$

Then  $\mathcal{P}$  and  $\mathcal{Q}$  are called **conjugate**.

# CONJUGATE PRIORS

## Closure under sampling

If the posterior is an element of the prior family, i.e. if

$$\Pi(\theta|x_1, \dots, x_n) = q(\theta|\tilde{\phi})$$

for *some*  $\tilde{\phi}$ , the model is called **closed under sampling**. Clearly, every conjugate model is closed under sampling.

## Remark

Closure under sampling is a weaker property than conjugacy; for example, any Bayesian model with

$$\mathcal{Q} = \{ \text{all probability distributions on } \mathcal{T} \}$$

is trivially closed under sampling, but not conjugate.

**Warning:** Many Bayesian texts use conjugacy and closure under sampling equivalently.

## Which models are conjugate?

It can be shown that, up a few "borderline" cases, the only parametric models which admit conjugate priors are exponential family models.

# NATURAL CONJUGATE POSTERIOR

## Generic posterior updates

For an exponential family  $\mathcal{P}$  with natural conjugate family  $\mathcal{Q}$ , the posterior is computed as the hyperparameter update

$$T_n(x_1, \dots, x_n, \lambda, y) = (\lambda + n, y + \sum_{i=1}^n S(x_i)) .$$

## Effect of hyperparameters

The natural conjugate prior  $q(\theta|\lambda, y)$  has expected value  $\mathbb{E}[\Theta] = y$ . The parameter  $\lambda$  is a concentration, i.e.

large  $\lambda \iff$  prior peaks sharply around  $y$  .

## Interpretation of posterior updates

The posterior mean is

$$\mathbb{E}[\Theta] = y + \sum_{i=1}^n S(x_i) ,$$

i.e. we linearly interpolate the prior guess and the sufficient statistics of the data in parameter space. The more data we observe, the larger the posterior concentration  $\lambda + n$ , which reflects increasing certainty regarding  $\Theta$  given more data.

# SAMPLING ALGORITHMS

## In general

- ▶ A **sampling algorithm** is an algorithm that outputs samples  $x_1, x_2, \dots$  from a given distribution  $P$  or density  $p$ .
- ▶ Sampling algorithms can for example be used to approximate expectations:

$$\mathbb{E}_p[f(X)] \approx \frac{1}{n} \sum_{i=1}^n f(x_i)$$

## Inference in Bayesian models

Suppose we work with a Bayesian model whose posterior  $\Pi$  cannot be computed analytically.

- ▶ We will see that it can still be possible to *sample* from  $\Pi$ .
- ▶ Doing so, we obtain samples  $\theta_1, \theta_2, \dots$  distributed according to  $\Pi$ .
- ▶ This reduces posterior estimation to a density estimation problem (i.e. estimate  $\Pi$  from  $\theta_1, \theta_2, \dots$ ).

## Posterior expectations

If we are only interested in some statistic of the posterior of the form  $\mathbb{E}_{\Pi}[f(\Theta)]$  (e.g. the posterior mean  $\mathbb{E}_{\Pi}[\Theta]$ ), we can again approximate by

$$\mathbb{E}_{\Pi}[f(\Theta)] \approx \frac{1}{m} \sum_{i=1}^m f(\theta_i) .$$

## Example: Predictive distribution

The **posterior predictive distribution** is our best guess of what the next data point  $x_{n+1}$  looks like, given the posterior under previous observations:

$$p(x_{n+1}|x_1, \dots, x_n) := \int_{\mathcal{T}} p(x_{n+1}|\theta)\Pi(\theta|x_1, \dots, x_n)d\theta .$$

This is one of the key quantities of interest in Bayesian statistics.

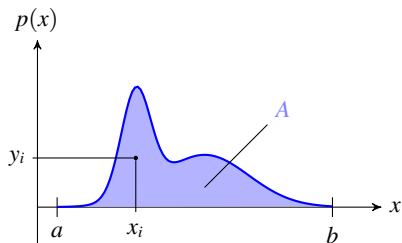
## Computation from samples

The predictive is a posterior expectation, and can be approximated as a sample average:

$$p(x_{n+1}|x_{1:n}) = \mathbb{E}_{\Pi}[p(x_{n+1}|\Theta)] \approx \frac{1}{m} \sum_{i=1}^m p(x_{n+1}|\theta_i)$$

# BASIC SAMPLING: AREA UNDER CURVE

Say we are interested in a probability density  $p$  on the interval  $[a, b]$ .



## Key observation

Suppose we can define a uniform distribution  $U_A$  on the blue area  $A$  under the curve.

If we sample

$$(x_1, y_1), (x_2, y_2), \dots \sim_{\text{iid}} U_A$$

and discard the vertical coordinates  $y_i$ , **the  $x_i$  are distributed according to  $p$ ,**

$$x_1, x_2, \dots \sim_{\text{iid}} p .$$

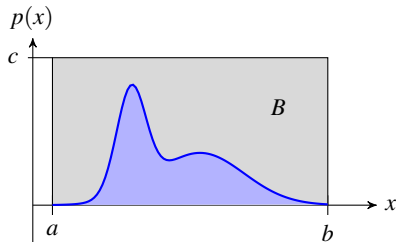
**Problem:** Defining a uniform distribution is easy on a rectangular area, but difficult on an arbitrarily shaped one.



# REJECTION SAMPLING ON THE INTERVAL

## Solution: Rejection sampling

We can enclose  $p$  in box, and sample uniformly from the box  $B$ .



- ▶ We can sample  $(x_i, y_i)$  uniformly on  $B$  by sampling

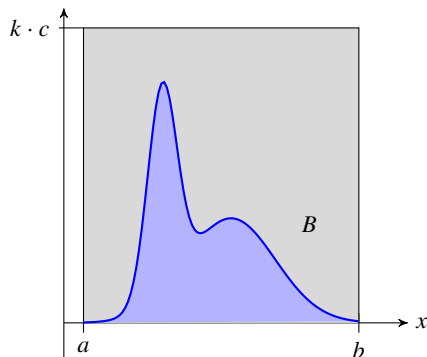
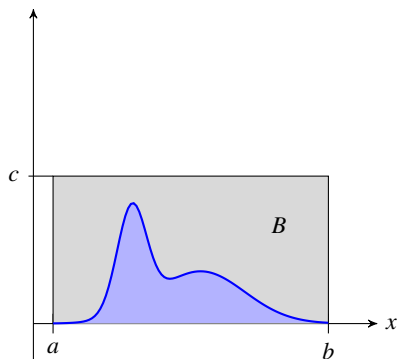
$$x_i \sim \text{Uniform}[a, b] \quad \text{and} \quad y_i \sim \text{Uniform}[0, c] .$$

- ▶ If  $(x_i, y_i) \in A$  (that is: if  $y_i \leq p(x_i)$ ), keep the sample.  
Otherwise: discard it ("reject" it).

**Result:** The remaining (non-rejected) samples are uniformly distributed on  $A$ .

# SCALING

This strategy still works if we scale the vertically by some constant  $k > 0$ :



We simply sample  $y_i \sim \text{Uniform}[0, kc]$  instead of  $y_i \sim \text{Uniform}[0, c]$ .

## Consequence

For sampling, it is sufficient if  $p$  is known only up to normalization (i.e. if only the shape of  $p$  is known).

# DISTRIBUTIONS KNOWN UP TO SCALING

Sampling methods usually assume that we can evaluate the target distribution  $p$  up to a constant. That is:

$$p(x) = \frac{1}{\tilde{Z}} \tilde{p}(x) ,$$

and we can compute  $\tilde{p}(x)$  for any given  $x$ , but we do not know  $\tilde{Z}$ .

We have to pause for a moment and convince ourselves that there are useful examples where this assumption holds.

## Example 1: Simple posterior

For an arbitrary posterior computed with Bayes' theorem, we could write

$$\Pi(\theta|x_{1:n}) = \frac{\prod_{i=1}^n p(x_i|\theta)q(\theta)}{\tilde{Z}} \quad \text{with} \quad \tilde{Z} = \int_{\mathcal{T}} \prod_{i=1}^n p(x_i|\theta)q(\theta)d\theta .$$

Provided that we can compute the numerator, we can sample without computing the normalization integral  $\tilde{Z}$ .

## Example 2: Bayesian Mixture Model

Recall that the posterior of the BMM is (up to normalization):

$$\Pi(c_{1:K}, \theta_{1:K} | x_{1:n}) \propto \prod_{i=1}^n \left( \sum_{k=1}^K c_k p(x_i | \theta_k) \right) \left( \prod_{k=1}^K q(\theta_k | \lambda, y) \right) q_{\text{Dirichlet}}(c_{1:K})$$

We already know that we can discard the normalization constant, but can we evaluate the non-normalized posterior  $\tilde{\Pi}$ ?

- ▶ The problem with computing  $\tilde{\Pi}$  (as a function of unknowns) is that the term  $\prod_{i=1}^n \left( \sum_{k=1}^K \dots \right)$  blows up into  $K^n$  individual terms.
- ▶ If we *evaluate*  $\tilde{\Pi}$  for specific values of  $c$ ,  $x$  and  $\theta$ ,  $\sum_{k=1}^K c_k p(x_i | \theta_k)$  collapses to a single number for each  $x_i$ , and we just have to multiply those  $n$  numbers.

So: Computing  $\tilde{\Pi}$  as a formula in terms of unknowns is difficult; evaluating it for specific values of the arguments is easy.

# DISTRIBUTIONS KNOWN UP TO SCALING

## Example 3: Markov random field

In a MRF, the normalization function is the real problem.

For example, recall the Ising model:

$$p(\theta_{1:n}) = \frac{1}{Z(\beta)} \exp\left(\sum_{(i,j) \text{ is an edge}} \beta \mathbb{I}\{\theta_i = \theta_j\}\right)$$

The normalization function is

$$Z(\beta) = \sum_{\theta_{1:n} \in \{0,1\}^n} \exp\left(\sum_{(i,j) \text{ is an edge}} \beta \mathbb{I}\{\theta_i = \theta_j\}\right)$$

and hence a sum over  $2^n$  terms. The general Potts model is even more difficult.

On the other hand, evaluating

$$\tilde{p}(\theta_{1:n}) = \exp\left(\sum_{(i,j) \text{ is an edge}} \beta \mathbb{I}\{\theta_i = \theta_j\}\right)$$

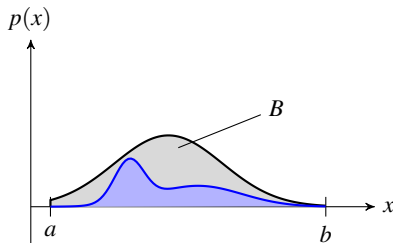
for a given configuration  $\theta_{1:n}$  is straightforward.

# REJECTION SAMPLING ON $\mathbb{R}^d$

If we are not on the interval, sampling uniformly from an enclosing box is not possible (since there is no uniform distribution on all of  $\mathbb{R}$  or  $\mathbb{R}^d$ ).

## Solution: Proposal density

Instead of a box, we use *another distribution*  $q$  to enclose  $p$ :

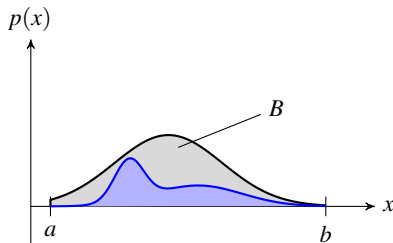


To generate  $B$  under  $q$ , we apply similar logic as before backwards:

- ▶ Sample  $x_i \sim q$ .
- ▶ Sample  $y_i \sim \text{Uniform}[0, q(x_i)]$ .

$q$  is always a simple distribution which we can sample and evaluate.

# REJECTION SAMPLING ON $\mathbb{R}^d$



- ▶ Choose a simple distribution  $q$  from which we know how to sample.
- ▶ Scale  $\tilde{p}$  such that  $\tilde{p}(x) < q(x)$  everywhere.
- ▶ Sampling: For  $i = 1, 2, \dots$ :
  1. Sample  $x_i \sim q$ .
  2. Sample  $y_i \sim \text{Uniform}[0, q(x_i)]$ .
  3. If  $y_i < \tilde{p}(x_i)$ , keep  $x_i$ .
  4. Else, discard  $x_i$  and start again at (1).
- ▶ The surviving samples  $x_1, x_2, \dots$  are distributed according to  $p$ .

# FACTORIZATION PERSPECTIVE

The rejection step can be interpreted in terms of probabilities and densities.

## Factorization

We factorize the target distribution or density  $p$  as

$$p(x) = q(x) \cdot A(x)$$

↓ distribution from which we  
know how to sample

↑ probability function we can evaluate  
*once a specific value of  $x$  is given*

## Sampling from the factorization

sampling  $x$  from  $p$  = sampling  $x$  from  $q$  + coin flip with probability  $A(x)$

By "coin flip", we mean a binary variable with  $\Pr(1) = A(x)$  (ie a Bernoulli variable).

## Sampling Bernoulli variables with uniform variables

$$Z \sim \text{Bernoulli}(A(x)) \quad \Leftrightarrow \quad Z = \mathbb{I}\{U < A(x)\} \quad \text{where} \quad U \sim \text{Uniform}[0, 1].$$



If we draw proposal samples  $x_i$  i.i.d. from  $q$ , the resulting sequence of accepted samples produced by rejection sampling is again i.i.d. with distribution  $p$ . Hence:

Rejection samplers produce i.i.d. sequences of samples.

## Important consequence

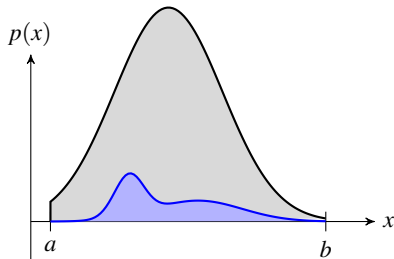
If samples  $x_1, x_2, \dots$  are drawn by a rejection sampler, the sample average

$$\frac{1}{m} \sum_{i=1}^m f(x_i)$$

(for some function  $f$ ) is an unbiased estimate of the expectation  $\mathbb{E}_p[f(X)]$ .

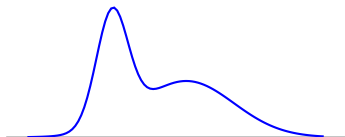
# EFFICIENCY

The fraction of accepted samples is the ratio  $\frac{|A|}{|B|}$  of the areas under the curves  $\tilde{p}$  and  $q$ .

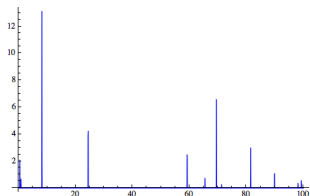


If  $q$  is not a reasonably close approximation of  $p$ , we will end up rejecting a lot of proposal samples.

# AN IMPORTANT BIT OF IMPRECISE INTUITION



Example figures for sampling methods tend to look like this.



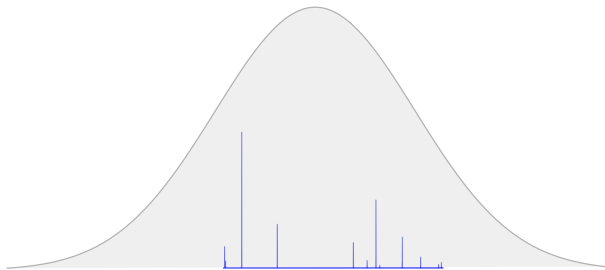
A high-dimensional distribution of correlated RVs will look rather more like this.

Sampling is usually used in multiple dimensions. Reason, roughly speaking:

- ▶ Intractable posterior distributions arise when there are several *interacting* random variables. The interactions make the joint distribution complicated.
- ▶ In one-dimensional problems (1 RV), we can usually compute the posterior analytically.
- ▶ Independent multi-dimensional distributions factorize and reduce to one-dimensional case.

**Warning:** Never (!!!) use sampling if you can solve analytically.

# WHY IS NOT EVERY SAMPLER A REJECTION SAMPLER?



We can easily end up in situations where we accept only one in  $10^6$  (or  $10^{10}$ , or  $10^{20}$ , ...) proposal samples. Especially in higher dimensions, we have to expect this to be not the exception but the rule.

# THE GIBBS SAMPLER

# GIBBS SAMPLING

By far the most widely used MCMC algorithm is the Gibbs sampler.

## Full conditionals

Suppose  $p$  is a distribution on  $\mathbb{R}^D$ , so  $x = (x_1, \dots, x_D)$ . The conditional probability of the entry  $x_d$  given all other entries,

$$p(x_d | x_1, \dots, x_{d-1}, x_{d+1}, \dots, x_D)$$

is called the **full conditional** distribution of  $x_D$ .

## Gibbs sampling

The Gibbs sampler is a special case of the Metropolis-Hastings algorithm which uses the full conditionals to generate proposals.

- ▶ Gibbs sampling is only applicable if we can compute the full conditionals for each dimension  $d$ .
- ▶ If so, it provides us with a *generic* way to derive a proposal distribution.

## Proposal distribution

Suppose  $p$  is a distribution on  $\mathbb{R}^D$ , so each sample is of the form  $x_i = (x_{i,1}, \dots, x_{i,D})$ . We generate a proposal  $x_{i+1}$  coordinate by coordinate as follows:

$$\begin{aligned}x_{i+1,1} &\sim p(\cdot | x_{i,2}, \dots, x_{i,D}) \\ &\vdots \\ x_{i+1,d} &\sim p(\cdot | x_{i+1,1}, \dots, x_{i+1,d-1}, x_{i,d+1}, \dots, x_{i,D}) \\ &\vdots \\ x_{i+1,D} &\sim p(\cdot | x_{i+1,1}, \dots, x_{i+1,D-1})\end{aligned}$$

Note: Each new  $x_{i+1,d}$  is immediately used in the update of the next dimension  $d + 1$ .

A Metropolis-Hastings algorithms with proposals generated as above is called a **Gibbs sampler**.

## No rejections

It is straightforward to show that the Metropolis-Hastings acceptance probability for each  $x_{i+1,d+1}$  is 1, so *proposals in Gibbs sampling are always accepted*.

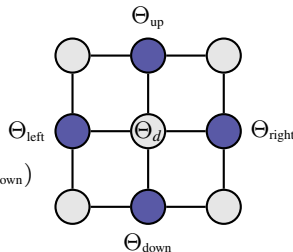
# EXAMPLE: MRF

In a MRF with  $D$  nodes, each dimension  $d$  corresponds to one vertex.

## Full conditionals

In a grid with 4-neighborhoods for instance, the Markov property implies that

$$p(\theta_d | \theta_1, \dots, \theta_{d-1}, \theta_{d+1}, \dots, \theta_D) = p(\theta_d | \theta_{\text{left}}, \theta_{\text{right}}, \theta_{\text{up}}, \theta_{\text{down}})$$



## Specifically: Potts model with binary weights

Recall that, for sampling, knowing only  $\tilde{p}$  (unnormalized) is sufficient:

$$\begin{aligned} \tilde{p}(\theta_d | \theta_1, \dots, \theta_{d-1}, \theta_{d+1}, \dots, \theta_D) = \\ \exp\left(\beta(\mathbb{I}\{\theta_d = \theta_{\text{left}}\} + \mathbb{I}\{\theta_d = \theta_{\text{right}}\} + \mathbb{I}\{\theta_d = \theta_{\text{up}}\} + \mathbb{I}\{\theta_d = \theta_{\text{down}}\})\right) \end{aligned}$$

This is clearly very efficiently computable.



# EXAMPLE: MRF

## Sampling the Potts model

Each step of the sampler generates a sample

$$\theta_i = (\theta_{i,1}, \dots, \theta_{i,D}),$$

where  $D$  is the number of vertices in the grid.

## Gibbs sampler

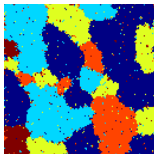
Each step of the Gibbs sampler generates  $n$  updates according to

$$\begin{aligned} \theta_{i+1,1} &\sim p(\cdot | \theta_{i+1,1}, \dots, \theta_{i+1,d-1}, \theta_{i,d+1}, \dots, \theta_{i,D}) \\ &\propto \exp\left(\beta(\mathbb{I}\{\theta_{i+1,d} = \theta_{\text{left}}\} + \mathbb{I}\{\theta_{i+1,d} = \theta_{\text{right}}\} + \mathbb{I}\{\theta_{i+1,d} = \theta_{\text{up}}\} + \mathbb{I}\{\theta_{i+1,d} = \theta_{\text{down}}\})\right) \end{aligned}$$

# EXAMPLE: BURN-IN MATTERS

This example is due to Erik Sudderth (Brown University).

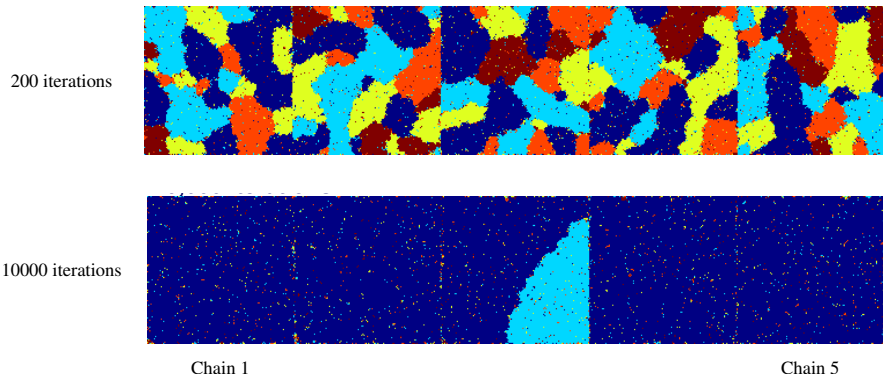
## MRFs as "segmentation" priors



- ▶ MRFs were introduced as tools for image smoothing and segmentation by D. and S. Geman in 1984.
- ▶ They sampled from a Potts model with a Gibbs sampler, discarding 200 iterations as burn-in.
- ▶ Such a sample (after 200 steps) is shown above, for a Potts model in which each variable can take one out of 5 possible values.
- ▶ These patterns led computer vision researchers to conclude that MRFs are "natural" priors for image segmentation, since samples from the MRF resemble a segmented image.

# EXAMPLE: BURN-IN MATTERS

E. Sudderth ran a Gibbs sampler on the same model and sampled after 200 iterations (as the Geman brothers did), and again after 10000 iterations:

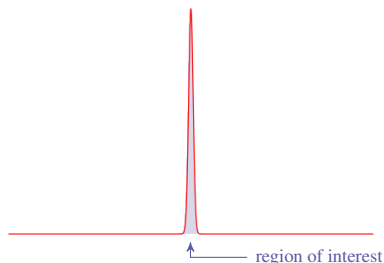


- ▶ The "segmentation" patterns are not sampled from the MRF distribution  $p \equiv$ , but rather from  $P_{200} \neq$ .
- ▶ The patterns occur not because MRFs are "natural" priors for segmentations, but because *the sampler's Markov chain has not mixed*.
- ▶ MRFs are smoothness priors, not segmentation priors.

# MARKOV CHAIN MONTE CARLO

# MOTIVATION

Suppose we rejection-sample a distribution like this:



Once we have drawn a sample in the narrow region of interest, we would like to continue drawing samples within the same region. That is only possible if each sample *depends on the location of the previous sample*.

Proposals in rejection sampling are i.i.d. Hence, once we have found the region where  $p$  concentrates, we forget about it for the next sample.

## Recall: Markov chain

- ▶ A sufficiently nice Markov chain (MC) has an invariant distribution  $P_{\text{inv}}$ .
- ▶ Once the MC has converged to  $P_{\text{inv}}$ , each sample  $x_i$  from the chain has marginal distribution  $P_{\text{inv}}$ .

## Markov chain Monte Carlo

We want to sample from a distribution with density  $p$ . Suppose we can define a MC with invariant distribution  $P_{\text{inv}} \equiv p$ . If we sample  $x_1, x_2, \dots$  from the chain, then once it has converged, we obtain samples

$$x_i \sim p .$$

This sampling technique is called **Markov chain Monte Carlo (MCMC)**.

**Note:** For a Markov chain,  $x_{i+1}$  can depend on  $x_i$ , so at least in principle, it is possible for an MCMC sampler to "remember" the previous step and remain in a high-probability location.

# CONTINUOUS MARKOV CHAIN

The Markov chains we discussed so far had a finite state space  $\mathbf{X}$ . For MCMC, state space now has to be the domain of  $p$ , so we often need to work with continuous state spaces.

## Continuous Markov chain

A continuous Markov chain is defined by an initial distribution  $P_{\text{init}}$  and conditional probability  $t(y|x)$ , the **transition probability** or **transition kernel**.

In the discrete case,  $t(y = i|x = j)$  is the entry  $\mathbf{p}_{ij}$  of the transition matrix  $\mathbf{p}$ .

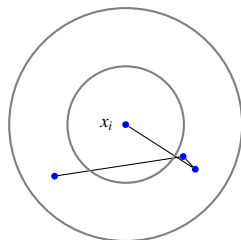
## Example: A Markov chain on $\mathbb{R}^2$

We can define a very simple Markov chain by sampling

$$x_{i+1} \sim g(\cdot | x_i, \sigma^2)$$

where  $g(x|\mu, \sigma^2)$  is a spherical Gaussian with fixed variance. In other words, the transition distribution is

$$t(x_{i+1}|x_i) := g(x_{i+1}|x_i, \sigma^2).$$



A Gaussian (gray contours) is placed around the current point  $x_i$  to sample  $x_{i+1}$ .

# INVARIANT DISTRIBUTION

## Recall: Finite case

- ▶ The invariant distribution  $P_{\text{inv}}$  is a distribution on the finite state space  $\mathbf{X}$  of the MC (i.e. a vector of length  $|\mathbf{X}|$ ).
- ▶ "Invariant" means that, if  $x_i$  is distributed according to  $P_{\text{inv}}$ , and we execute a step  $x_{i+1} \sim t(\cdot | x_i)$  of the chain, then  $x_{i+1}$  again has distribution  $P_{\text{inv}}$ .
- ▶ In terms of the transition matrix  $\mathbf{p}$ :

$$\mathbf{p} \cdot P_{\text{inv}} = P_{\text{inv}}$$

## Continuous case

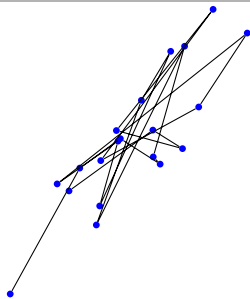
- ▶  $\mathbf{X}$  is now uncountable (e.g.  $\mathbf{X} = \mathbb{R}^d$ ).
- ▶ The transition matrix  $\mathbf{p}$  is substituted by the conditional probability  $t$ .
- ▶ A distribution  $P_{\text{inv}}$  with density  $p_{\text{inv}}$  is invariant if

$$\int_{\mathbf{X}} t(y|x)p_{\text{inv}}(x)dx = p_{\text{inv}}(y)$$

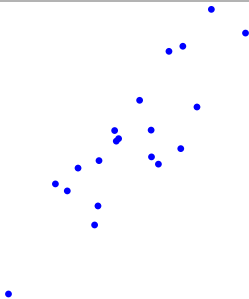
This is simply the continuous analogue of the equation  $\sum_i \mathbf{p}_{ij}(P_{\text{inv}})_i = (P_{\text{inv}})_j$ .



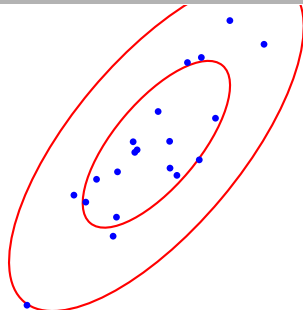
# MARKOV CHAIN SAMPLING



We run the Markov chain  $n$  for steps. Each step moves from the current location  $x_i$  to a new  $x_{i+1}$ .



We "forget" the order and regard the locations  $x_{1:n}$  as a random set of points.



If  $p$  (red contours) is both the invariant and initial distribution, each  $x_i$  is distributed as  $x_i \sim p$ .

## Problems we need to solve

1. We have to construct a MC with invariant distribution  $p$ .
2. We cannot actually start sampling with  $x_1 \sim p$ ; if we knew how to sample from  $p$ , all of this would be pointless.
3. Each point  $x_i$  is *marginally* distributed as  $x_i \sim p$ , but the points are *not* i.i.d.

# CONSTRUCTING THE MARKOV CHAIN

Given is a continuous target distribution with density  $p$ .

## Metropolis-Hastings (MH) kernel

1. We start by defining a conditional probability  $q(y|x)$  on  $\mathbf{X}$ .  
 $q$  has nothing to do with  $p$ . We could e.g. choose  $q(y|x) = g(y|x, \sigma^2)$ , as in the previous example.
2. We define a **rejection kernel**  $A$  as

$$A(x_{n+1}|x_n) := \min\left\{1, \frac{q(x_i|x_{i+1})p(x_{i+1})}{q(x_{i+1}|x_i)p(x_i)}\right\}$$

The normalization of  $p$  cancels in the quotient, so knowing  $\tilde{p}$  is again enough.

total probability that  
a proposal is sampled  
and then rejected

3. We define the transition probability of the chain as

$$t(x_{i+1}|x_i) := q(x_{i+1}|x_i)A(x_{i+1}|x_i) + \delta_{x_i}(x_{i+1})c(x_i) \quad \text{where} \quad c(x_i) := \int q(y|x_i)(1-A(y|x_i))dy$$

## Sampling from the MH chain

At each step  $i + 1$ , generate a proposal  $x^* \sim q(\cdot | x_i)$  and  $U_i \sim \text{Uniform}[0, 1]$ .

- ▶ If  $U_i \leq A(x^* | x_i)$ , accept proposal: Set  $x_{i+1} := x^*$ .
- ▶ If  $U_i > A(x^* | x_i)$ , reject proposal: Set  $x_{i+1} := x_i$ .

# PROBLEM 1: INITIAL DISTRIBUTION

## Recall: Fundamental theorem on Markov chains

Suppose we sample  $x_1 \sim P_{\text{init}}$  and  $x_{i+1} \sim t(\cdot | x_i)$ . This defines a distribution  $P_i$  of  $x_i$ , which can change from step to step. If the MC is nice (recall: recurrent and aperiodic), then

$$P_i \rightarrow P_{\text{inv}} \quad \text{for} \quad i \rightarrow \infty .$$

**Note:** Making precise what aperiodic means in a continuous state space is a bit more technical than in the finite case, but the theorem still holds. We will not worry about the details here.

## Implication

- ▶ If we can show that  $P_{\text{inv}} \equiv p$ , we do not have to know how to sample from  $p$ .
- ▶ Instead, we can start with *any*  $P_{\text{init}}$ , and will get arbitrarily close to  $p$  for sufficiently large  $i$ .

# BURN-IN AND MIXING TIME

The number  $m$  of steps required until  $P_m \approx P_{\text{inv}} \equiv p$  is called the **mixing time** of the Markov chain. (In probability theory, there is a range of definitions for what exactly  $P_m \approx P_{\text{inv}}$  means.)

In MC samplers, the first  $m$  samples are also called the **burn-in** phase. The first  $m$  samples of each run of the sampler are discarded:

$$\underbrace{x_1, \dots, x_{m-1}}_{\text{Burn-in; discard.}} \quad \underbrace{x_m, x_{m+1}, \dots}_{\text{Samples from (approximately) } p; \text{ keep.}}$$

## Convergence diagnostics

In practice, we do not know how large  $j$  is. There are a number of methods for assessing whether the sampler has mixed. Such heuristics are often referred to as **convergence diagnostics**.

## PROBLEM 2: SEQUENTIAL DEPENDENCE

Even after burn-in, the samples from a MC are not i.i.d.

### Strategy

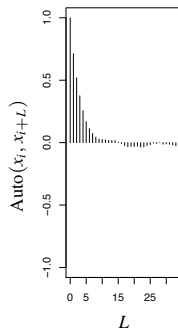
- ▶ Estimate empirically how many steps  $L$  are needed for  $x_i$  and  $x_{i+L}$  to be approximately independent. The number  $L$  is called the **lag**.
- ▶ After burn-in, keep only every  $L$ th sample; discard samples in between.

### Estimating the lag

The most common method uses the **autocorrelation function**:

$$\text{Auto}(x_i, x_j) := \frac{\mathbb{E}[x_i - \mu_i] \cdot \mathbb{E}[x_j - \mu_j]}{\sigma_i \sigma_j}$$

We compute  $\text{Auto}(x_i, x_{i+L})$  empirically from the sample for different values of  $L$ , and find the smallest  $L$  for which the autocorrelation is close to zero.

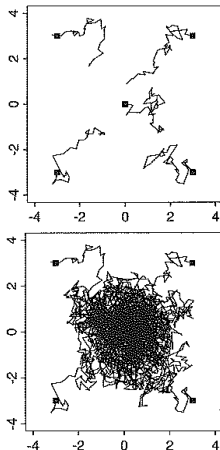


# CONVERGENCE DIAGNOSTICS

There are about half a dozen popular convergence criteria; the one below is an example.

## Gelman-Rubin criterion

- ▶ Start several chains at random. For each chain  $k$ , sample  $x_i^k$  has a marginal distribution  $P_i^k$ .
- ▶ The distributions of  $P_i^k$  will differ between chains in early stages.
- ▶ Once the chains have converged, all  $P_i = P_{\text{inv}}$  are identical.
- ▶ Criterion: Use a hypothesis test to compare  $P_i^k$  for different  $k$  (e.g. compare  $P_i^2$  against null hypothesis  $P_i^1$ ). Once the test does not reject anymore, assume that the chains are past burn-in.



Reference: A. Gelman and D. B. Rubin: "Inference from Iterative Simulation Using Multiple Sequences", *Statistical Science*, Vol. 7 (1992) 457-511.

# STOCHASTIC HILL-CLIMBING

The Metropolis-Hastings rejection kernel was defined as:

$$A(x_{n+1}|x_n) = \min \left\{ 1, \frac{q(x_i|x_{i+1})p(x_{i+1})}{q(x_{i+1}|x_i)p(x_i)} \right\} .$$

Hence, we certainly accept if the second term is larger than 1, i.e. if

$$q(x_i|x_{i+1})p(x_{i+1}) > q(x_{i+1}|x_i)p(x_i) .$$

That means:

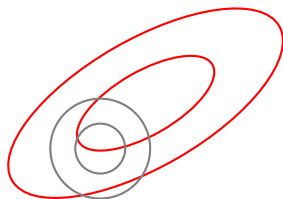
- ▶ We always accept the proposal  $x_{i+1}$  if it *increases* the probability under  $p$ .
- ▶ If it *decreases* the probability, we still accept with a probability which depends on the difference to the current probability.

## Hill-climbing interpretation

- ▶ The MH sampler somewhat resembles a gradient ascent algorithm on  $p$ , which *tends* to move in the direction of increasing probability  $p$ .
- ▶ However:
  - ▶ The actual steps are chosen at random.
  - ▶ The sampler can move "downhill" with a certain probability.
  - ▶ When it reaches a local maximum, it does not get stuck there.

# SELECTING A PROPOSAL DISTRIBUTION

## Everyone's favorite example: Two Gaussians



red = target distribution  $p$   
gray = proposal distribution  $q$

- ▶  $\text{Var}[q]$  too large:  
Will overstep  $p$ ; many rejections.
- ▶  $\text{Var}[q]$  too small:  
Many steps needed to achieve good coverage of domain.

If  $p$  is unimodal and can be roughly approximated by a Gaussian,  $\text{Var}[q]$  should be chosen as smallest covariance component of  $p$ .

## More generally

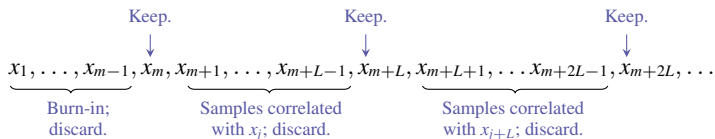
For complicated posteriors (recall: small regions of concentration, large low-probability regions in between) choosing  $q$  is much more difficult. To choose  $q$  with good performance, we already need to know something about the posterior.

There are many strategies, e.g. mixture proposals (with one component for large steps and one for small steps).

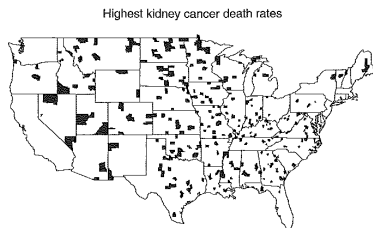


# SUMMARY: MH SAMPLER

- ▶ MCMC samplers construct a MC with invariant distribution  $p$ .
- ▶ The MH kernel is one generic way to construct such a chain from  $p$  and a proposal distribution  $q$ .
- ▶ Formally,  $q$  does not depend on  $p$  (but arbitrary choice of  $q$  usually means bad performance).
- ▶ We have to discard an initial number  $m$  of samples as burn-in to obtain samples (approximately) distributed according to  $p$ .
- ▶ After burn-in, we keep only every  $L$ th sample (where  $L = \text{lag}$ ) to make sure the  $x_i$  are (approximately) independent.



# EXAMPLE



## The data

- ▶ Kidney cancer deaths per county are recorded over a ten-year period.
- ▶  $x_j$  = number of deaths in county  $j$ .
- ▶ Compute empirical annual death rate (per person):

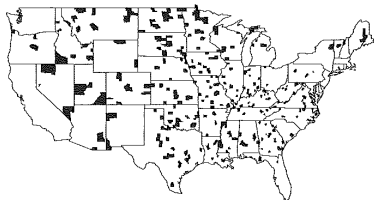
$$\theta_j = \frac{x_j}{10n_j}$$

- ▶ Rank counties by death rate. Top 10% are colored black on map.

[See Gelman et al., "Bayesian Data Analysis]

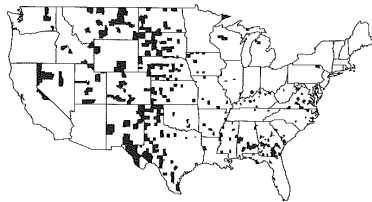
# EXAMPLE

Highest kidney cancer death rates



Top 10%.

Lowest kidney cancer death rates



Bottom 10% © Gelman et al., "Bayesian Data Analysis"

# REGULARIZATION: EXAMPLE

## Baseline

Nationwide, death rate is ca.  $4.7 \cdot 10^{-5}$ .

## Problem: Small sample size effect

For example: County with  $n_j = 1000$  inhabitants.

- ▶ Probably, no death at all in 10 years.
- ▶ If a single death happens to occur, empirical rate is  $1 \cdot \frac{1}{1000 \cdot 10} = 10^{-4}$ .
- ▶ This is more than double the average rate, so country will end up in top 10%.

## Consequence

- ▶ We observe a lot of random effects in counties with small populations.
- ▶ That does not mean the death rate there is not high (or low), just that the raw numbers are not informative either way.

# REGULARIZATION: EXAMPLE

## Approach

- ▶ Model death rate in each county  $j$  separately with parameter  $\theta_j$ .
- ▶ Define a *joint* prior (ie the same distribution  $q$ ) on the parameters for all counties.

## Model

- ▶ Data: Counts of rare events  $\rightarrow$  use Poisson distribution.
- ▶ Generative model:

$$X_j \sim \text{Poisson}(10 \cdot n_j \cdot \Theta_j)$$

- ▶ The maximum likelihood estimator is

$$\hat{\theta}_j^{\text{ML}} = \frac{x_j}{10 \cdot n_j},$$

which is precisely the empirical death rate.

## Bayesian model

- ▶ Use a conjugate prior; the natural conjugate prior of the Poisson is a gamma distribution:

$$q(\theta|\alpha_0, \beta_0) = \theta^{\alpha_0-1} \frac{\beta_0^{\alpha_0} e^{-\beta_0\theta}}{\Gamma(\alpha_0)}$$

- ▶ Posterior:

$$\Pi(\theta|x_j, \alpha_0, \beta_0) = q(\theta|\alpha_0 + x_j, \beta_0 + 10n_j)$$

How do we choose  $\alpha_0$  and  $\beta_0$ ?

## Choice of hyperparameters

Gelman et al: Use data to choose  $(\alpha_0, \beta_0)$  (careful: dangerous!).

- ▶ Look at entire data (whole nation) and compute its mean  $\hat{\mu}_{\text{emp}}$  and variance  $\hat{\sigma}_{\text{emp}}^2$ .
- ▶ The marginal distribution of the data under the model is

$$p(x|\alpha_0, \beta_0) = \int p(x|\theta)q(\theta|\alpha_0, \beta_0)d\theta ,$$

which is a negative binomial distribution with parameters  $\alpha_0$  and  $\beta_0$ .

- ▶ Choose  $\alpha_0$  and  $\beta_0$  such that  $p(x|\alpha_0, \beta_0)$  has mean  $\hat{\mu}_{\text{emp}}$  and variance  $\hat{\sigma}_{\text{emp}}^2$ .

# REGULARIZATION: EXAMPLE

## Prior hyperparameters

Gelman et al. find:

$$\alpha_0 = 20 \quad \text{and} \quad \beta = 430000 .$$

The hyperparameters of the conjugate posterior for county  $j$  are

$$\alpha_j := \alpha_0 + x_j \quad \text{and} \quad \beta_j := \beta_0 + 10n_j .$$

## Recall: Baseline

The national rate is ca.  $4.7 \cdot 10^{-5}$ .

## Effect of regularization

For example, for a small county of size  $n_j = 1000$ : The posterior mean is

$$\mathbb{E}[\Theta_j | x_j] = \frac{\alpha_j}{\beta_j} = \frac{20 + x_j}{430000 + 10n_j} = \frac{20 + x_j}{440000} .$$

Depending on  $x_j$ :

$x_j$ (# of deaths)	MLE	Posterior mean
0	0	$4.55 \cdot 10^{-5}$
1	$1 \cdot 10^{-4}$	$4.77 \cdot 10^{-5}$
2	$2 \cdot 10^{-4}$	$5.00 \cdot 10^{-5}$

# REGULARIZATION: EXAMPLE

## Remarks: Regularization

- ▶ Prior is used here mainly as regularization.
- ▶ We could have regularized in some other (non-Bayesian) way, however:
- ▶ Individual building blocks of model have a well-specified meaning.

## Remarks: Prior choice

The use of the data in choosing the prior raises confounding issues. Here:

- ▶ Data enters only through its aggregate mean and variance.
- ▶ The posteriors interpolates with the national average, i.e. in cases where there is little data we fall back on the national values.

Usually knowledge about the data informs the prior choice in *some* way.

## Advice

Once you have fully defined your model, take half an hour to review very carefully which information has entered in the prior. Make a list and consider implications.



# BAYESIAN MIXTURE MODELS AND ADMIXTURES

## In the following

We will consider two variations:

- ▶ *Bayesian mixture models* (mixtures with priors).
- ▶ *Admixtures*, in which the generation of each observation (e.g. document) can be influenced by several components (e.g. topics).
- ▶ One particular admixture model, called *latent Dirichlet allocation*, is one of the most successful machine learning models of the past ten years.

## Inference: Sampling

These models are examples of models in which the exact posterior is intractable. Inference uses Markov chain Monte Carlo sampling, which will be our main topic for the last two lectures.

## Recall: Finite mixture models

$$\pi(x) = \sum_{k=1}^K c_k p(x|\theta_k) = \int_{\mathcal{T}} p(x|\theta) m(\theta) d\theta \quad \text{with} \quad m := \sum_{k=1}^K c_k \delta_{\theta_k}$$

All parameters are summarized in the *mixing distribution*  $m$ .

## Bayesian mixture model: Idea

In a Bayesian model, parameters are random variables. Here, that means a *random* mixing distribution:

$$M(\cdot) = \sum_{k=1}^K C_k \delta_{\Theta_k}(\cdot)$$

# RANDOM MIXING DISTRIBUTION

## How can we define a random distribution?

Since  $M$  is discrete with finitely many terms, we only have to generate the random variables  $C_k$  and  $\Theta_k$ :

$$M(\cdot) = \sum_{k=1}^K C_k \delta_{\Theta_k}(\cdot)$$

## More precisely

Specifically, the term BMM implies that all priors are natural conjugate priors. That is:

- ▶ The mixture components  $p(x|\theta)$  are an exponential family model.
- ▶ The prior on each  $\Theta_k$  is a natural conjugate prior of  $p$ .
- ▶ The prior of the vector  $(C_1, \dots, C_K)$  is a Dirichlet distribution.

## Explanation: Dirichlet distribution

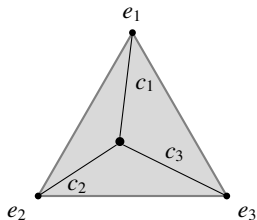
- ▶ When we sample from a finite mixture, we choose a component  $k$  from a multinomial distribution with parameter vector  $(c_1, \dots, c_k)$ .
- ▶ The conjugate prior of the multinomial is the Dirichlet distribution.

# THE DIRICHLET DISTRIBUTION

## Recall: Probability simplex

The set of all probability distributions on  $K$  events is the *simplex*

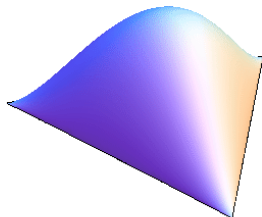
$$\Delta_K := \{(c_1, \dots, c_k) \in \mathbb{R}^K \mid c_k \geq 0 \text{ and } \sum_k c_k = 1\}.$$



## Dirichlet distribution

The **Dirichlet distribution** is the distribution on  $\Delta_K$  with density

$$q_{\text{Dirichlet}}(c_{1:K} \mid \alpha, g_{1:K}) := \frac{1}{K(\alpha, g_{1:K})} \exp\left(\sum_{k=1}^K (\alpha g_k - 1) \log(c_k)\right)$$



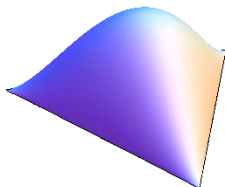
Parameters:

- ▶  $g_{1:K} \in \Delta_K$ : Mean parameter, i.e.  $\mathbb{E}[c_{1:K}] = g_{1:K}$ .
- ▶  $\alpha \in \mathbb{R}_+$ : Concentration.  
Larger  $\alpha \rightarrow$  sharper concentration around  $g_{1:K}$ .

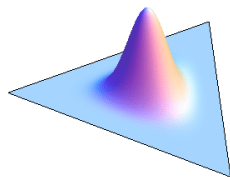
# THE DIRICHLET DISTRIBUTION

In all plots,  $g_{1:K} = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ . Light colors = large density values.

## Density plots

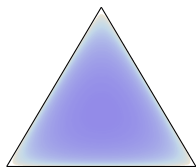


$\alpha = 1.8$



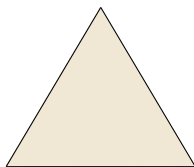
$\alpha = 10$

## As heat maps



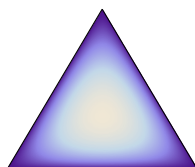
$\alpha = 0.8$

Large density values  
at extreme points



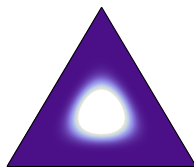
$\alpha = 1$

Uniform distribution  
on  $\Delta_K$



$\alpha = 1.8$

Density peaks  
around its mean



$\alpha = 10$

Peak sharpens  
with increasing  $\alpha$

# MULTINOMIAL-DIRICHLET MODEL

## Model

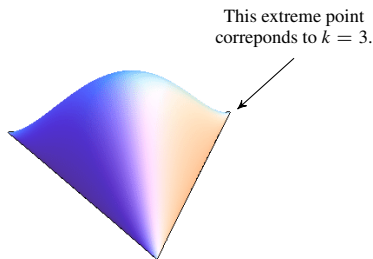
The Dirichlet is the natural conjugate prior on the multinomial parameters. If we observe  $h_k$  counts in category  $k$ , the posterior is

$$\Pi(c_{1:K}|h_1, \dots, h_K) = q_{\text{Dirichlet}}(c_{1:K}|\alpha + n, (\alpha g_1 + h_1, \dots, \alpha g_K + h_K))$$

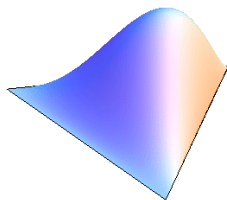
where  $n = \sum_k h_k$  is the total number of observations.

## Illustration: One observation

Suppose  $K = 3$  and we obtain a single observation in category 3.



Prior: Mean at the center.



Posterior: Shifted mean, increased concentration.

## Definition

A model of the form

$$\pi(x) = \sum_{k=1}^K c_k p(x|\theta_k) = \int_{\mathcal{T}} p(x|\theta) M(\theta) d\theta$$

is called a **Bayesian mixture model** if  $p(x|\theta)$  is an exponential family model and  $M$  a random mixing distribution, where:

- ▶  $\Theta_1, \dots, \Theta_k \sim_{\text{iid}} q(\cdot | \lambda, y)$ , where  $q$  is a natural conjugate prior for  $p$ .
- ▶  $(C_1, \dots, C_K)$  is sampled from a  $K$ -dimensional Dirichlet distribution.



# BAYESIAN MIXTURE: INFERENCE

## Posterior distribution

The posterior of a BMM under observations  $x_1, \dots, x_n$  is (up to normalization):

$$\Pi(c_{1:K}, \theta_{1:K} | x_{1:n}) \propto \prod_{i=1}^n \left( \sum_{k=1}^K c_k p(x_i | \theta_k) \right) \left( \prod_{k=1}^K q(\theta_k | \lambda, y) \right) q_{\text{Dirichlet}}(c_{1:K})$$

## The posterior is analytically intractable

- ▶ Thanks to conjugacy, we *can* evaluate each term of the posterior.
- ▶ However: Due to the  $\prod_{k=1}^K \left( \sum_{i=1}^n \dots \right)$  bit, the posterior has  $K^n$  terms!
- ▶ Even for 10 clusters and 100 observations, that is impossible to compute.

## Solution

The posterior can be sampled with a very simple MCMC sampler (which looks strikingly similar to an EM algorithm). We will discuss sampling algorithms in the next lecture.

## Recall: Multinomial text clustering

We assume the corpus is generated by a multinomial mixture model of the form

$$\pi(\mathbf{H}) = \sum_{k=1}^K c_k P(\mathbf{H}|\theta_k) ,$$

where  $P(\mathbf{H}|\theta_k)$  is multinomial.

- ▶ A document is represented by a histogram  $\mathbf{H}$ .
- ▶ Topics  $\theta_1, \dots, \theta_K$ .
- ▶  $\theta_{kj} = \Pr\{\text{word } j \text{ in topic } k\}$ .

## Problem

Each document is generated by a single topic; that is a very restrictive assumption.

## Parameters

Suppose we consider a corpus with  $K$  topics and a vocabulary of  $d$  words.

- ▶  $\phi \in \Delta_K$  topic proportions ( $\phi_k = \Pr\{\text{topic } k\}$ ).
- ▶  $\theta_1, \dots, \theta_K \in \Delta_d$  topic parameter vectors ( $\theta_{kj} = \Pr\{\text{word } j \text{ in topic } k\}$ ).

**Note:** For random generation of documents, we assume that  $\phi$  and the topic parameters  $\theta_k$  are given (they properties of the corpus). To train the model, they have to be learned from data.

## Model 1: Multinomial mixture

To sample a document containing  $M$  words:

1. Sample topic  $k \sim \text{Multinomial}(\phi)$ .
2. For  $i = 1, \dots, M$ : Sample word  $i \sim \text{Multinomial}(\theta_k)$ .

The entire document is sample from topic  $k$ .

## Mixtures of topics

Whether we sample words or entire documents makes a big difference.

- ▶ When we sample from the multinomial mixture, we choose a topic at random, then sample the *entire* document from that topic.
- ▶ For several topics to be represented in the document, we have to sample each word individually (i.e. choose a new topic for each word).
- ▶ Problem: If we do that in the mixture above, every document has the same topic proportions.

## Model 2: Admixture model

Each document explained as a *mixture* of topics, with mixture weights  $c_{1:K}$ .

1. Sample topic proportions  $c_{1:K} \sim \text{Dirichlet}(\phi)$ .
2. For  $i = 1, \dots, M$ :
  - 2.1 Sample topic for word  $i$  as  $k_i \sim \text{Multinomial}(c_{1:K})$ .
  - 2.2 Sample word  $i \sim \text{Multinomial}(\theta_k)$ .

This model is known as **Latent Dirichlet Allocation** (LDA).

# COMPARISON: LDA AND BMM

## Observation

LDA is *almost* a Bayesian mixture model: Both use multinomial components and a Dirichlet prior on the mixture weights. However, they are not identical.

## Comparison

Bayesian MM	Admixture (LDA)
Sample $c_{1:K} \sim \text{Dirichlet}(\phi)$ .	Sample $c_{1:K} \sim \text{Dirichlet}(\phi)$ .
Sample topic $k \sim \text{Multinomial}(c_{1:K})$ .	For $i = 1, \dots, M$ :
For $i = 1, \dots, M$ :	Sample topic $k_i \sim \text{Multinomial}(c_{1:K})$ .
Sample word $w_i \sim \text{Multinomial}(\theta_k)$ .	Sample word $w_i \sim \text{Multinomial}(\theta_{k_i})$ .

In admixtures:

- ▶  $c_{1:K}$  is generated at random, *once for each document*.
- ▶ Each word is sampled from its own topic.

## What do we learn in LDA?

LDA explains each document by a separate parameter  $c_{1:K} \in \Delta_K$ . That is, LDA models documents as *topic proportions*.

# EXAMPLE: MIXTURE OF TOPICS

“Arts”	“Budgets”	“Children”	“Education”
NEW	MILLION	CHILDREN	SCHOOL
FILM	TAX	WOMEN	STUDENTS
SHOW	PROGRAM	PEOPLE	SCHOOLS
MUSIC	BUDGET	CHILD	EDUCATION
MOVIE	BILLION	YEARS	TEACHERS
PLAY	FEDERAL	FAMILIES	HIGH
MUSICAL	YEAR	WORK	PUBLIC
BEST	SPENDING	PARENTS	TEACHER
ACTOR	NEW	SAYS	BENNETT
FIRST	STATE	FAMILY	MANIGAT
YORK	PLAN	WELFARE	NAMPHY
OPERA	MONEY	MEN	STATE
THEATER	PROGRAMS	PERCENT	PRESIDENT
ACTRESS	GOVERNMENT	CARE	ELEMENTARY
LOVE	CONGRESS	LIFE	HAITI

The William Randolph Hearst Foundation will give \$1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. “Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services,” Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center’s share will be \$200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive \$400,000 each. The Juilliard School, where music and the performing arts are taught, will get \$250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual \$100,000 donation, too.