Computer Vision Group
Prof. Daniel Cremers

Technische Universität München

Machine Learning for Computer Vision

Dr. Rudolph Triebel

Lecturers

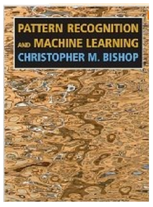
- Dr. Rudolph Triebel
- rudolph.triebel@in.tum.de
- Room number 02.09.059
- Main lecture

- Dipl. Inf. Jan Stühmer
- jan.stuehmer@in.tum.de
- Room number 02.09.059
- Assistance and exercises

Class Schedule

Date	Topic
25.10.13	Introduction
8.11.13	Regression
15.11.13	Probabilistic Graphical Models I
22.11.13	Probabilistic Graphical Models II
29.11.13	Boosting
6.12.13	Kernel Methods
13.12.13	Gaussian Processes
20.12.13	Mixture Models and EM
10.1.14	Variational Inference
17.1.14	Sampling Methods
24.1.14	MCMC
31.1.14	Unsupervised Learning
7.2.14	Online Learning

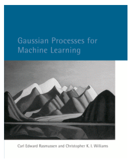
Literature

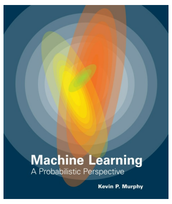


Recommended textbook for the lecture: Christopher M. Bishop: “Pattern Recognition and Machine Learning”

More detailed:

- “Gaussian Processes for Machine Learning” Rasmussen/Williams
- “Machine Learning - A Probabilistic Perspective” Murphy





The Tutorials

- Weekly tutorial classes
- Participation in tutorial classes and submission of solved assignment sheets is totally free
- The submitted solutions can be corrected and returned
- In class, you have the opportunity to present your solution
- Assignments will be theoretical and practical problems

The Exam

- No “qualification” necessary for the final exam
- Final exam will be oral
- From a given number of known questions, some will be drawn by chance
- Usually, from each part a fixed number of questions appears

Class Webpage

http://vision.in.tum.de/teaching/ws2013/ml_ws13

- Contains the slides and assignments for download
- Also used for communication, in addition to email list
- Some further material will be developed in class

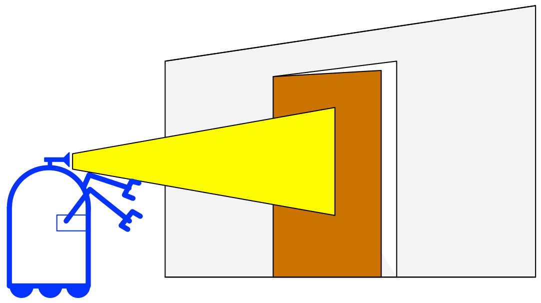
Computer Vision Group
Prof. Daniel Cremers

Technische Universität München

1. Introduction to Learning and Probabilistic Reasoning

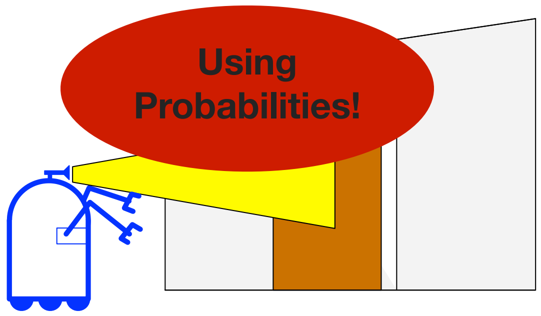
Motivation

Suppose a robot stops in front of a door. It has a sensor (e.g. a camera) to measure the state of the door (open or closed). **Problem**: the sensor may fail.



Motivation

Question: How can we obtain knowledge about the environment from sensors that may return incorrect results?



Basics of Probability Theory

Definition 1.1: A *sample space* $\mathcal{S}$ is a set of outcomes of a given experiment.

Examples:

a) Coin toss experiment: $\mathcal{S} = \{H, T\}$

b) Distance measurement: $\mathcal{S} = \mathbb{R}_0^+$

Definition 1.2: A *random variable* X is a function that assigns a real number to each element of $\mathcal{S}$.

Example: Coin toss experiment: $H = 1, T = 0$

Values of random variables are denoted with small letters, e.g.: $X = x$

Discrete and Continuous

If $\mathcal{S}$ is countable then X is a *discrete* random variable, else it is a *continuous* random variable.

The probability that X takes on a certain value x is a real number between 0 and 1. It holds:

$$\sum_x p(X = x) = 1 \qquad \int p(X = x)dx = 1$$

Discrete case Continuous case

A Discrete Random Variable

Suppose a robot knows that it is in a room, but it does not know in *which* room. There are 4 possibilities:

Kitchen, Office, Bathroom, Living room

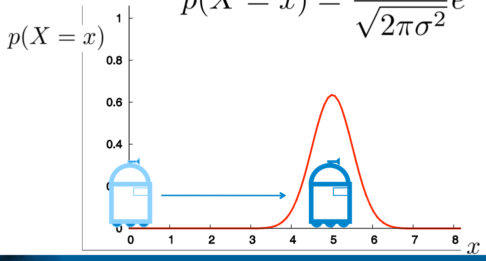
Then the random variable *Room* is discrete, because it can take on one of four values. The probabilities are, for example:

$$P(Room = \text{kitchen}) = 0.7$$
$$P(Room = \text{office}) = 0.2$$
$$P(Room = \text{bathroom}) = 0.08$$
$$P(Room = \text{living room}) = 0.02$$

A Continuous Random Variable

Suppose a robot travels 5 meters forward from a given start point. Its position X is a continuous random variable with a *Normal distribution*:

$$p(X = x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-5)^2}{\sigma^2}}$$



Shorthand:

$$\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}$$
$$\downarrow$$
$$\mathcal{N}(x; \mu, \sigma^2)$$

Joint and Conditional Probability

The *joint probability* of two random variables X and Y is the probability that the events $X = x$ and $Y = y$ occur at the same time:

$$p(X = x \text{ and } Y = y)$$

Shorthand: $p(X = x) \longrightarrow p(x)$

$p(X = x \text{ and } Y = y) \longrightarrow p(x, y)$

Definition 1.3: The *conditional probability* of X given is defi Yd as:

$$p(X = x \mid Y = y) = p(x \mid y) := \frac{p(x, y)}{p(y)}$$

Independency, Sum and Product Rule

Definition 1.4: Two random variables X and Y are *independent* iff:

$$p(x, y) = p(x)p(y)$$

For independent random variables Xnd Ye have:

$$p(x \mid y) = \frac{p(x, y)}{p(y)} = \frac{p(x)p(y)}{p(y)} = p(x)$$

Furthermore, it holds:

$$p(x) = \sum_y p(x, y) \qquad p(x, y) = p(y \mid x)p(x)$$

"Sum Rule" "Product Rule"

Law of Total Probability

Theorem 1.1: For two random variables X and Y it holds:

$$p(x) = \sum_y p(x \mid y)p(y) \qquad p(x) = \int p(x \mid y)p(y)dy$$

Discrete case — Continuous case

The process of obtaining $p(x)$ from $p(x,y)$ by summing or integrating over all values of y is called **Marginalisation**

Bayes Rule

Theorem 1.2: For two random variables X and Y it holds:

$$p(x \mid y) = \frac{p(y \mid x)p(x)}{p(y)}$$ **"Bayes Rule"**

Proof:

I. $p(x \mid y) = \frac{p(x,y)}{p(y)}$ *(definition)*

II. $p(y \mid x) = \frac{p(x,y)}{p(x)}$ *(definition)*

III. $p(x,y) = p(y \mid x)p(x)$ *(from II.)*

Bayes Rule: Background Knowledge

For $p(y \mid z) \neq 0$ it holds:

Background knowledge

$$p(x \mid y, z) = \frac{p(y \mid x, z)p(x \mid z)}{p(y \mid z)}$$

Shorthand: $p(y \mid z)^{-1} \longrightarrow \eta$

"Normalizer"

$$p(x \mid y, z) = \eta\, p(y \mid x, z)p(x \mid z)$$

Computing the Normalizer

$$p(x \mid y) = \frac{p(y \mid x)p(x)}{p(y)}$$

Bayes rule

$$p(y) = \sum_x p(y \mid x)p(x)$$

Total probability

$$p(x \mid y) = \frac{p(y \mid x)p(x)}{\sum_{x'} p(y \mid x')p(x')}$$

$p(x \mid y)$ can be computed without knowing $p(y)$

Conditional Independence

Definition 1.5: Two random variables X and Y are ***conditional independent*** given a third random variable Z iff:

$$p(x, y \mid z) = p(x \mid z)p(y \mid z)$$

This is equivalent to:

$$p(x \mid z) = p(x \mid y, z) \quad \text{and}$$
$$p(y \mid z) = p(y \mid x, z)$$

Expectation and Covariance

Definition 1.6: The ***expectation*** of a random variable X is defined as:

$$E[X] = \sum_x x\, p(x) \qquad \text{(discrete case)}$$

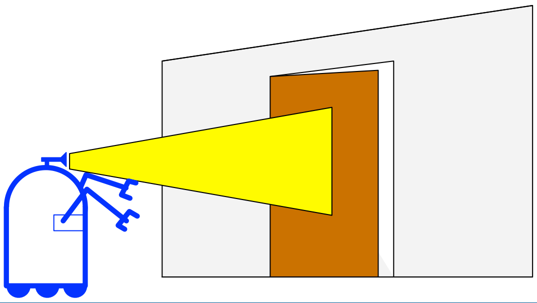
$$E[X] = \int x\, p(x)dx \qquad \text{(continuous case)}$$

Definition 1.7: The ***covariance*** of a random variable X is defined as:

$$Cov[X] = E[(X - E[X])^2] = E[X^2] - E[X]^2$$

Mathematical Formulation of Our Example

We define two binary random variables: z and open, where z is "light on" or "light off". Our question is: What is $p(\text{open} \mid z)$?



Causal vs. Diagnostic Reasoning

- Searching for $p(\text{open} \mid z)$ is called ***diagnostic reasoning***
- Searching for $p(z \mid \text{open})$ is called ***causal reasoning***
- Often causal knowledge is easier to obtain
- Bayes rule allows us to use causal knowledge:

$$\begin{aligned} p(\text{open} \mid z) &= \frac{p(z \mid \text{open})p(\text{open})}{p(z)} \\ &= \frac{p(z \mid \text{open})p(\text{open})}{p(z \mid \text{open})p(\text{open}) + p(z \mid \neg\text{open})p(\neg\text{open})} \end{aligned}$$

Example with Numbers

Assume we have this *sensor model:*

$$p(z \mid \text{open}) = 0.6 \qquad p(z \mid \neg\text{open}) = 0.3$$

and: $p(\text{open}) = p(\neg\text{open}) = 0.5$ *"Prior prob."*

then:

$$p(\text{open} \mid z) = \frac{p(z \mid \text{open})p(\text{open})}{p(z \mid \text{open})p(\text{open}) + p(z \mid \neg\text{open})p(\neg\text{open})}$$
$$= \frac{0.6 \cdot 0.5}{0.6 \cdot 0.5 + 0.3 \cdot 0.5} = \frac{2}{3} = 0.67$$

"z raises the probability that the door is open"

Combining Evidence

Suppose our robot obtains another observation z_2, where the index is the point in time.

Question: How can we integrate this new information?

Formally, we want to estimate $p(\text{open} \mid z_1, z_2)$. Using Bayes formula with background knowledge:

$$p(\text{open} \mid z_1, z_2) = \frac{\overset{?}{}\ p(z_2 \mid \text{open}, z_1)\ \overset{?}{} p(\text{open} \mid z_1)}{p(z_2 \mid z_1)}$$

Markov Assumption

"If we know the state of the door at time $t = 1$ then the measurement z_1 does not give any further information about z_2."

Formally: "z_1 and z_2 are conditional independent given open." This means:

$$p(z_2 \mid \text{open}, z_1) = p(z_2 \mid \text{open})$$

This is called the *Markov Assumption.*

Example with Numbers

Assume we have a second sensor:

$$p(z_2 \mid \text{open}) = 0.5 \qquad p(z_2 \mid \neg\text{open}) = 0.6$$
$$p(\text{open} \mid z_1) = \tfrac{2}{3} \text{ (from above)}$$

Then: $p(\text{open} \mid z_1, z_2) =$

$$\frac{p(z_2 \mid \text{open})p(\text{open} \mid z_1)}{p(z_2 \mid \text{open})p(\text{open} \mid z_1) + p(z_2 \mid \neg\text{open})p(\neg\text{open} \mid z_1)}$$
$$= \frac{\frac{1}{2} \cdot \frac{2}{3}}{\frac{1}{2} \cdot \frac{2}{3} + \frac{3}{5} \cdot \frac{1}{3}} = \frac{5}{8} = 0.625$$

"z_2 lowers the probability that the door is open"

General Form

Measurements: $z_1, \ldots, z_n$

Markov assumption: z_n and $z_1, \ldots, z_{n-1}$ are conditionally independent given the state x.

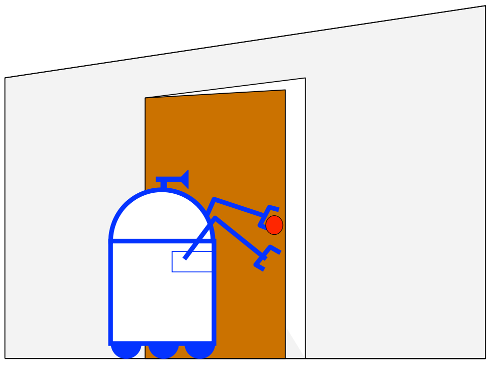
$$p(x \mid z_1, \ldots, z_n) = \frac{p(z_n \mid x)p(x \mid z_1, \ldots, z_{n-1})}{p(z_n \mid z_1, \ldots, z_{n-1})}$$

Recursion

$$\eta_n \ p(z_n \mid x)p(x \mid z_1, \ldots, z_{n-1})$$
$$= \prod_{i=1}^{n} \eta_i \ p(z_i \mid x)p(x)$$

Example: Sensing and Acting

Now the robot *senses* the door state and *acts* (it opens or closes the door).



State Transitions

The *outcome* of an action is modeled as a random variable U where $U = u$ in our case means "state after closing the door".

State transition example:

$$\begin{tikzcd}
\text{open} \arrow[loop left, "0.1"] \arrow[r, bend left, "0.9"] & \text{closed} \arrow[loop right, "1"] \arrow[l, bend left, "0"]
\end{tikzcd}$$

If the door is open, the action "close door" succeeds in 90% of all cases.

The Outcome of Actions

For a given action u we want to know the probability $p(x \mid u)$. We do this by integrating over all possible previous states x'.

If the state space is discrete:

$$p(x \mid u) = \sum_{x'} p(x \mid u, x')p(x')$$

If the state space is continuous:

$$p(x \mid u) = \int p(x \mid u, x')p(x')dx'$$

Back to the Example

$$\begin{aligned} p(\text{open} \mid u) &= \sum_{x'} p(\text{open} \mid u, x') p(x') \\ &= p(\text{open} \mid u, \text{open}') p(\text{open}') + \\ &\quad\; p(\text{open} \mid u, \neg\text{open}') p(\neg\text{open}') \\ &= \frac{1}{10} \cdot \frac{5}{8} + 0 \cdot \frac{3}{8} \\ &= \frac{1}{16} = 0.0625 \end{aligned}$$

$$p(\neg\text{open} \mid u) = 1 - p(\text{open} \mid u) = \frac{15}{16} = 0.9375$$

Sensor Update and Action Update

So far, we learned two different ways to update the system state:

- Sensor update: $p(x \mid z)$
- Action update: $p(x \mid u)$
- Now we want to combine both:

Definition 2.1: Let $D_t = u_1, z_1, \ldots, u_t, z_t$ be a sequence of sensor measurements and actions until time t Then the ***belief*** of the current state x_t is defined as

$$\text{Bel}(x_t) = p(x_t \mid u_1, z_1, \ldots, u_t, z_t)$$

Graphical Representation

We can describe the overall process using a ***Dynamic Bayes Network:***

$$\begin{tikzcd} & u_{t-1} \arrow[dr] & & u_t \arrow[dr] & & u_{t+1} \arrow[dr] & & \\ \cdots \arrow[rr] & & x_{t-1} \arrow[rr] \arrow[d] & & x_t \arrow[rr] \arrow[d] & & x_{t+1} \arrow[r] \arrow[d] & \cdots \\ & & z_{t-1} & & z_t & & z_{t+1} & \end{tikzcd}$$

This incorporates the following Markov assumptions:

$$p(z_t \mid x_{0:t}, u_{1:t}, z_{1:t}) = p(z_t \mid x_t) \quad \text{(measurement)}$$

$$p(x_t \mid x_{0:t-1}, u_{1:t}, z_{1:t}) = p(x_t \mid x_{t-1}, u_t) \quad \text{(state)}$$

The Overall Bayes Filter

$$\begin{aligned} \text{Bel}(x_t) &= p(x_t \mid u_1, z_1, \ldots, u_t, z_t) \\ \text{(Bayes)} \quad &= \eta\, p(z_t \mid x_t, u_1, z_1, \ldots, u_t) p(x_t \mid u_1, z_1, \ldots, u_t) \\ \text{(Markov)} \quad &= \eta\, p(z_t \mid x_t) p(x_t \mid u_1, z_1, \ldots, u_t) \\ \text{(Tot. prob.)} \quad &= \eta\, p(z_t \mid x_t) \int p(x_t \mid u_1, z_1, \ldots, u_t, x_{t-1}) \\ &\qquad p(x_{t-1} \mid u_1, z_1, \ldots, u_t) dx_{t-1} \\ \text{(Markov)} \quad &= \eta\, p(z_t \mid x_t) \int p(x_t \mid u_t, x_{t-1}) p(x_{t-1} \mid u_1, z_1, \ldots, u_t) dx_{t-1} \\ \text{(Markov)} \quad &= \eta\, p(z_t \mid x_t) \int p(x_t \mid u_t, x_{t-1}) p(x_{t-1} \mid u_1, z_1, \ldots, z_{t-1}) dx_{t-1} \\ &= \eta\, p(z_t \mid x_t) \int p(x_t \mid u_t, x_{t-1}) \text{Bel}(x_{t-1}) dx_{t-1} \end{aligned}$$

The Bayes Filter Algorithm

$$\boxed{\text{Bel}(x_t) = \eta\, p(z_t \mid x_t) \int p(x_t \mid u_t, x_{t-1}) \text{Bel}(x_{t-1}) dx_{t-1}}$$

```
Algorithm Bayes_filter $(\text{Bel}(x), d)$
1. if $d$ is a sensor measurement $z$ then
2.     $\eta = 0$
3.     for all $x$ do
4.         $\text{Bel}'(x) \leftarrow p(z \mid x)\text{Bel}(x)$
5.         $\eta \leftarrow \eta + \text{Bel}'(x)$
6.     for all $x$ do $\text{Bel}'(x) \leftarrow \eta^{-1}\text{Bel}'(x)$
7. else if $d$ is an action $u$ then
8.     for all $x$ do $\text{Bel}'(x) \leftarrow \int p(x \mid u, x')\text{Bel}(x')dx'$
9. return $\text{Bel}'(x)$
```

Bayes Filter Variants

$$\boxed{\text{Bel}(x_t) = \eta\, p(z_t \mid x_t) \int p(x_t \mid u_t, x_{t-1}) \text{Bel}(x_{t-1}) dx_{t-1}}$$

The Bayes filter principle is used in

- Kalman filters
- Particle filters
- Hidden Markov models
- Dynamic Bayesian networks
- Partially Observable Markov Decision Processes (POMDPs)

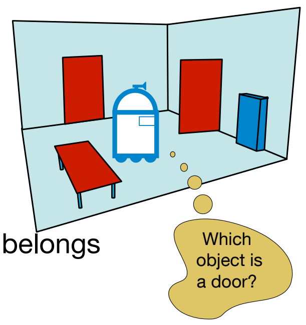
Summary

- ***Probabilistic reasoning*** is necessary to deal with uncertain information, e.g. sensor measurements
- Using ***Bayes rule***, we can do diagnostic reasoning based on causal knowledge
- The outcome of a robot‘s action can be described by a ***state transition diagram***
- Probabilistic state estimation can be done recursively using the ***Bayes filter*** using a sensor and a motion update
- A graphical representation for the state estimation problem is the ***Dynamic Bayes Network***

2. Introduction to Learning

Motivation

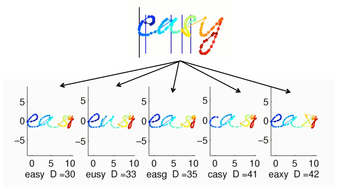
- Most objects in the environment can be classified, e.g. with respect to their size, functionality, dynamic properties, etc.
- Robots need to ***interact*** with the objects (move around, manipulate, inspect, etc.) and with humans
- For all these tasks it is necessary that the robot knows to which ***class*** an object belongs



Object Classification Applications

Two major types of applications:

- **Object detection:** For a given test data set find all previously "learned" objects, e.g. pedestrians
- **Object recognition:** Find the particular "kind" of object as it was learned from the training data, e.g. handwritten character recognition



Learning

- A natural way to do object classification is to first **learn** the categories of the objects and then **infer** from the learned data a possible class for a new object.
- The area of **machine learning** deals with the formulization and investigates methods to do the learning automatically.
- Nowadays, machine learning algorithms are more and more used in robotics and computer vision

Mathematical Formulation

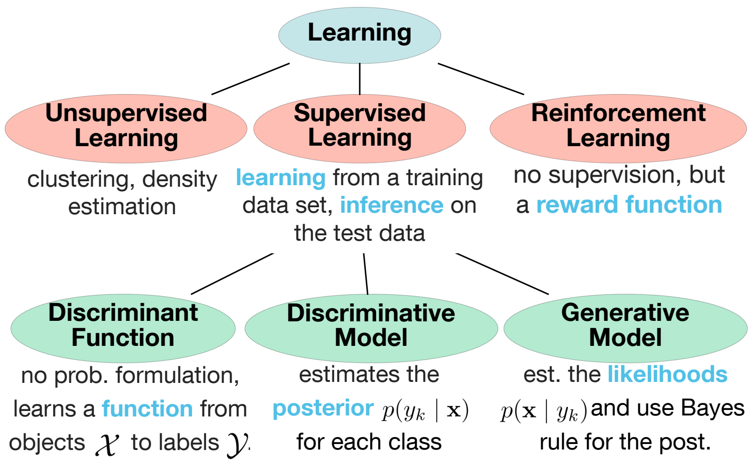
Suppose we are given a set $\mathcal{X}$ of objects and a set $\mathcal{Y}$ of object categories (classes). In the learning task we search for a mapping $\varphi : \mathcal{X} \rightarrow \mathcal{Y}$ such that **similar** elements in $\mathcal{X}$ are mapped to **similar** elements in $\mathcal{Y}$.

Examples:

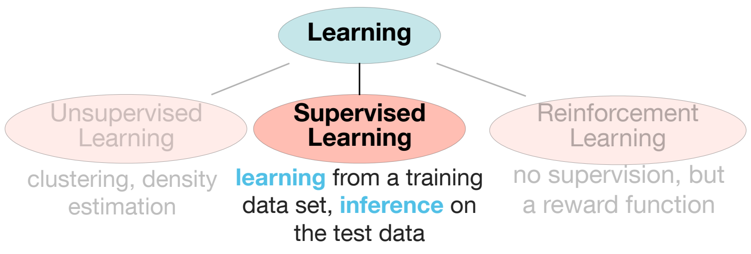
- Object classification: chairs, tables, etc.
- Optical character recognition
- Speech recognition

Important problem: Measure of similarity!

Categories of Learning


Learning
Unsupervised Learning
clustering, density estimation
Supervised Learning
learning from a training data set, inference on the test data
Reinforcement Learning
no supervision, but a reward function
Discriminant Function
no prob. formulation, learns a function from objects $\mathcal{X}$ to labels $\mathcal{Y}$.
Discriminative Model
estimates the posterior $p(y_k \mid \mathbf{x})$ for each class
Generative Model
est. the likelihoods $p(\mathbf{x} \mid y_k)$ and use Bayes rule for the post.

Categories of Learning

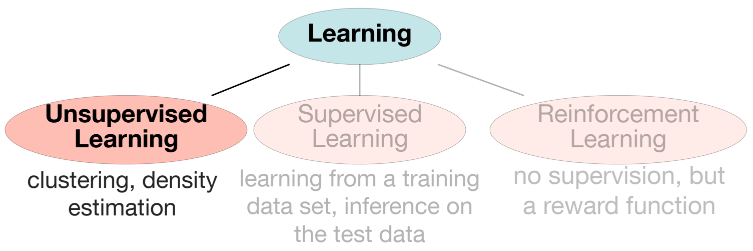

Learning
Unsupervised Learning
clustering, density estimation
Supervised Learning
learning from a training data set, inference on the test data
Reinforcement Learning
no supervision, but a reward function

Supervised Learning is the main topic of this lecture!

Methods used in Computer Vision include:

- Regression
- Conditional Random Fields
- Boosting
- Support Vector Machines
- Gaussian Processes
- Hidden Markov Models

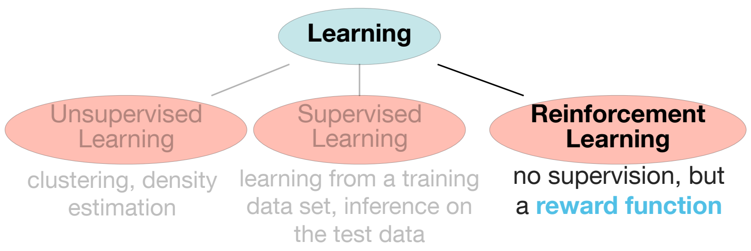
Categories of Learning


Learning
Unsupervised Learning
clustering, density estimation
Supervised Learning
learning from a training data set, inference on the test data
Reinforcement Learning
no supervision, but a reward function

Most Unsupervised Learning methods are based on Clustering.

➡Will be handled at the end of this semester

Categories of Learning


Learning
Unsupervised Learning
clustering, density estimation
Supervised Learning
learning from a training data set, inference on the test data
Reinforcement Learning
no supervision, but a reward function

Reinforcement Learning requires an ***action***

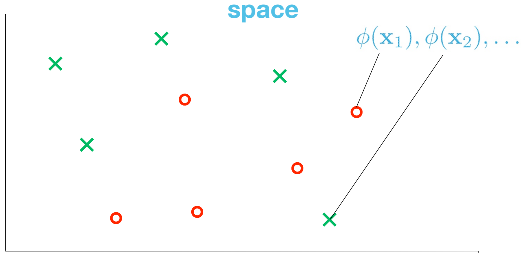
- the reward defines the quality of an action
- mostly used in robotics (e.g. manipulation)
- can be dangerous, actions need to be "tried out"
- not handled in this course

Generative Model: Example

Nearest-neighbor classification:

- Given: data points $(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \ldots$
- Rule: Each new data point is assigned to the class of its nearest neighbor in feature space

1. Training instances in feature space

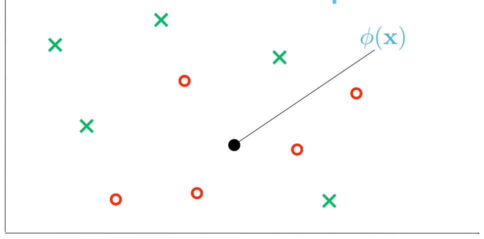


Generative Model: Example

Nearest-neighbor classification:

- Given: data points $(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \ldots$
- Rule: Each new data point is assigned to the class of its nearest neighbor in feature space

2. Map new data point into feature space

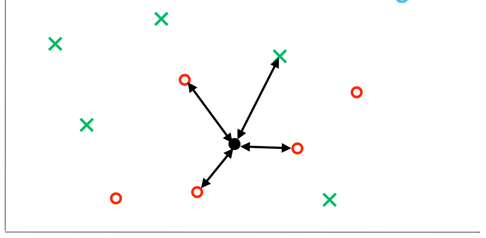


Generative Model: Example

Nearest-neighbor classification:

- Given: data points $(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \ldots$
- Rule: Each new data point is assigned to the class of its nearest neighbor in feature space

3. Compute the distances to the neighbors

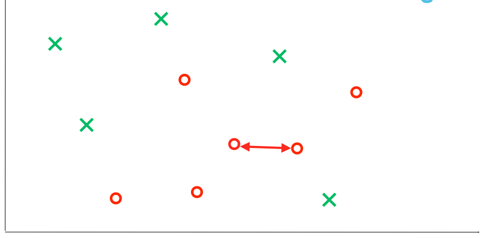


Generative Model: Example

Nearest-neighbor classification:

- Given: data points $(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \ldots$
- Rule: Each new data point is assigned to the class of its nearest neighbor in feature space

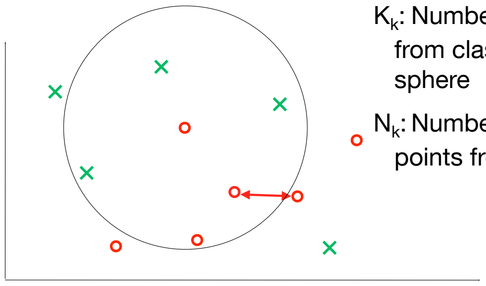
4. Assign the label of the nearest training instance



Generative Model: Example

Nearest-neighbor classification:

- General case: K nearest neighbors
- We consider a sphere around each training instance that has a fixed volume V.



K_k: Number of points from class k inside sphere

N_k: Number of all points from class k

Generative Model: Example

Nearest-neighbor classification:

- General case: K nearest neighbors
- We consider a sphere around each training instance that has a fixed volume V.
- With this we can estimate: $p(\mathbf{x} \mid y = k) = \frac{K_k}{N_k V}$ **"likelihood"**
- and likewise: $p(\mathbf{x}) = \frac{K}{NV}$ **"uncond. prob."**

  # points in sphere

  # all points
- using Bayes rule.

$$p(y = k \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid y = k)p(y = k)}{p(\mathbf{x})} = \frac{K_k}{K}$$ **"posterior"**

Generative Model: Example

Nearest-neighbor classification:

- General case: K nearest neighbors

$$p(y = k \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid y = k)p(y = k)}{p(\mathbf{x})} = \frac{K_k}{K}$$

- To classify the new data point $\mathbf{x}$ we compute the posterior for each class k = 1,2,… and assign the label that maximizes the posterior.

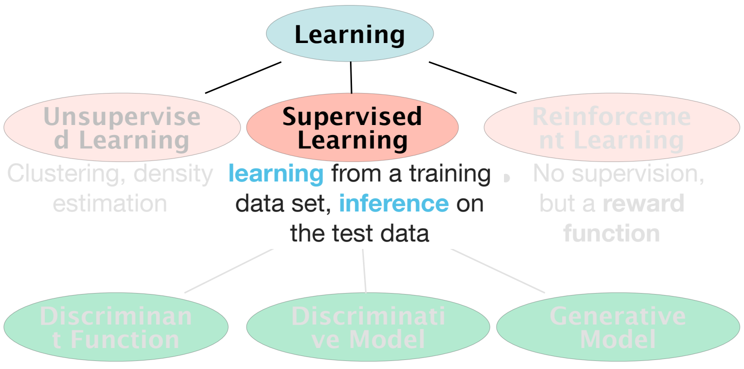
$$t := \arg\max_k p(y = k \mid \mathbf{x})$$

Summary

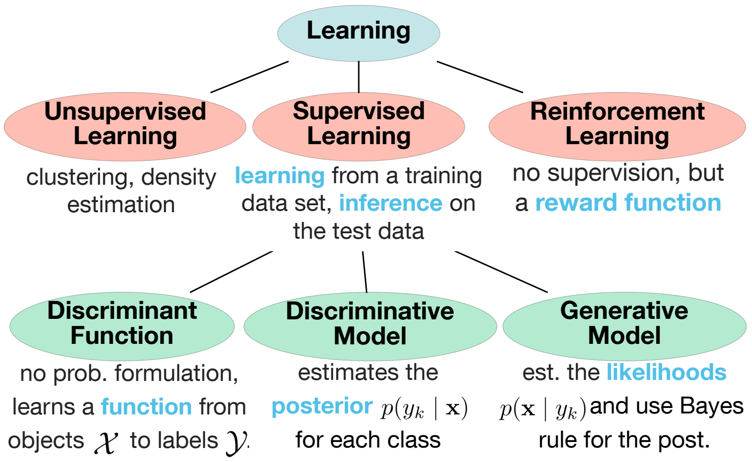
- Learning is a two-step process consisting in a *training* and an *inference* step
- Learning is useful to extract *semantic* information, e.g. about the objects in an environment
- There are three main categories of learning: *unsupervised*, *supervised* and *reinforcement* learning
- Supervised learning can be split into *discriminant function*, *discriminant model*, and *generative model* learning
- An example for a generative model is *nearest neighbor classification*

3. Regression

Categories of Learning (Rep.)


Learning
Unsupervised Learning
Clustering, density estimation
Supervised Learning
learning from a training data set, inference on the test data
Reinforcement Learning
No supervision, but a reward function
Discriminant Function
Discriminative Model
Generative Model

Categories of Learning


Learning
Unsupervised Learning
clustering, density estimation
Supervised Learning
learning from a training data set, inference on the test data
Reinforcement Learning
no supervision, but a reward function
Discriminant Function
no prob. formulation, learns a function from objects $\mathcal{X}$ to labels $\mathcal{Y}$.
Discriminative Model
estimates the posterior $p(y_k \mid \mathbf{x})$ for each class
Generative Model
est. the likelihoods $p(\mathbf{x} \mid y_k)$ and use Bayes rule for the post.

Mathematical Formulation (Rep.)

Suppose we are given a set $\mathcal{X}$ of objects and a set $\mathcal{Y}$ of object categories (classes). In the learning task we search for a mapping $\varphi : \mathcal{X} \rightarrow \mathcal{Y}$ such that ***similar*** elements in $\mathcal{X}$ are mapped to ***similar*** elements in $\mathcal{Y}$.

Difference between regression and classification:

- In regression, $\mathcal{Y}$ is ***continuous***, in classification it is discrete
- Regression learns a **function**, classification usually learns **class labels**

For now we will treat regression

Basis Functions

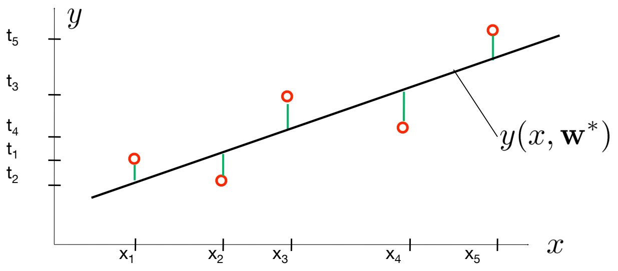
In principal, the elements of $\mathcal{X}$ can be anything (e.g. real numbers, graphs, 3D objects). To be able to treat these objects mathematically we need functions ϕ that map from $\mathcal{X}$ to $\mathbb{R}^N$. We call these the **basis functions**.

We can also interpret the basis functions as functions that extract **features** from the input data.

Features reflect the **properties** of the objects (width, height, etc.).

Simple Example: Linear Regression

- Assume: $\mathcal{X} = \mathbb{R}, \ \mathcal{Y} = \mathbb{R}, \ \phi = I$ (identity)
- **Given:** data points $(x_1, t_1), (x_2, t_2), \ldots$
- **Goal:** predict the value t of a new example x
- Parametric formulation: $y(x, \mathbf{w}) = w_0 + w_1 x$



Linear Regression

To evaluate the function y, we need an error function:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} (y(x_i, \mathbf{w}) - t_i)^2$$

"Sum of Squared Errors"

We search for parameters $\mathbf{w}^*$ s.th. $E(\mathbf{w}^*)$ is minimal:

$$\nabla E(\mathbf{w}) = \sum_{i=1}^{N} (y(x_i, \mathbf{w}) - t_i) \nabla y(x_i, \mathbf{w}) \doteq (0 \quad 0)$$

$$y(x_i, \mathbf{w}) = w_0 + w_1 x_i \qquad \Rightarrow \qquad \nabla y(x_i, \mathbf{w}) = (1 \quad x_i)$$

Using vector notation: $\mathbf{x}_i := (1 \quad x_i)^T \qquad y(x_i, \mathbf{w}) = \mathbf{w}^T \mathbf{x}_i$

$$\nabla E(\mathbf{w}) = \sum_{i=1}^{N} \mathbf{w}^T \mathbf{x}_i \mathbf{x}_i^T - \sum_{i=1}^{N} t_i \mathbf{x}_i^T = (0 \quad 0) \Rightarrow \mathbf{w}^T \underbrace{\sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T}_{=:A^T} = \underbrace{\sum_{i=1}^{N} t_i \mathbf{x}_i^T}_{=:b^T}$$

Polynomial Regression

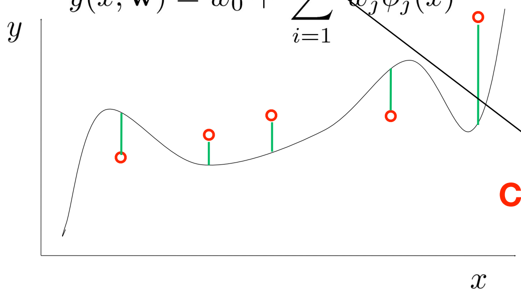
Now we have: $\mathcal{X} = \mathbb{R}, \ \mathcal{Y} = \mathbb{R}, \ \phi_j(x) = x^j$

Given: data points $(x_1, t_1), (x_2, t_2), \ldots, (x_N, t_N)$

$$y(x, \mathbf{w}) = w_0 + \sum_{i=1}^{M-1} w_j \phi_j(x)$$

Data Set Size

Model Complexity



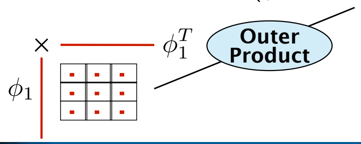
Polynomial Regression

We define: $\boldsymbol{\phi}(x) := (1, \phi_1(x), \dots, \phi_{M-1}(x))^T$ "Basis functions"

And obtain: $y(x, \mathbf{w}) = \mathbf{w}^T \boldsymbol{\phi}(x)$

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} (\mathbf{w}^T \phi(x_i) - t_i)^2$$

$$\nabla E(\mathbf{w}) = \mathbf{w}^T \left(\sum_{i=1}^{N} \phi(x_i)\phi(x_i)^T \right) - \sum_{i=1}^{N} t_i \phi(x_i)^T$$



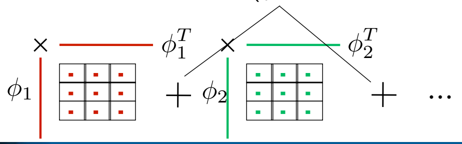
Polynomial Regression

We define: $\boldsymbol{\phi}(x) := (1, \phi_1(x), \dots, \phi_{M-1}(x))^T$

And obtain: $y(x, \mathbf{w}) = \mathbf{w}^T \boldsymbol{\phi}(x)$

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} (\mathbf{w}^T \phi(x_i) - t_i)^2$$

$$\nabla E(\mathbf{w}) = \mathbf{w}^T \left(\sum_{i=1}^{N} \phi(x_i)\phi(x_i)^T \right) - \sum_{i=1}^{N} t_i \phi(x_i)^T$$



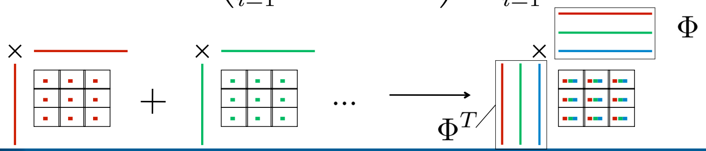
Polynomial Regression

We define: $\boldsymbol{\phi}(x) := (1, \phi_1(x), \dots, \phi_{M-1}(x))^T$

And obtain: $y(x, \mathbf{w}) = \mathbf{w}^T \boldsymbol{\phi}(x)$

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} (\mathbf{w}^T \phi(x_i) - t_i)^2$$

$$\nabla E(\mathbf{w}) = \mathbf{w}^T \left(\sum_{i=1}^{N} \phi(x_i)\phi(x_i)^T \right) - \sum_{i=1}^{N} t_i \phi(x_i)^T$$



Polynomial Regression

Thus, we have: $\displaystyle\sum_{i=1}^{N} \boldsymbol{\phi}(x_i)\boldsymbol{\phi}(x_i)^T = \Phi^T \Phi$

where $\Phi = \begin{pmatrix} \phi_0(x_1) & \phi_1(x_1) & \dots & \phi_{M-1}(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \dots & \phi_{M-1}(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \dots & \phi_{M-1}(x_N) \end{pmatrix}$

$\nabla E(\mathbf{w}) = \mathbf{w}^T \Phi^T \Phi - \mathbf{t}^T \Phi \quad \Rightarrow \quad \Phi^T \Phi \mathbf{w} = \Phi^T \mathbf{t}$

"Normal Equation"

It follows:

$\mathbf{w} = \underbrace{(\Phi^T \Phi)^{-1} \Phi^T}_{} \mathbf{t}$ "Pseudoinverse" Φ^+

Computing the Pseudoinverse

Mathematically, a pseudoinverse Φ^+ exists for every matrix Φ.

However: If Φ is (close to) singular the direct solution of Φ is numerically unstable.

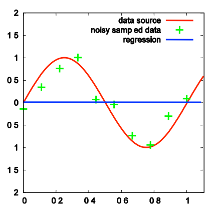
Therefore: Singular Value Decomposition (SVD) is used: $\Phi = UDV^T$ where

- matrices U and V are orthogonal matrices
- D is a diagonal matrix

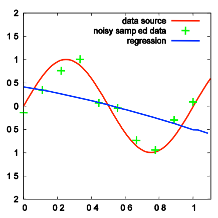
Then: $\Phi^+ = VD^+U^T$ where D^+ contains the *reciprocal* of all non-zero elements of D

A Simple Example

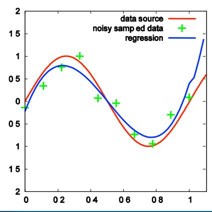
$\phi_j(x) = x^j$



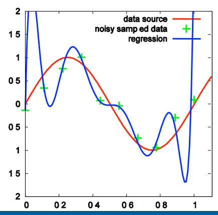
$N = 10$
$M = 1$



$N = 10$
$M = 3$

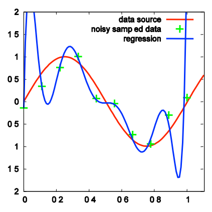


$N = 10$
$M = 5$

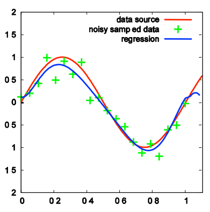


$N = 10$
$M = 10$

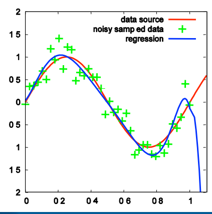
Varying the Sample Size



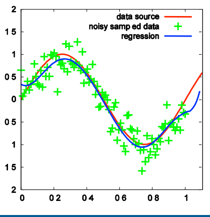
$N = 10$
$M = 10$



$N = 20$
$M = 10$

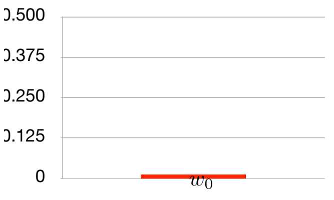


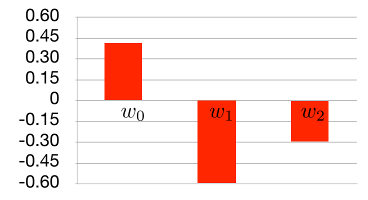
$N = 40$
$M = 10$

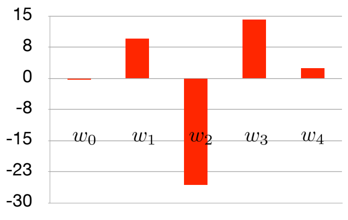


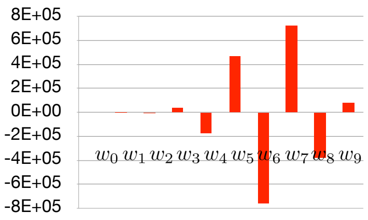
$N = 100$
$M = 10$

The Resulting Model Parameters









Other Basis Functions

Other basis functions are possible:

- Gaussian basis function:

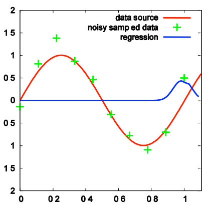
$$\phi_j(x) := \exp\left(-\frac{(x-\mu_j)^2}{2s^2}\right) \quad \text{where} \quad \begin{array}{l} \mu_j \triangleq \text{mean val} \\ s \triangleq \text{scale} \end{array}$$

- Sigmoidal basis function:

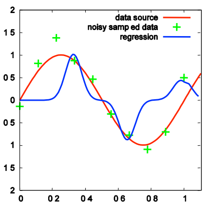
$$\phi_j(x) := \sigma\left(\frac{x-\mu_j}{s}\right) \quad \text{where} \quad \sigma(a) = \frac{1}{1+\exp(-a)}$$

In both cases a set of mean values is required. These define the **locations** of the basis functions.

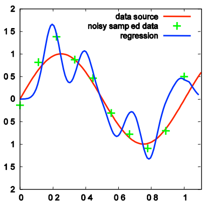
Gaussian Basis Functions



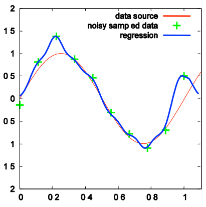
$N = 10$
$M = 1$



$N = 10$
$M = 3$

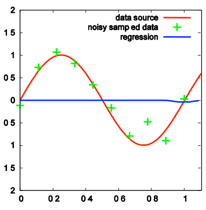


$N = 10$
$M = 5$

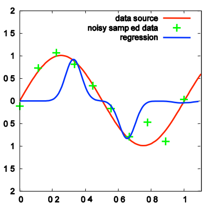


$N = 10$
$M = 10$

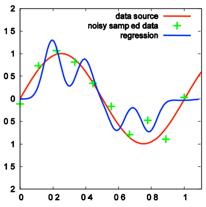
Sigmoidal Basis Functions



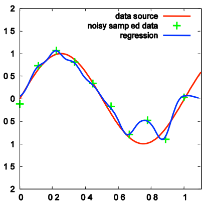
$N = 10$
$M = 1$



$N = 10$
$M = 3$



$N = 10$
$M = 5$



$N = 10$
$M = 10$

Observations

- The higher the model complexity grows, the better is the fit to the data
- If the model complexity is too high, all data points are explained well, but the resulting model oscillates very much. It can not generalize well. This is called ***overfitting.***
- By increasing the size of the data set (number of samples), we obtain a better fit of the model
- More complex models have larger parameters

Problem: How can we find a good model complexity for a given data set with a fixed size?

Regularization

We observed that complex models yield large parameters, leading to oscillation. Idea:

Minimize the error function and the magnitude of the parameters simultaneously

We do this by adding a regularization term :

$$\tilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(\mathbf{w}^T\boldsymbol{\phi}(x) - t_i\right)^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

where λ rules the influence of the regularization.

Regularization

As above, we set the derivative to zero:

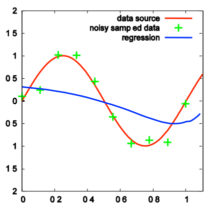
$$\nabla\tilde{E}(\mathbf{w}) = \sum_{i=1}^{N}\left(\mathbf{w}^T\boldsymbol{\phi}(x) - t_i\right)\boldsymbol{\phi}(x)^T + \lambda\mathbf{w}^T \doteq \mathbf{0}^T$$

$$\mathbf{w}^T\Phi^T\Phi + \lambda\mathbf{w}^T = \mathbf{t}^T\Phi \quad \Rightarrow \quad (\lambda I + \Phi^T\Phi)\mathbf{w} = \Phi^T\mathbf{t}$$

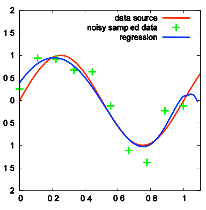
$$\mathbf{w} = (\lambda I + \Phi^T\Phi)^{-1}\Phi^T\mathbf{t}$$

With regularization, we can find a complex model for a small data set. However, the problem now is to find an appropriate regularization coefficient λ.

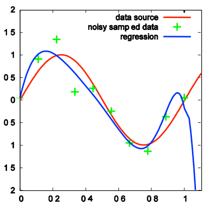
Regularized Results



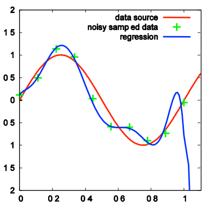
$N = 10$
$M = 10$
$\lambda = 1$



$N = 10$
$M = 10$
$\lambda = 10^{-3}$



$N = 10$
$M = 10$
$\lambda = 10^{-6}$



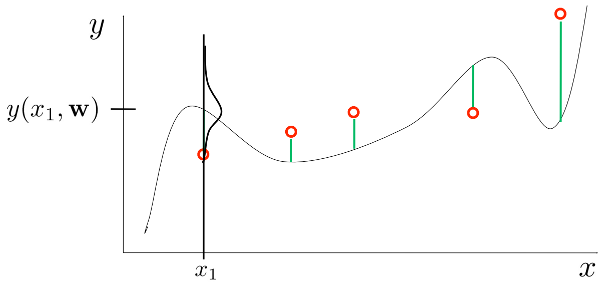
$N = 10$
$M = 10$
$\lambda = 10^{-11}$

The Problem from a Different View

Assume that y is affected by Gaussian noise :

$$t = y(x, \mathbf{w}) + \epsilon \quad \text{where} \quad \epsilon \rightsquigarrow \mathcal{N}(.; 0, \sigma^2)$$

Thus, we have $p(t \mid x, \mathbf{w}, \sigma) = \mathcal{N}(t; y(x, \mathbf{w}), \sigma^2)$



Maximum Likelihood Estimation

Aim: we want to find the $\mathbf{w}$ that maximizes p.

$p(t \mid x, \mathbf{w}, \sigma)$ is the *likelihood* of the measured data given a model. Intuitively:

Find parameters $\mathbf{w}$ that maximize the probability of measuring the already measured data t.

"Maximum Likelihood Estimation"

We can think of this as fitting a model $\mathbf{w}$ to the data t.

Note: σ is also part of the model and can be estimated.

For now, we assume σ is known.

Maximum Likelihood Estimation

Given data points: $(x_1, t_1), (x_2, t_2), \ldots, (x_N, t_N)$

Assumption: points are drawn independently from p:

$$\begin{aligned} p(\mathbf{t} \mid \mathbf{x}, \mathbf{w}, \sigma) &= \prod_{i=1}^{N} p(t_i \mid \mathbf{x}, \mathbf{w}, \sigma) \\ &= \prod_{i=1}^{N} \mathcal{N}(t_i; \mathbf{w}^T \boldsymbol{\phi}(x_i), \sigma^2) \end{aligned}$$

where:

$$\begin{aligned} \mathbf{x} &= (x_1, x_2, \ldots, x_N) \\ \mathbf{t} &= (t_1, t_2, \ldots, t_N) \end{aligned}$$

Instead of maximizing p we can also maximize its **logarithm** (monotonicity of the logarithm)

Maximum Likelihood Estimation

$$\mathcal{N} \rightarrow \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}$$

$$\begin{aligned} \ln p(\mathbf{t} \mid \mathbf{x}, \mathbf{w}, \sigma) &= \sum_{i=1}^{N} \ln p(t_i \mid \mathbf{x}, \mathbf{w}, \sigma) \\ &= \frac{1}{2} \sum_{i=1}^{N} -\ln(\sigma^2) - \ln(2\pi) - \frac{1}{\sigma^2}(\mathbf{w}^T \boldsymbol{\phi}(x_i) - t_i)^2 \\ &= \underbrace{\frac{-N(\ln(\sigma^2) + \ln(2\pi))}{2}}_{\text{Constant for all } \mathbf{w}} - \frac{1}{\sigma^2} \underbrace{\sum_{i=1}^{N} (\mathbf{w}^T \boldsymbol{\phi}(x_i) - t_i)^2}_{\text{Is equal to } E(\mathbf{w})} \end{aligned}$$

The parameters that maximize the likelihood are equal to the minimum of the sum of squared errors

Maximum Likelihood Estimation

$$\mathcal{N} \rightarrow \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}$$

$$\begin{aligned} \ln p(\mathbf{t} \mid \mathbf{x}, \mathbf{w}, \sigma) &= \sum_{i=1}^{N} \ln p(t_i \mid \mathbf{x}, \mathbf{w}, \sigma) \\ &= \frac{1}{2} \sum_{i=1}^{N} -\ln(\sigma^2) - \ln(2\pi) - \frac{1}{\sigma^2}(\mathbf{w}^T \boldsymbol{\phi}(x_i) - t_i)^2 \\ &= \frac{-N(\ln(\sigma^2) + \ln(2\pi))}{2} - \frac{1}{\sigma^2} \sum_{i=1}^{N} (\mathbf{w}^T \boldsymbol{\phi}(x_i) - t_i)^2 \end{aligned}$$

$$\mathbf{w}_{ML} := \arg\max_{\mathbf{w}} \ln p(\mathbf{t} \mid \mathbf{x}, \mathbf{w}, \sigma) = \arg\min_{\mathbf{w}} E(\mathbf{w}) = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{t}$$

The ML solution is obtained using the Pseudoinverse

Maximum A-Posteriori Estimation

So far, we searched for parameters $\mathbf{w}$, that maximize the data likelihood. Now, we assume a Gaussian *prior*:

$$p(\mathbf{w} \mid \sigma_2) = \mathcal{N}(\mathbf{w}; \mathbf{0}, \sigma_2 I)$$

Using this, we can compute the *posterior* (Bayes):

$$\underbrace{p(\mathbf{w} \mid x, \mathbf{t}, \sigma_1, \sigma_2)}_{\text{Posterior}} \propto \underbrace{p(t \mid x, \mathbf{w}, \sigma_1)}_{\text{Likelihood}} \underbrace{p(\mathbf{w} \mid \sigma_2)}_{\text{Prior}}$$

"Maximum A-Posteriori Estimation (MAP)"

Maximum A-Posteriori Estimation

So far, we searched for parameters $\mathbf{w}$, that maximize the data likelihood. Now, we assume a Gaussian *prior*:

$$p(\mathbf{w} \mid \sigma_2) = \mathcal{N}(\mathbf{w}; \mathbf{0}, \sigma_2 I)$$

Using this, we can compute the *posterior* (Bayes):

$$p(\mathbf{w} \mid x, \mathbf{t}, \sigma_1, \sigma_2) \propto p(t \mid x, \mathbf{w}, \sigma_1) p(\mathbf{w} \mid \sigma_2)$$

strictly: $$p(\mathbf{w} \mid x, \mathbf{t}, \sigma_1, \sigma_2) = \frac{p(t \mid x, \mathbf{w}, \sigma_1) p(\mathbf{w} \mid \sigma_2)}{\int p(t \mid x, \mathbf{w}, \sigma_1) p(\mathbf{w} \mid \sigma_2) d\mathbf{w}}$$

but the denominator is independent of $\mathbf{w}$ and we want to maximize p.

Maximum A-Posteriori Estimation

$$\ln p(\mathbf{w} \mid x, \mathbf{t}, \sigma_1, \sigma_2) \propto \underbrace{\ln p(t \mid x, \mathbf{w}, \sigma_1)}_{\text{const.} - \frac{1}{\sigma_1^2} \sum_{i=1}^{N} (\mathbf{w}^T \boldsymbol{\phi}(x) - t_i)^2} + \underbrace{\ln p(\mathbf{w} \mid \sigma_2)}_{\text{const.} - \frac{1}{2\sigma_2^2} \mathbf{w}^T \mathbf{w}}$$

$$\propto -\frac{1}{\sigma_1^2} \left(\sum_{i=1}^{N} (\mathbf{w}^T \boldsymbol{\phi}(x) - t_i)^2 + \frac{\sigma_1^2}{\sigma_2^2} \mathbf{w}^T \mathbf{w} \right)$$

This is equal to the regularized error minimization.

The MAP Estimate corresponds to a regularized error minimization where $\lambda = (\sigma_1 / \sigma_2)^2$

Summary

- Regression is a method to find a mathematical model (function) for a given data set
- Regression can be done by minimizing the sum of squared (SSE) errors, i.e. the distances to the data
- Maximum-likelihood estimation uses a probabilis-tic representation to fit a model into noisy data
- Maximum-likelihood under Gaussian noise is equivalent to SSE regression.
- Maximum-a-posteriori (MAP) estimation assumes a (Gaussian) prior on the model parameters
- MAP is solved by regularized regression

Bayesian Linear Regression

Bayesian Linear Regression

- Using MAP, we can find optimal model parameters, but for practical applications two questions arise:
- What happens in the case of **sequential data**, i.e. the data points are observed subsequently?
- Can we model the probability of measuring a new data point, given all old data points? This is called the **predictive distribution**:

$$p(\underbrace{t}_{\text{New target}} \mid \underbrace{x}_{\text{New data}}, \underbrace{\mathbf{t}}_{\text{Old targets}}, \underbrace{\mathbf{x}}_{\text{Old data}})$$

Some Useful Formulas Before

If we are given this:

I. $p(\mathbf{x}) = \mathcal{N}(\mathbf{x} \mid \mu, \Sigma_1)$

II. $p(\mathbf{y} \mid \mathbf{x}) = \mathcal{N}(\mathbf{y} \mid A\mathbf{x} + \mathbf{b}, \Sigma_2)$

Then it follows (properties of Gaussians):

III. $p(\mathbf{y}) = \mathcal{N}(\mathbf{y} \mid A\mu + \mathbf{b}, \Sigma_2 + A\Sigma_1 A^T)$

IV. $p(\mathbf{x} \mid \mathbf{y}) = \mathcal{N}(\mathbf{x} \mid \Sigma(A^T\Sigma_2^{-1}(\mathbf{y} - \mathbf{b}) + \Sigma_1^{-1}\mu), \Sigma)$

where

$$\Sigma(\Sigma_1^{-1} + A^T\Sigma_2^{-1}A)^{-1}$$

Sequential Data

Given: Prior mean $\mathbf{m}_0$ and covariance S_0, noise covariance σ $p_0(\mathbf{w} \mid S_0) = \mathcal{N}(\mathbf{w}; \mathbf{m}_0, S_0)$

1. Set $i = 0$
2. Observe data point (x_i, t_i)
3. Formulate the likelihood $p(t_i \mid x_i, \mathbf{w})$ as a function of $\mathbf{w}$ (= Gaussian with mean $\phi(x_i)^T\mathbf{w}$ and covariance σ)
4. Multiply the likelihood with the prior $p_i(\mathbf{w} \mid S_i)$ and normalize (= Gaussian with $\mathbf{m}_{i+1}$ and S_{i+1})
5. This results in a new prior $p_{i+1}(\mathbf{w} \mid S_{i+1})$
6. Go back to 1. if there are still data points available

Comparison: the Standard Bayes Filter

$$\text{Bel}(x_t) = p(x_t \mid u_1, z_1, \ldots, u_t, z_t)$$

(Bayes) $= \eta\ p(z_t \mid x_t, u_1, z_1, \ldots, u_t)p(x_t \mid u_1, z_1, \ldots, u_t)$

(Markov) $= \eta\ p(z_t \mid x_t)p(x_t \mid u_1, z_1, \ldots, u_t)$

(Tot. prob.) $= \eta\ p(z_t \mid x_t) \int p(x_t \mid u_1, z_1, \ldots, u_t, x_{t-1})\, p(x_{t-1} \mid u_1, z_1, \ldots, u_t)dx_{t-1}$

(Markov) $= \eta\ p(z_t \mid x_t) \int p(x_t \mid u_t, x_{t-1})p(x_{t-1} \mid u_1, z_1, \ldots, u_t)dx_{t-1}$

(Markov) $= \eta\ p(z_t \mid x_t) \int p(x_t \mid u_t, x_{t-1})p(x_{t-1} \mid u_1, z_1, \ldots, z_{t-1})dx_{t-1}$

$= \eta\ p(z_t \mid x_t) \int p(x_t \mid u_t, x_{t-1})\text{Bel}(x_{t-1})dx_{t-1}$

Comparison: the Standard Bayes Filter

$$\text{Bel}(x_t) = p(x_t \mid u_1, z_1, \ldots, u_t, z_t)$$

(Bayes) $= \eta\ p(z_t \mid x_t, u_1, z_1, \ldots, u_t)p(x_t \mid u_1, z_1, \ldots, u_t)$

(Markov) $= \eta\ p(z_t \mid x_t)p(x_t \mid u_1, z_1, \ldots, u_t)$

Note: Different Notation!

A Simple Example

Our aim to fit a straight line into a set of data points.

Assume we have:

Basis functions are equal to identity $\phi(\mathbf{x}) = \mathbf{x}$

Prior mean is zero, prior covariance $\sigma_2^2 = 0.5$, noise variance is $\sigma_1^2 = 0.2^2$

Ground truth is $f(x, \mathbf{a}) = a_0 + a_1 x$ where $a_1 = 0.5$, $a_0 = -0.3$

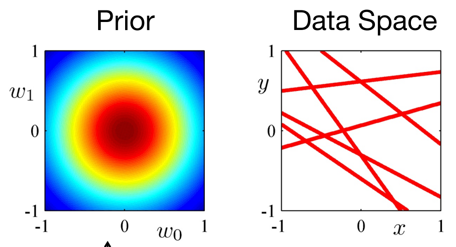
Data points are sampled from ground truth

Thus:

We want to recover a_0 and a_1 from the sequentially incoming data points $(x_1, t_1), (x_2, t_2), \ldots$

Bayesian Line Fitting

No data points observed



$\uparrow$

"Hough Space"

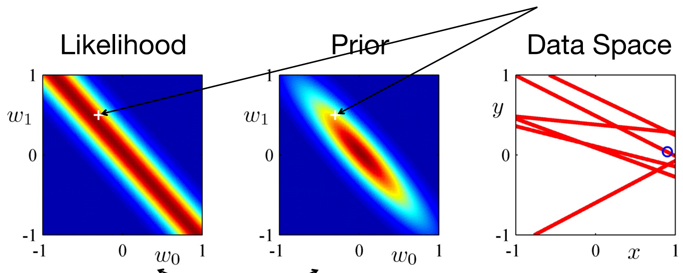
Line examples drawn from the prior

From: C.M. Bishop

Bayesian Line Fitting

One data point observed

Ground Truth



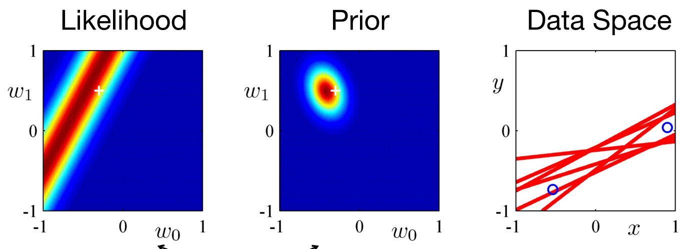
"Hough Space"

Line examples drawn from the prior

From: C.M. Bishop

Bayesian Line Fitting

Two data points observed



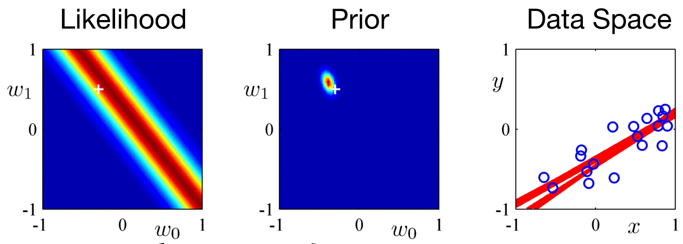
"Hough Space"

Line examples drawn from the prior

From: C.M. Bishop

Bayesian Line Fitting

20 data points observed



"Hough Space"

Line examples drawn from the prior

From: C.M. Bishop

The Predictive Distribution

We obtain the predictive distribution by integrating over all possible model parameters:

$$p(t \mid x, \mathbf{t}, \mathbf{x}) = \int \underbrace{p(t \mid x, \mathbf{w})}_{\text{New data likelihood}} \underbrace{p(\mathbf{w} \mid \mathbf{x}, \mathbf{t})}_{\text{Old data posterior}} d\mathbf{w}$$

As before the posterior is prop. to the likelihood times the prior. But now, we don't maximize. The posterior can be computed analytically, as the prior is Gaussian.

$p(\mathbf{w} \mid \mathbf{x}, \mathbf{t}) = \mathcal{N}(\mathbf{w} \mid \mathbf{m}_N, S_N)$ where

$$S_N^{-1} = \underbrace{S_0^{-1}}_{\text{Prior cov}} + \sigma^{-2}\Phi^T\Phi$$

$$\mathbf{w}_N = S_N(S_0^{-1}\underbrace{\mathbf{m}_0}_{\text{Prior mean}} + \sigma^{-2}\Phi^T\mathbf{t})$$

The Predictive Distribution

Using formula III. from above,

$$p(t \mid x, \mathbf{t}, \mathbf{x}) = \int p(t \mid x, \mathbf{w}) p(\mathbf{w} \mid \mathbf{x}, \mathbf{t}) d\mathbf{w}$$

$$= \int \mathcal{N}(t; \mathbf{w}^T\phi(x), \sigma)\mathcal{N}(\mathbf{w}; \mathbf{m}_N, S_N)$$

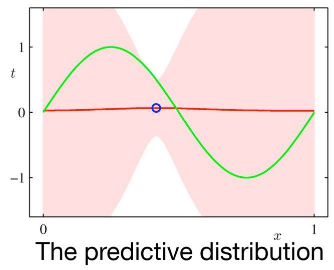
$$= \mathcal{N}(t; \mathbf{m}_N^T\phi(x), \sigma_N^2(x))$$

where

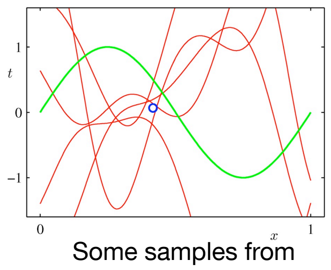
$$\sigma_N^2(x) = \sigma^2 + \phi(x)^T S_N \phi(x)$$

The Predictive Distribution (2)

- Example: Sinusoidal data, 9 Gaussian basis functions, 1 data point



The predictive distribution

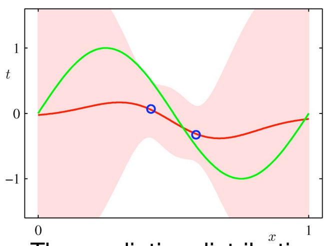


Some samples from the posterior

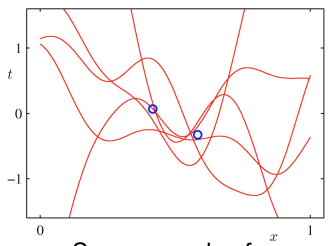
From: C.M. Bishop

Predictive Distribution (3)

- Example: Sinusoidal data, 9 Gaussian basis functions, 2 data points



The predictive distribution

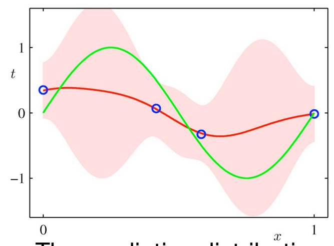


Some samples from the posterior

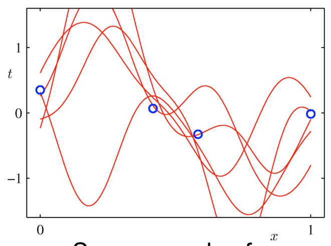
From: C.M. Bishop

Predictive Distribution (4)

- Example: Sinusoidal data, 9 Gaussian basis functions, 4 data points



The predictive distribution

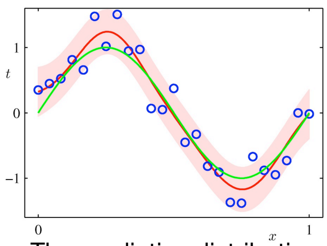


Some samples from the posterior

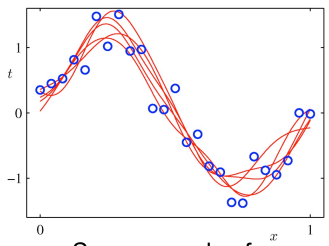
From: C.M. Bishop

Predictive Distribution (5)

- Example: Sinusoidal data, 9 Gaussian basis functions, 25 data points



The predictive distribution



Some samples from the posterior

From: C.M. Bishop

Summary

- Regression can be expressed as a least-squares problem
- To avoid overfitting, we need to introduce a regularisation term with an additional parameter λ
- Regression without regularisation is equivalent to Maximum Likelihood Estimation
- Regression + reg = Maximum A-Posteriori
- Bayesian Linear Regression operates on sequential data and provides the predictive distribution
- When using Gaussian priors (and Gaussian noise), all computations can be done analytically

4. Probabilistic Graphical Models Directed Models

The Bayes Filter (Rep.)

$$\boxed{\text{Bel}(x_t)} = p(x_t \mid u_1, z_1, \dots, u_t, z_t)$$

(Bayes) $= \eta\, p(z_t \mid x_t, u_1, z_1, \dots, u_t) p(x_t \mid u_1, z_1, \dots, u_t)$

(Markov) $= \eta\, p(z_t \mid x_t) p(x_t \mid u_1, z_1, \dots, u_t)$

(Tot. prob.) $= \eta\, p(z_t \mid x_t) \int p(x_t \mid u_1, z_1, \dots, u_t, x_{t-1})\, p(x_{t-1} \mid u_1, z_1, \dots, u_t) dx_{t-1}$

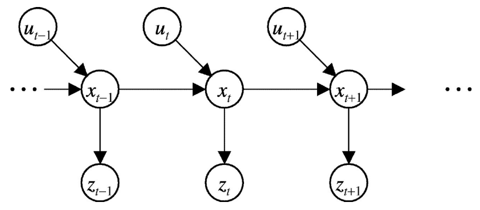
(Markov) $= \eta\, p(z_t \mid x_t) \int p(x_t \mid u_t, x_{t-1}) p(x_{t-1} \mid u_1, z_1, \dots, u_t) dx_{t-1}$

(Markov) $= \eta\, p(z_t \mid x_t) \int p(x_t \mid u_t, x_{t-1}) p(x_{t-1} \mid u_1, z_1, \dots, z_{t-1}) dx_{t-1}$

$$\boxed{= \eta\, p(z_t \mid x_t) \int p(x_t \mid u_t, x_{t-1}) \text{Bel}(x_{t-1}) dx_{t-1}}$$

Graphical Representation (Rep.)

We can describe the overall process using a ***Dynamic Bayes Network***:



- This incorporates the following Markov assumptions:

$p(z_t \mid x_{0:t}, u_{1:t}, z_{1:t}) = p(z_t \mid x_t)$ (measurement)

$p(x_t \mid x_{0:t-1}, u_{1:t}, z_{1:t}) = p(x_t \mid x_{t-1}, u_t)$ (state)

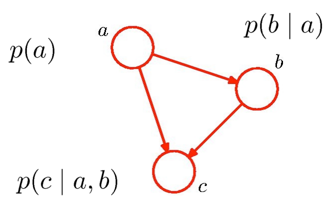
Definition

A Probabilistic Graphical Model is a diagrammatic representation of a probability distribution.

- In a Graphical Model, random variables are represented as nodes, and statistical dependencies are represented using edges between the nodes.
- The resulting graph can have the following properties:
- Cyclic / acyclic
- Directed / undirected
- The simplest graphs are Directed Acyclig Graphs (DAG).

Simple Example

- Given: 3 random variables a, b, and c
- Joint prob: $p(a,b,c) = p(c|a,b)p(a,b) = p(c|a,b)p(b|a)p(a)$



Random variables can be discrete or continuous

A Graphical Model based on a DAG is called a **Bayesian Network**

Simple Example

- In general: K random variables $x_1, x_2, \dots, x_K$
- Joint prob:

$$p(x_1, \dots, x_K) = p(x_K | x_1, \dots, x_{K-1}) \dots p(x_2|x_1)p(x_1)$$

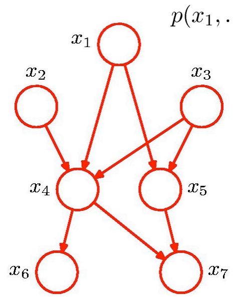
- This leads to a fully connected graph.
- Note: The ordering of the nodes in such a fully connected graph is arbitrary. They all represent the joint probability distribution:

$$p(a,b,c) = p(a|b,c)p(b|c)p(c)$$
$$p(a,b,c) = p(b|a,c)p(a|c)p(c)$$
$$\vdots$$

Bayesian Networks

Statistical independence can be represented by the absence of edges. This makes the computation efficient.

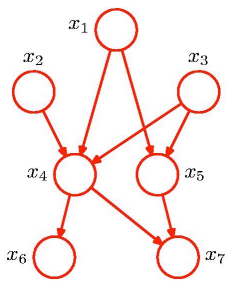
$$p(x_1,\ldots,x_7) = p(x_1)p(x_2)p(x_3)p(x_4|x_1,x_2,x_3)\,p(x_5|x_1,x_3)p(x_6|x_4)p(x_7|x_4,x_5)$$



Intuitively: only x_1 and x_3 have an influence on x_5

Bayesian Networks

We can now define a one-to-one mapping from graphical models to probabilistic formulations:



General Factorization:

$$p(\mathbf{x}) = \prod_{k=1}^{K} p(x_k|\mathrm{pa}_k)$$

where

$pa_k \triangleq$ ancestors of p_k

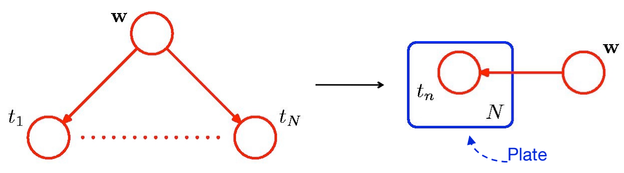
and

$$p(\mathbf{x}) = p(x_1,\ldots,x_K)$$

Elements of Graphical Models

In case of a series of random variables with equal dependencies, we can subsume them using a **plate:**

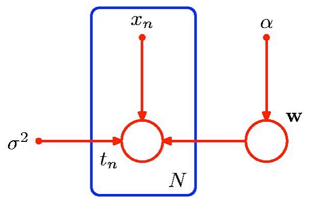
$$p(\mathbf{t},\mathbf{w}) = p(\mathbf{w})\prod_{n=1}^{N} p(t_n|\mathbf{w})$$



Elements of Graphical Models (2)

We distinguish between **input** variables and explicit **hyper-parameters**:

$$p(\mathbf{t},\mathbf{w}|\mathbf{x},\alpha,\sigma^2) = p(\mathbf{w}|\alpha)\prod_{n=1}^{N} p(t_n|\mathbf{w},x_n,\sigma^2).$$

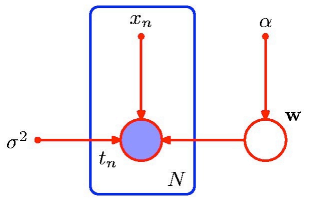


Elements of Graphical Models (3)

We distinguish between **observed** variables and **hidden** variables:

$$p(\mathbf{w}|\mathbf{t}) \propto p(\mathbf{w})\prod_{n=1}^{N} p(t_n|\mathbf{w})$$

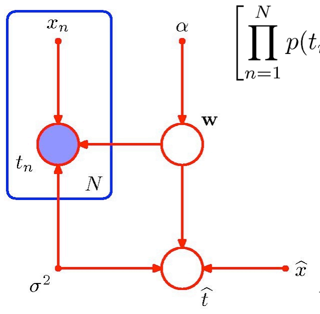
(deterministic parameters omitted)



Regression as a Graphical Model

Regression: Prediction of a new target value $\hat{t}$

$$p(\hat{t},\mathbf{t},\mathbf{w} \mid \hat{x},\mathbf{x},\alpha,\sigma^2) = \left[\prod_{n=1}^{N} p(t_n \mid x_n,\mathbf{w},\sigma^2)\right] p(\mathbf{w} \mid \alpha)p(\hat{t}|\hat{x},\mathbf{w},\sigma^2)$$



Here: conditioning on all deterministic parameters

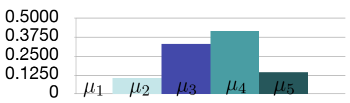
Using this, we can obtain the predictive distribution:

$$p(\hat{t}|\hat{x},\mathbf{x},\mathbf{t},\alpha,\sigma^2) \propto \int p(\hat{t},\mathbf{t},\mathbf{w}|\hat{x},\mathbf{x},\alpha,\sigma^2)\,\mathrm{d}\mathbf{w}$$

Two Special Cases

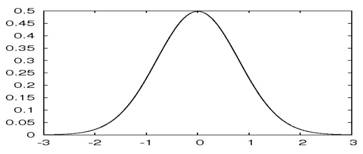
- We consider two special cases:
- All random variables are discrete; i.e. Each x_i is represented by values $\mu_1,\ldots,\mu_K$ where

$$p(x \mid \boldsymbol{\mu}) = \prod_{k=1}^{K}\mu_k^{x_k} \qquad \sum_{j=1}^{K}\mu_j = 1$$



- All random variables are Gaussian

$$x_i \sim \mathcal{N}(.;\mu_i,\sigma_i^2)$$



Discrete Variables: Example

- Two dependent variables: K² - 1 parameters

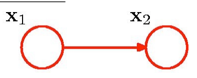
Here: K = 2

x_1	$p(x_1)$
1	0.2
2	0.8

$\}K-1$

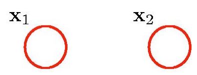
x_1	x_2	$p(x_2 \mid x_1)$
1	1	0.25
1	2	0.75
2	1	0.1
2	2	0.9

$\}K-1$ $\}K-1$ $\}K(K-1)$



$$K-1+K(K-1) = K^2-1$$

- Independent joint distribution: 2(K – 1) parameters

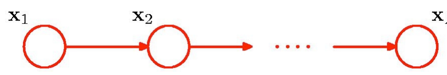


$$K-1+K-1 = 2(K-1)$$

Discrete Variables: General Case

In a general joint distribution with M variables we need to store $K^M - 1$ parameters

If the distribution can be described by this graph:



then, we have only K - 1 + (M - 1) K(K - 1) parameters.

This graph is called a **Markov chain** with M nodes.

The number of parameters grows only **linearly** with the number of variables.

Gaussian Variables

Assume all random variables are Gaussian and we define

$$p(x_i \mid \mathrm{pa}_i) = \mathcal{N}\left(x_i;\ \sum_{j \in \mathrm{pa}_i} w_{ij} x_j + b_i, v_i\right)$$

Then one can show that the joint probability p(x) is a multivariate Gaussian. Furthermore:

$$x_i = \sum_{j \in \mathrm{pa}_i} w_{ij} x_j + b_j + \sqrt{v_i}\epsilon_i \qquad \epsilon_i \sim \mathcal{N}(0,1)$$

Thus:

$$E[x_i] = \sum_{j \in \mathrm{pa}_i} w_{ij} E[x_j] + b_i$$

i.e., we can compute the mean values recursively.

Gaussian Variables

Assume all random variables are Gaussian and we define

$$p(x_i \mid \mathrm{pa}_i) = \mathcal{N}\left(x_i;\ \sum_{j \in \mathrm{pa}_i} w_{ij} x_j + b_i, v_i\right)$$

The same can be shown for the covariance. Thus:

- Mean and covariance can be calculated recursively

Furthermore it can be shown that:

- The **fully connected** graph corresponds to a Gaussian with a **general symmetric** covariance matrix
- The **non-connected** graph corresponds to a **diagonal** covariance matrix

Independence (Rep.)

Definition 1.4: Two random variables X and Y are ***independent*** iff: $p(x,y) = p(x)p(y)$

For independent random variables X and Y we have:

$$p(x \mid y) = \frac{p(x,y)}{p(y)} = \frac{p(x)p(y)}{p(y)} = p(x)$$

Notation: $x \perp\!\!\!\perp y \mid \emptyset$

Independence does not imply conditional independence.

The same is true for the opposite case.

Conditional Independence (Rep.)

Definition 1.5: Two random variables X and Y are ***conditional independent*** given a third random variable Z iff:

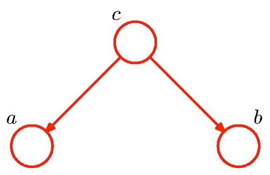
$$p(x, y \mid z) = p(x \mid z)p(y \mid z)$$

This is equivalent to:

$$p(x \mid z) = p(x \mid y, z) \quad \text{and}$$
$$p(y \mid z) = p(y \mid x, z)$$

Notation: $x \perp\!\!\!\perp y \mid z$

Conditional Independence: Example 1



This graph represents the probability distribution:

$$p(a,b,c) = p(a|c)p(b|c)p(c)$$

Marginalizing out *c* on both sides gives

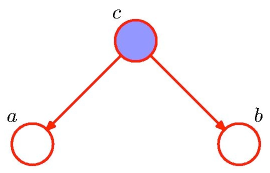
$$p(a,b) = \sum_c p(a|c)p(b|c)p(c)$$

This is in general not equal to $p(a)p(b)$.

Thus: a and b are not independent: $a \not\perp\!\!\!\perp b \mid \emptyset$

Conditional Independence: Example 1

- Now, we condition on c (it is assumed to be known):

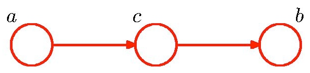


$$\begin{aligned} p(a,b|c) &= \frac{p(a,b,c)}{p(c)} \\ &= p(a|c)p(b|c) \end{aligned}$$

Thus: a and b are conditionally independent given c: $a \perp\!\!\!\perp b \mid c$

We say that the node at c is a **tail-to-tail node** on the path between a and b

Conditional Independence: Example 2



This graph represents the distribution:

$$p(a,b,c) = p(a)p(c|a)p(b|c)$$

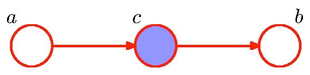
Again, we marginalize over c:

$$\begin{aligned} p(a,b) &= p(a)\sum_c p(c|a)p(b|c) = p(a)\sum_c p(c|a)p(b|c,a) \\ &= p(a)\sum_c \frac{p(c,a)p(b,c,a)}{p(a)p(c,a)} = p(a)\sum_c p(b,c \mid a) \\ &= p(a)p(b|a) \end{aligned}$$

And we obtain: $a \not\perp\!\!\!\perp b \mid \emptyset$

Conditional Independence: Example 2

As before, now we condition on c :



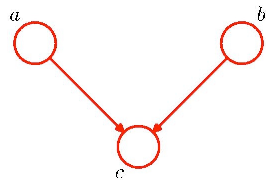
$$\begin{aligned} p(a,b|c) &= \frac{p(a,b,c)}{p(c)} \\ &= \frac{p(a)p(c|a)p(b|c)}{p(c)} \\ &= p(a|c)p(b|c) \end{aligned}$$

And we obtain: $a \perp\!\!\!\perp b \mid c$

We say that the node at c is a **head-to-tail node** on the path between a and b.

Conditional Independence: Example 3

Now consider this graph:



$$p(a,b,c) = p(a)p(b)p(c|a,b)$$

using:

$$\sum_c p(a,b,c) = p(a)p(b) \sum_c p(c \mid a,b)$$

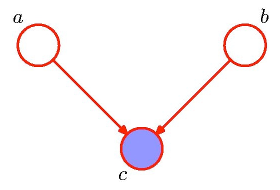
we obtain:

$$p(a,b) = p(a)p(b)$$

And the result is: $a \perp\!\!\!\perp b \mid \emptyset$

Conditional Independence: Example 3

Again, we condition on c



$$\begin{aligned} p(a,b|c) &= \frac{p(a,b,c)}{p(c)} \\ &= \frac{p(a)p(b)p(c|a,b)}{p(c)} \end{aligned}$$

This results in: $a \not\perp\!\!\!\perp b \mid c$

We say that the node at c is a **head-to-head node** on the path between a and b.

To Summarize

- When does the graph represent (conditional) independence?

  Tail-to-tail case: if we condition on the tail-to-tail node

  Head-to-tail case: if we cond. on the head-to-tail node

  Head-to-head case: if we do **not** condition on the head-to-head node (and neither on any of its descendants)

In general, this leads to the notion of D-separation for directed graphical models.

D-Separation

Say: A, B, and C are non-intersecting subsets of nodes in a directed graph.

- A path from A to B is **blocked** by C if it contains a node such that either

  a) the arrows on the path meet either **head-to-tail** or **tail-to-tail** at the node, and the node is **in** the set C, or

  b) the arrows meet **head-to-head** at the node, and neither the node, nor any of its descendants, are in the set C.

- If all paths from A to B are blocked, A is said to be **d-separated** from B by C.

Notation: $\mathrm{dsep}(A,B|C)$

D-Separation

Say: A, B, and C are non-intersecting subsets of nodes

•A path ntains

a nod

a) the a tail-to-

tail at t

b) the a neither

the noc C.

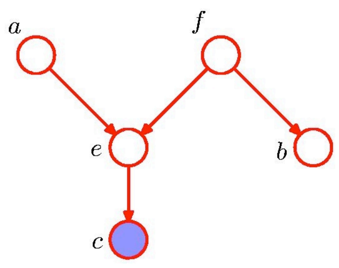
•If all p aid to

be **d-separated** from B by C.

Notation: $\mathrm{dsep}(A,B|C)$

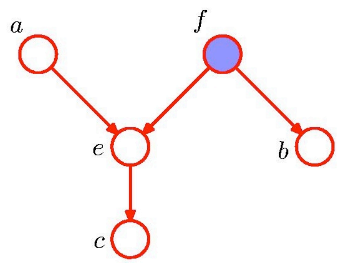
D-Separation is a property of graphs and not of probability distributions

D-Separation: Example



$\neg\mathrm{dsep}(a,b|c)$

We condition on a descendant of e, i.e. it does not block the path from a to b.



$\mathrm{dsep}(a,b|f)$

We condition on a tail-to-tail node on the only path from a to b, i.e f blocks the path.

I-Map

Definition 4.1: A graph G is called an **I-map** for a distribution p if every D-separation of G corresponds to a conditional independence relation satisfied by p:

$$\forall A,B,C : \mathrm{dsep}(A,B,C) \Rightarrow A \perp\!\!\!\perp B \mid C$$

Example: The fully connected graph is an I-map for any distribution, as there are no D-separations in that graph.

D-Map

Definition 4.2: A graph G is called an D-map for a distribution p if for every conditional independence relation satisfied by p there is a D-separation in G :

$$\forall A, B, C : A \perp\!\!\!\perp B \mid C \Rightarrow \text{dsep}(A, B, C)$$

Example: The graph without any edges is a D-map for any distribution, as all pairs of subsets of nodes are D-separated in that graph.

Perfect Map

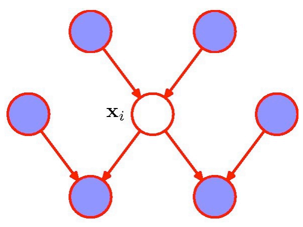
Definition 4.3: A graph G is called a perfect map for a distribution p if it is a D-map and an I-map of p.

$$\forall A, B, C : A \perp\!\!\!\perp B \mid C \Leftrightarrow \text{dsep}(A, B, C)$$

A perfect map uniquely defines a probability distribution.

The Markov Blanket

- Consider a distribution of a node x_i conditioned on all other nodes:



$$p(\mathbf{x}_i|\mathbf{x}_{\{j\neq i\}}) = \frac{p(\mathbf{x}_1,\dots,\mathbf{x}_M)}{\int p(\mathbf{x}_1,\dots,\mathbf{x}_M)d\mathbf{x}_i} = \frac{\prod_k p(\mathbf{x}_k|\text{pa}_k)}{\int \prod_k p(\mathbf{x}_k|\text{pa}_k)d\mathbf{x}_i} = p(\mathbf{x}_i \mid \mathbf{x}_{\mathcal{M}_i})$$

Markov blanket $\mathcal{M}_i$ at x_i : all parents, children and co-parents of x_i.

Factors independent of x_i cancel between numerator and denominator.

Summary

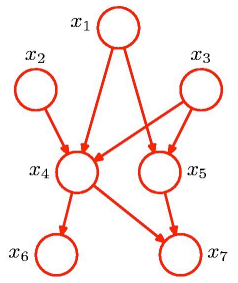
- Graphical models represent joint probability distributions using nodes for the random variables and edges to express (conditional) (in)dependence
- A prob. dist. can always be represented using a fully connected graph, but this is inefficient
- In a directed acyclic graph, conditional independence is determined using D-separation
- A perfect map implies a one-to-one mapping between c.i. relations and D-separations
- The Markov blanket is the minimal set of observed nodes to obtain conditional independence

Computer Vision Group
Prof. Daniel Cremers

Technische Universität München

4. Probabilistic Graphical Models Undirected Models

Repetition: Bayesian Networks

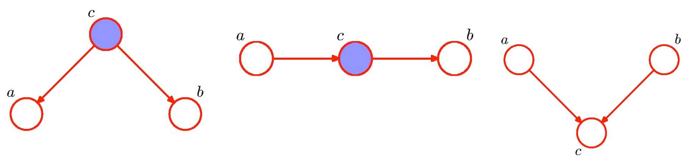


Directed graphical models can be used to represent **probability distributions**

This is useful to do **inference** and to **generate samples** from the distribution efficiently

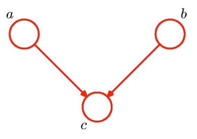
$$p(x_1,\dots,x_7) = p(x_1)p(x_2)p(x_3)p(x_4|x_1,x_2,x_3) p(x_5|x_1,x_3)p(x_6|x_4)p(x_7|x_4,x_5)$$

Repetition: D-Separation



- D-separation is a property of graphs that can be easily determined
- An I-map assigns every d-separation a c.i. rel
- A D-map assigns every c.i. rel a d-separation
- Every Bayes net determines a unique prob. dist.

In-depth: The Head-to-Head Node

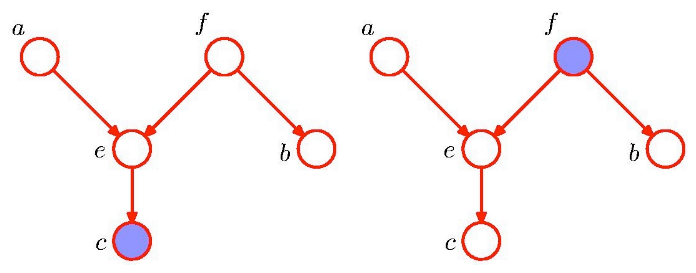


$p(a) = 0.9 \qquad p(b) = 0.9$

a	b	p(c)
1	1	0.8
1	0	0.2
0	1	0.2
0	0	0.1

Example:
a: Battery charged (0 or 1)
b: Fuel tank full (0 or 1)
c: Fuel gauge says full (0 or 1)
We can compute $p(\neg c) = 0.315$
and $p(\neg c \mid \neg b) = 0.81$
and obtain $p(\neg b \mid \neg c) \approx 0.257$
similarly: $p(\neg b \mid \neg c, \neg a) \approx 0.111$
"a explains c away"

Repetition: D-Separation



$\neg\text{dsep}(a, b|c)$ $\text{dsep}(a, b|f)$

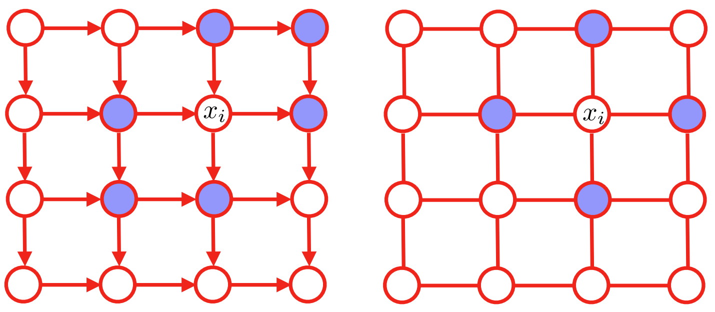
Directed vs. Undirected Graphs

Using D-separation we can identify conditional independencies in directed graphical models, but:

- Is there a simpler, more intuitive way to express conditional independence in a graph?
- Can we find a representation for cases where an „ordering“ of the random variables is inappropriate (e.g. the pixels in a camera image)?

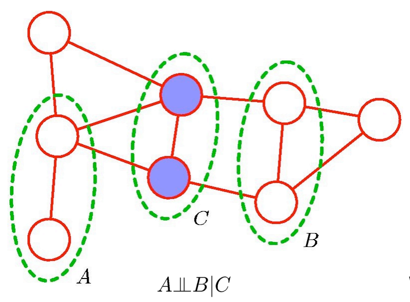
Yes, we can: by removing the directions of the edges we obtain an Undirected Graphical Model, also known as a **Markov Random Field**

Example: Camera Image



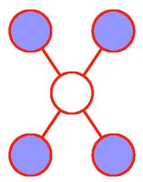
- directions are counter-intuitive for images
- Markov blanket is not just the direct neighbors when using a directed model

Markov Random Fields



All paths from A to B go through C, i.e. C blocks all paths.

Markov Blanket



We only need to condition on the **direct neighbors** of $\mathbf{x}$ to get c.i., because these already block every path from $\mathbf{x}$ to any other node.

Factorization of MRFs

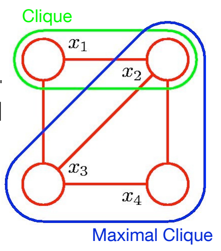
Any two nodes x_i and x_j that are not connected in an MRF are conditionally independent given all other nodes:

$$p(x_i, x_j \mid \mathbf{x}_{\setminus\{i,j\}}) = p(x_i \mid \mathbf{x}_{\setminus\{i,j\}}) p(x_j \mid \mathbf{x}_{\setminus\{i,j\}})$$

In turn: each factor contains only nodes that are connected

This motivates the consideration of cliques in the graph:

- A **clique** is a fully connected subgraph.
- A **maximal** clique can not be extended with another node without loosing the property of full connectivity.



Factorization of MRFs

In general, a Markov Random Field is factorized as

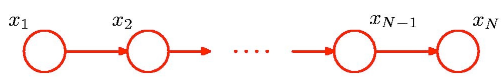
$$p(\mathbf{x}) = \frac{\prod_C \phi_C(\mathbf{x}_C)}{\sum_{\mathbf{x}'} \prod_C \phi_C(\mathbf{x}'_C)} = \frac{1}{Z} \prod_C \phi_C(\mathbf{x}_C) \qquad (4.1)$$

where C is the set of all (maximal) cliques and Φ_C is a positive function of a given clique $\mathbf{x}_C$ of nodes, called the **clique potential**. Z is called the **partition function**.

Theorem (Hammersley/Clifford): Any undirected model with associated clique potentials Φ_C is a perfect map for the probability distribution defined by Equation (4.1).

As a conclusion, all probability distributions that can be factorized as in (4.1), can be represented as an MRF.

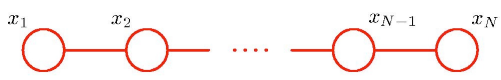
Converting Directed to Undirected Graphs (1)



$$p(\mathbf{x}) = \underbrace{p(x_1)p(x_2|x_1)}\ p(x_3|x_2) \cdots p(x_N|x_{N-1})$$

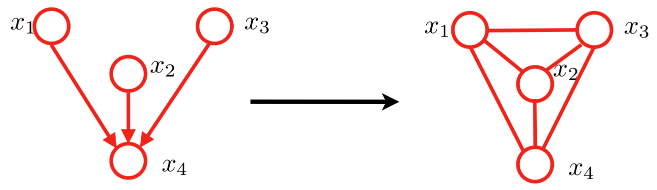
$$\Updownarrow$$

$$p(\mathbf{x}) = \frac{1}{Z}\ \psi_{1,2}(x_1, x_2)\ \psi_{2,3}(x_2, x_3) \cdots \psi_{N-1,N}(x_{N-1}, x_N)$$



In this case: Z=1

Converting Directed to Undirected Graphs (2)



$$p(\mathbf{x}) = p(x_1)p(x_2)p(x_2)p(x_4 \mid x_1, x_2, x_3)$$

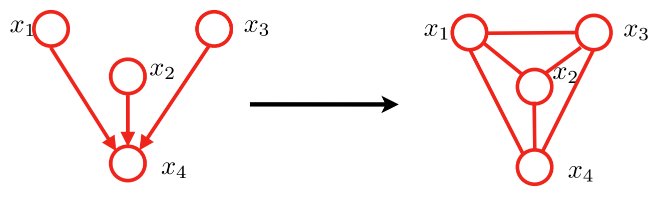
$$\uparrow$$

In general: conditional distributions in the directed graph are mapped to cliques in the undirected graph

However: the variables are **not** conditionally independent given the head-to-head node

Therefore: Connect all parents of head-to-head nodes with each other (**moralization**)

Converting Directed to Undirected Graphs (2)



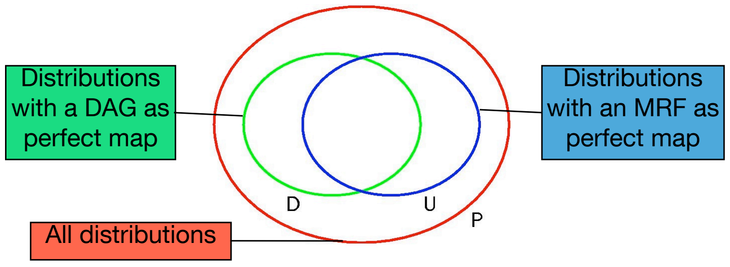
$p(\mathbf{x}) = p(x_1)p(x_2)p(x_2)p(x_4 \mid x_1, x_2, x_3)$ $\quad$ $p(\mathbf{x}) = \phi(x_1, x_2, x_3, x_4)$

Problem: This process can remove conditional independence relations (inefficient)

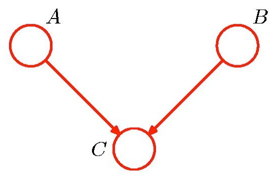
Generally: There is no one-to-one mapping between the distributions represented by directed and by undirected graphs.

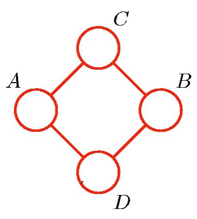
Representability

- As for DAGs, we can define an I-map, a D-map and a perfect map for MRFs.
- The set of all distributions for which a DAG exists that is a perfect map is different from that for MRFs.


Distributions with a DAG as perfect map
Distributions with an MRF as perfect map
D
U
P
All distributions

Directed vs. Undirected Graphs





$A \perp\!\!\!\perp B \mid \emptyset$ $\qquad$ $A \not\perp\!\!\!\perp B \mid \emptyset$

$A \not\perp\!\!\!\perp B \mid C$ $\qquad$ $A \perp\!\!\!\perp B \mid C \cup D$

$C \perp\!\!\!\perp D \mid A \cup B$

Both distributions can not be represented in the other framework (directed/undirected) with all conditional independence relations.

Using Graphical Models

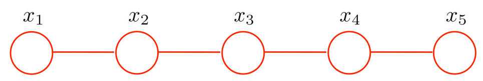
We can use a graphical model to do **inference**:

- Some nodes in the graph are **observed**, for others we want to find the posterior distribution
- Also, computing the local **marginal distribution** $p(x_n)$ at any node x_n can be done using inference.

Question: How can inference be done with a graphical model?

We will see that when exploiting conditional independences we can do efficient inference.

Inference on a Chain



The joint probability is given by

$$p(\mathbf{x}) = \frac{1}{Z}\psi_{1,2}(x_1,x_2)\psi_{2,3}(x_2,x_3)\psi_{3,4}(x_3,x_4)\psi_{4,5}(x_4,x_5)$$

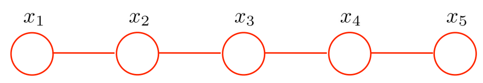
The marginal at x_3 is $\quad p(x_3) = \sum_{x_1}\sum_{x_2}\sum_{x_4}\sum_{x_5} p(\mathbf{x})$

In the general case with N nodes we have

$$p(\mathbf{x}) = \frac{1}{Z}\psi_{1,2}(x_1,x_2)\psi_{2,3}(x_2,x_3)\cdots\psi_{N-1,N}(x_{N-1},x_N)$$

and $\quad p(x_n) = \sum_{x_1}\cdots\sum_{x_{n-1}}\sum_{x_{n+1}}\cdots\sum_{x_N} p(\mathbf{x})$

Inference on a Chain



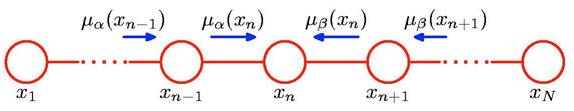
$$p(x_3) = \sum_{x_1}\sum_{x_2}\sum_{x_4}\sum_{x_5} p(\mathbf{x})$$

- This would mean K^N computations! A more efficient way is obtained by rearranging:

$$\begin{aligned}
p(x_3) &= \frac{1}{Z}\sum_{x_1}\sum_{x_2}\sum_{x_4}\sum_{x_5}\psi_{1,2}(x_1,x_2)\psi_{2,3}(x_2,x_3)\psi_{3,4}(x_3,x_4)\psi_{4,5}(x_4,x_5)\\
&= \frac{1}{Z}\cancel{\sum_{x_2}\sum_{x_1}}\sum_{x_4}\sum_{x_5}\psi_{1,2}(x_1,x_2)\psi_{2,3}(x_2,x_3)\psi_{3,4}(x_3,x_4)\psi_{4,5}(x_4,x_5)\\
&= \frac{1}{Z}\underbrace{\sum_{x_2}\psi_{2,3}(x_2,x_3)\sum_{x_1}\psi_{1,2}(x_1,x_2)}_{\mu_\alpha(x_3)}\underbrace{\sum_{x_4}\psi_{3,4}(x_3,x_4)\sum_{x_5}\psi_{4,5}(x_4,x_5)}_{\mu_\beta(x_3)}
\end{aligned}$$

$\mu_\alpha(x_3) \leftarrow$ Vectors of size K $\rightarrow \mu_\beta(x_3)$

Inference on a Chain



In general, we have

$$\begin{aligned}
p(x_n) &= \frac{1}{Z}\underbrace{\left[\sum_{x_{n-1}}\psi_{n-1,n}(x_{n-1},x_n)\cdots\left[\sum_{x_1}\psi_{1,2}(x_1,x_2)\right]\cdots\right]}_{\mu_\alpha(x_n)}\\
&\quad \underbrace{\left[\sum_{x_{n+1}}\psi_{n,n+1}(x_n,x_{n+1})\cdots\left[\sum_{x_N}\psi_{N-1,N}(x_{N-1},x_N)\right]\cdots\right]}_{\mu_\beta(x_n)}
\end{aligned}$$

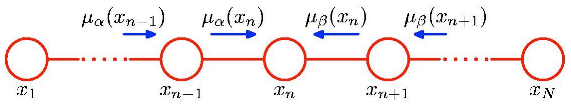
Inference on a Chain

The messages μ_α and μ_β can be computed recursively:

$$\begin{aligned}
\mu_\alpha(x_n) &= \sum_{x_{n-1}}\psi_{n-1,n}(x_{n-1},x_n)\left[\sum_{x_{n-2}}\cdots\right]\\
&= \sum_{x_{n-1}}\psi_{n-1,n}(x_{n-1},x_n)\mu_\alpha(x_{n-1}).\\
\mu_\beta(x_n) &= \sum_{x_{n+1}}\psi_{n,n+1}(x_n,x_{n+1})\left[\sum_{x_{n+2}}\cdots\right]\\
&= \sum_{x_{n+1}}\psi_{n,n+1}(x_n,x_{n+1})\mu_\beta(x_{n+1}).
\end{aligned}$$

Computation of μ_α starts at the first node and computation of μ_β starts at the last node.

Inference on a Chain



- The first values of μ_α and μ_β are:

$$\mu_\alpha(x_2) = \sum_{x_1} \psi_{1,2}(x_1, x_2) \qquad \mu_\beta(x_{N-1}) = \sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N)$$

- The partition function can be computed at any node:

$$Z = \sum_{x_n} \mu_\alpha(x_n)\mu_\beta(x_n)$$

- Overall, we have $O(NK^2)$ operations to compute the marginal $p(x_n)$

Inference on a Chain

To compute local marginals:

- Compute and store all forward messages, $\mu_\alpha(x_n)$.
- Compute and store all backward messages, $\mu_\beta(x_n)$
- Compute Z **once** at a node x_m: $Z = \sum_{x_m} \mu_\alpha(x_m)\mu_\beta(x_m)$
- Compute

$$p(x_n) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n)$$

for all variables required.

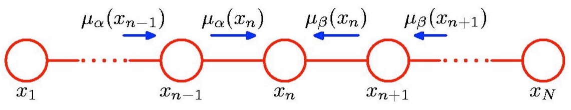
Summary

- Undirected Models (also known as Markov random fields) provide a simpler method to check for conditional independence
- A MRF is defined as a factorization over clique potentials and normalized globally
- Directed models can be converted into undirected ones, but there are distributions that can be represented only in one kind of model
- For undirected Markov chains there is a very efficient inference method based on message passing

Computer Vision Group
Prof. Daniel Cremers
Technische Universität München

4a. Inference in Graphical Models

Inference on a Chain (Rep.)



- The first values of μ_α and μ_β are:

$$\mu_\alpha(x_2) = \sum_{x_1} \psi_{1,2}(x_1, x_2) \qquad \mu_\beta(x_{N-1}) = \sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N)$$

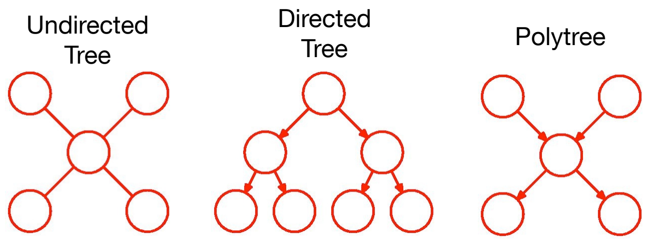
- The partition function can be computed at any node:

$$Z = \sum_{x_n} \mu_\alpha(x_n)\mu_\beta(x_n)$$

- Overall, we have $O(NK^2)$ operations to compute the marginal $p(x_n)$

More General Graphs

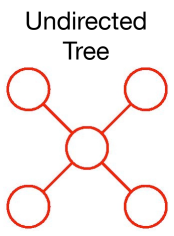
The message-passing algorithm can be extended to more general graphs:



It is then known as the **sum-product algorithm.**
A special case of this is **belief propagation**.

More General Graphs

The message-passing algorithm can be extended to more general graphs:

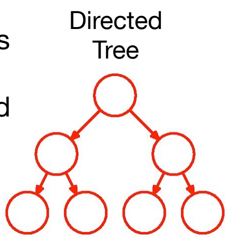


An **undirected tree** is defined as a graph that has exactly one path between any two nodes

More General Graphs

The message-passing algorithm can be extended to more general graphs:

A **directed tree** has only one node without parents and all other nodes have exactly one parent



Conversion from a directed to an undirected tree is no problem, because no links are inserted

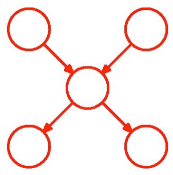
The same is true for the conversion back to a directed tree

More General Graphs

The message-passing algorithm can be extended to more general graphs:

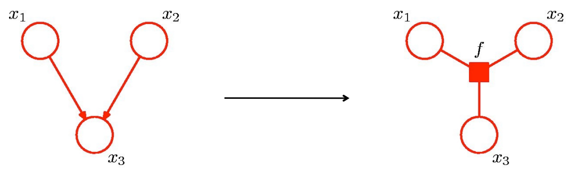
Polytrees can contain nodes with several parents, therefore moralization can remove independence relations

Polytree



Factor Graphs

- The Sum-product algorithm can be used to do inference on undirected and directed graphs.
- A representation that generalizes directed and undirected models is the **factor graph**.

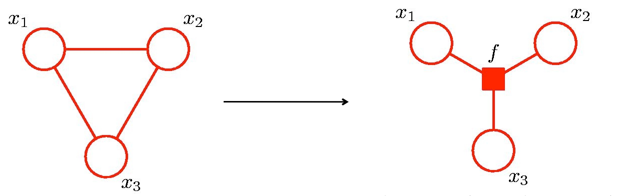


$p(\mathbf{x}) = p(x_1)p(x_2)p(x_3|x_1, x_2)$ $\quad$ $f(x_1, x_2, x_3) = p(x_1)p(x_2)p(x_3 \mid x_1, x_2)$

Directed graph $\quad$ Factor graph

Factor Graphs

- The Sum-product algorithm can be used to do inference on undirected and directed graphs.
- A representation that generalizes directed and undirected models is the **factor graph**.



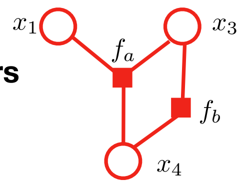
$\psi(x_1, x_2, x_3)$ $\quad$ $f(x_1, x_2, x_3) = \psi(x_1, x_2, x_3)$

Undirected graph $\quad$ Factor graph

Factor Graphs

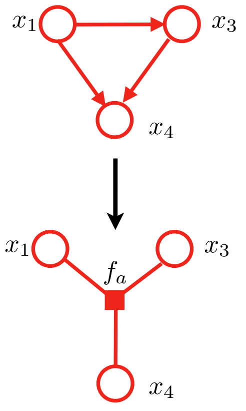
Factor graphs

- can contain **multiple factors** for the same nodes
- are more general than undirected graphs
- are **bipartite**, i.e. they consist of two kinds of nodes and all edges connect nodes of different kind



Factor Graphs

- Directed trees convert to tree-structured factor graphs
- The same holds for undirected trees
- Also: directed polytrees convert to tree-structured factor graphs
- And: Local cycles in a directed graph can be removed by converting to a factor graph



The Sum-Product Algorithm

Assumptions:

- all variables are discrete
- the factor graph has a tree structure

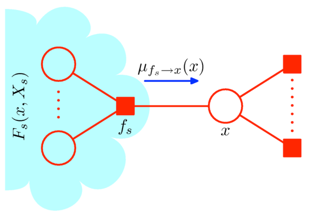
The factor graph represents the joint distribution as a product of factor nodes:

$$p(\mathbf{x}) = \prod_s f_s(\mathbf{x}_s)$$

The marginal distribution at a given node x is

$$p(x) = \sum_{\mathbf{x} \setminus x} p(\mathbf{x})$$

The Sum-Product Algorithm



For a given node x the joint can be written as

$$p(\mathbf{x}) = \prod_{s \in \text{ne}(x)} F_s(x, X_s)$$

Product of all factors associated with f_s

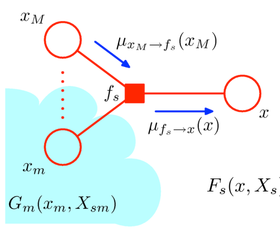
Thus, we have $p(x) = \sum_{\mathbf{x} \setminus x} \prod_{s \in \text{ne}(x)} F_s(x, X_s)$

Key insight: Sum and product can be exchanged!

$$p(x) = \prod_{s \in \text{ne}(x)} \sum_{X_s} F_s(x, X_s) = \prod_{s \in \text{ne}(x)} \mu_{f_s \to x}(x)$$

"Messages from factors to node x"

The Sum-Product Algorithm



The factors in the messages can be factorized further:

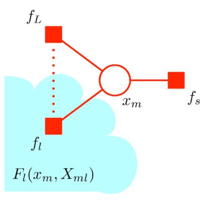
$$F_s(x, X_s) = f_s(x, x_1, \ldots, x_M) G_1(x_1, X_{s_1}) \ldots G_M(x_M, X_{s_M})$$

The messages can then be computed as

$$\mu_{f_s \to x}(x) = \sum_{x_1} \cdots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in \text{ne}(f_s) \setminus x} \sum_{X_{s_m}} G_m(x_m, X_{s_m})$$
$$= \sum_{x_1} \cdots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in \text{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)$$

"Messages from nodes to factors"

The Sum-Product Algorithm



The factors G of the neighboring nodes can again be factorized further:

$$G_M(x_m, X_{s_m}) = \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} F_l(x_m, X_{m_l})$$

This results in the exact same situation as above! We can now recursively apply the derived rules:

$$\mu_{x_m \to f_s}(x_m) = \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \sum_{X_{m_l}} F_l(x_m, X_{m_l})$$
$$= \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)$$

The Sum-Product Algorithm

Summary marginalization:

1. Consider the node x as a root note
2. Initialize the recursion at the leaf nodes as:
   $\mu_{f \to x}(x) = 1$ (var) or $\mu_{x \to f}(x) = f(x)$ (fac)
3. Propagate the messages from the leaves to the root x
4. Propagate the messages back from the root to the leaves
5. We can get the marginals at every node in the graph by multiplying all incoming messages

The Max-Sum Algorithm

Sum-product is used to find the marginal distributions at every node, but:

How can we find the setting of all variables that **maximizes** the joint probability? And what is the value of that maximal probability?

Idea: use sum-product to find all marginals and then report the value for each node x that maximizes the marginal $p(x)$

However: this does not give the **overall** maximum of the joint probability

The Max-Sum Algorithm

Observation: the max-operator is distributive, just like the multiplication used in sum-product:

$$\max(ab, ac) = a \max(b, c) \qquad \text{if} \qquad a \geq 0$$

Idea: use max instead of sum as above and exchange it with the product

Chain example:

$$\max_{\mathbf{x}} p(\mathbf{x}) = \frac{1}{Z} \max_{x_1} \dots \max [\psi_{1,2}(x_1, x_2) \dots \psi_{N-1,N}(x_{N-1}, x_N)]$$
$$= \frac{1}{Z} \max_{x_1} [\psi_{1,2}(x_1, x_2) [\dots \max \psi_{N-1,N}(x_{N-1}, x_N)]]$$

Message passing can be used as above!

The Max-Sum Algorithm

To find the maximum value of $p(\mathbf{x})$, we start again at the leaves and propagate to the root.

Two problems:

- no summation, but many multiplications; this leads to numerical instability (very small values)
- when propagating back, multiple configurations of $\mathbf{x}$ can maximize $p(\mathbf{x})$, leading to wrong assignments of the overall maximum

Solution to the first:

Transform everything into log-space and use sums

The Max-Sum Algorithm

Solution to the second problem:

Keep track of the arg max in the forward step, i.e. store at each node which value was responsible for the maximum:

$$\phi(x_n) = \arg\max_{x_{n-1}} [\ln f_{n-1,n}(x_{n-1}, x_n) + \mu_{x_{n-1} \to f_{n-1,n}}(x_n)]$$

Then, in the back-tracking step we can recover the arg max by recursive substitution of:

$$x_{n-1}^{\max} = \phi(x_n^{\max})$$

Other Inference Algorithms

Junction Tree Algorithm:

- Provides exact inference on general graphs.
- Works by turning the initial graph into a **junction tree** and then running a sum-product-like algorithm
- A junction tree is obtained from an undirected model by **triangulation** and mapping cliques to nodes and connections of cliques to edges
- It is the maximal spanning tree of cliques

Problem: Intractable on graphs with large cliques.

Cost grows exponentially with the number of variables in the largest clique ("tree width").

Other Inference Algorithms

Loopy Belief Propagation:

- Performs Sum-Product on general graphs, particularly when they have loops
- Propagation has to be done several times, until a convergence criterion is met
- No guarantee of convergence and no global optimum
- Messages have to be **scheduled**
- Initially, unit messages passed across all edges
- Approximate, but tractable for large graphs

Conditional Random Fields

- Another kind of undirected graphical model is known as Conditional Random Field (CRF).
- CRFs are used for classification where labels are represented as discrete random variables $\mathbf{y}$ and features as continuous random variables $\mathbf{x}$
- A CRF represents the conditional probability

$$p(\mathbf{y} \mid \mathbf{x}, \mathbf{w}) = \frac{\prod_C \phi_C(\mathbf{x}_C, \mathbf{y}_C; \mathbf{w})}{\sum_{\mathbf{y}'} \prod_C \phi_C(\mathbf{x}_C, \mathbf{y}'_C; \mathbf{w})}$$

  where $\mathbf{w}$ are parameters learned from training data.
- CRFs are **discriminative** and MRFs are **generative**

Conditional Random Fields

Derivation of the formula for CRFs:

$$p(\mathbf{y} \mid \mathbf{x}, \mathbf{w}) = \frac{p(\mathbf{y}, \mathbf{x} \mid \mathbf{w})}{p(\mathbf{x} \mid \mathbf{w})} = \frac{p(\mathbf{y}, \mathbf{x} \mid \mathbf{w})}{\sum_{y'} p(\mathbf{y}', \mathbf{x} \mid \mathbf{w})} = \frac{\prod_C \phi_C(\mathbf{x}_C, \mathbf{y}_C; \mathbf{w}) \quad Z}{Z \quad \sum_{\mathbf{y}'} \prod_C \phi_C(\mathbf{x}_C, \mathbf{y}'_C; \mathbf{w})}$$

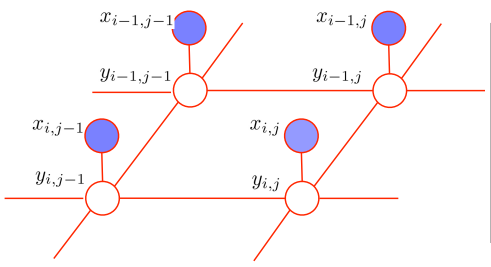
In the training phase, we compute parameters $\mathbf{w}$ that maximize the posterior:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} p(\mathbf{w} \mid \mathbf{x}^*, \mathbf{y}^*) \propto p(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{w}) p(\mathbf{w})$$

where *(x*,y*)* is the training data and *p(w)* is a Gaussian prior. In the inference phase we maximize

$$\arg\max_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x}, \mathbf{w}^*)$$

Conditional Random Fields



Typical example: **observed** variables $x_{i,j}$ are intensity values of pixels in an image and **hidden** variables $y_{i,j}$ are object labels

Note: the definition of $x_{i,j}$ and $y_{i,j}$ is different from the one in C.M. Bishop (pg.389)!

CRF Training

We minimize the negative log-posterior:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \{-\ln p(\mathbf{w} \mid \mathbf{x}^*, \mathbf{y}^*)\} = \arg\min_{\mathbf{w}} \{-\ln p(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{w}) - \ln p(\mathbf{w})\}$$

Computing the likelihood is intractable, as we have to compute the partition function for each $\mathbf{w}$. We can approximate the likelihood using **pseudo-likelihood**:

$$p(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{w}) \approx \prod_i p(y_i^* \mid \mathcal{M}(y_i^*), \mathbf{x}^*, \mathbf{w})$$

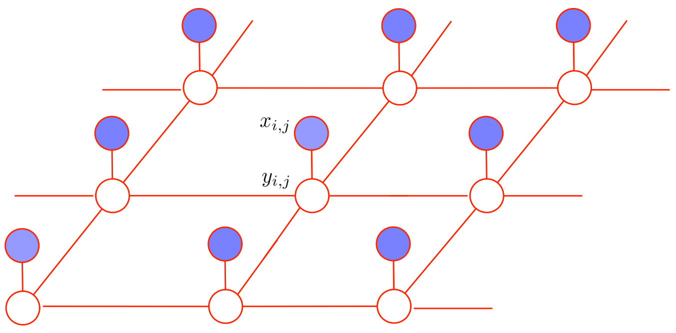
Markov blanket

C_i: All cliques containing y_i

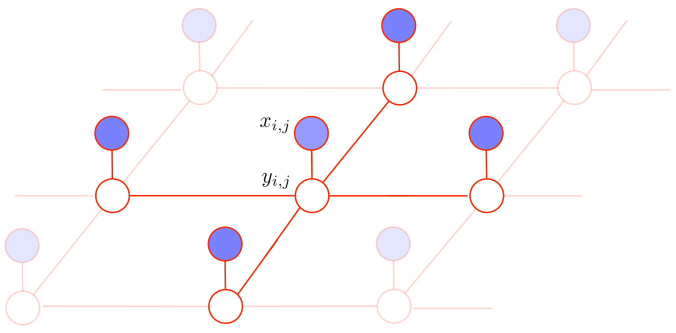
where

$$p(y_i^* \mid \mathcal{M}(y_i^*), \mathbf{x}^*, \mathbf{w}) = \frac{\prod_{C_i} \phi_{C_i}(\mathbf{x}^*_{C_i}, y_i^*, \mathbf{y}^*_{C_i}; \mathbf{w})}{\sum_{y'_i} \prod_{C_i} \phi_C(\mathbf{x}^*_{C_i}, y'_i, \mathbf{y}^*_{C_i}; \mathbf{w})}$$

Pseudo Likelihood



Pseudo Likelihood



Pseudo-likelihood is computed only on the Markov blanket of y_i and its corresp. feature nodes.

Potential Functions

- The only requirement for the potential functions is that they are positive. We achieve that with:

$$\phi_C(\mathbf{x}_C, \mathbf{y}_C, \mathbf{w}) := \exp(\mathbf{w}^T f(\mathbf{x}_C, \mathbf{y}_C))$$

  where f is a compatibility function that is large if the labels $\mathbf{y}_C$ fit well to the features $\mathbf{x}_C$.
- This is called the **log-linear model.**
- The function *f* can be, e.g. a local classifier

CRF Training and Inference

Training:

- Using pseudo-likelihood, training is efficient. We have to minimize:

$$L(\mathbf{w}) = \underbrace{-lpl(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{w})}_{\text{Log-pseudo-likelihood}} + \underbrace{\frac{1}{2\sigma^2}\mathbf{w}^T\mathbf{w}}_{\text{Gaussian prior}}$$

- This is a convex function that can be minimized using gradient descent

Inference:

- Only approximatively, e.g. using loopy belief propagation

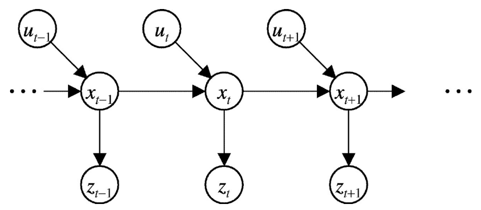
Summary

- Undirected Graphical Models represent conditional independence more intuitively using graph separation
- Their factorization is done based on potential functions The normalizer is called the partition function, which in general is intractable to compute
- Inference in graphical models can be done efficiently using the sum-product algorithm (message passing).
- Another inference algorithm is loopy belief propagation, which is approximate, but tractable
- Conditional Random Fields are a special kind of MRFs and can be used for classification

5. Hidden Markov Models

Graphical Representation (Rep.)

We can describe the overall process using a ***Dynamic Bayes Network***:

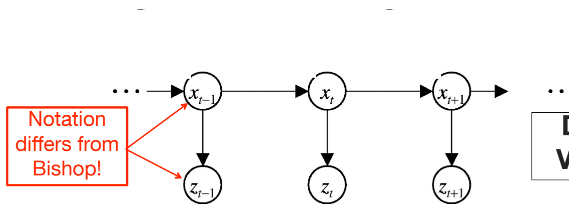


- This incorporates the following Markov assumptions:

$$p(z_t \mid x_{0:t}, u_{1:t}, z_{1:t}) = p(z_t \mid x_t) \text{ (measurement)}$$
$$p(x_t \mid x_{0:t-1}, u_{1:t}, z_{1:t}) = p(x_t \mid x_{t-1}, u_t) \text{ (state)}$$

Graphical Representation

We can describe the overall process using a ***Markov chain of latent variables***:



Discrete Variables

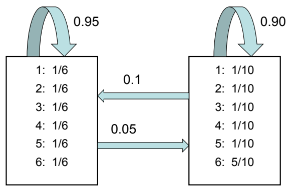
- This incorporates the following Markov assumptions:

$$p(z_t \mid x_{0:t}, \qquad z_{1:t}) = p(z_t \mid x_t) \text{ (measurement)}$$
$$p(x_t \mid x_{0:t-1}, \qquad z_{1:t}) = p(x_t \mid x_{t-1} \qquad) \text{ (state)}$$

Example

"Occasionally dishonest casino":

- observations: faces of a die $z_t \in \{1, 2, \dots, 6\}$
- hidden states: two different dice, one fair, one loaded



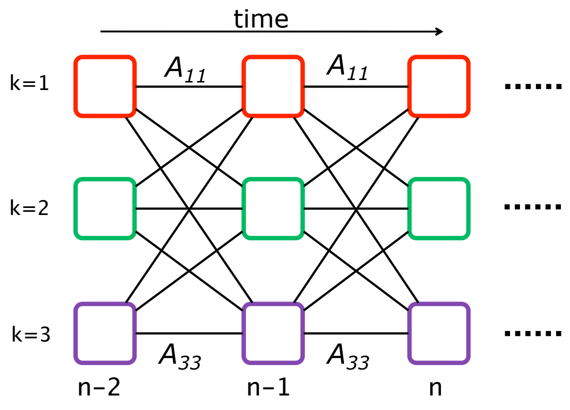
```
Rolls:  664153216162115234653214356634261655234232315142464156663246
Die:    LLLLLLLLLLLLLLFFFFFFLLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFLLLLLLLL
```

Formulation as HMM

1. Discrete random variables
   - Observation variables: $\{z_n\}$, n = 1..N
   - State variables (unobservable): $\{x_n\}$, n = 1..N
   - Number of states K: $x_n \epsilon \{1..K\}$
2. Transition model $p(x_i \mid x_{i-1})$
   - Markov assumption (x_i only depends on x_{i-1})
   - Represented as a $K \times K$ **transition matrix** A
   - Initial probability: $p(x_0)$ repr. as π_1, π_2, π_3
3. Observation model $p(z_i|x_i)$ with parameters φ
   - Observation only depends on the current state
   - Example: output of a "local" place classifier

Model Parameters θ

The Trellis Representation



Application Example (1)

- Given an observation sequence $z_1, z_2, z_3 \dots$
- Assume that the model parameters $\theta = (A, \pi, \varphi)$ are known
- What is the probability that the given observation sequence is actually observed under this model, i.e. $p(Z \mid \theta)$?
- If we are given several different models, we can choose the one with highest probability
- Expressed as a **supervised learning problem**, this can be interpreted as the inference step (classification step)

Application Example (2)

- Given an observation sequence $z_1, z_2, z_3 \ldots$
- Assume that the model parameters $\theta = (A, \pi, \varphi)$ are known
- What is the state sequence $x_1, x_2, x_3 \ldots$ that explains best the given observation sequence?
- In the case of place recognition: which is the sequence of truly visited places that explains best the sequence of obtained place labels (classifications)?

Application Example (3)

- Given an observation sequence $z_1, z_2, z_3 \ldots$
- What are the optimal model parameters $\theta = (\mathrm{A}, \pi, \varphi)$?
- This can be interpreted as the **training step**
- It is in general the most difficult problem

Summary: 3 Operations on HMMs

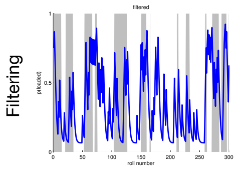
1. Compute data likelihood $p(Z|\theta)$ from a known model
   - Can be computed with the **forward-backward** algorithm
2. Compute optimal state sequence with a known model
   - Can be computed with the **Viterbi**-Algorithm
3. Learn model parameters for an observation sequence
   - Can be computed using **Expectation-Maximization** (or Baum-Welch)

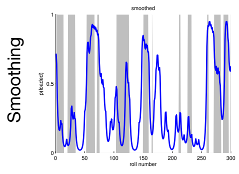
1. Computing the Data Likelihood

- Assume: given a state sequence $x_1, x_2, x_3 \ldots$

Two possible operations:

- **Filtering:** computes $p(x_t \mid \mathbf{z}_{1:t})$, i.e. state probability only based on previous observations
- **Smoothing:** computes $p(x_t \mid \mathbf{z}_{1:T})$, state probability based on **all** observations (including those from the future)





The Forward Algorithm

- First, we compute the prediction from the last time step:

$$p(x_t = j \mid \mathbf{z}_{1:t-1}) = \sum_i p(x_t = j \mid x_{t-1} = i) p(x_{t-1} = i \mid \mathbf{z}_{1:t-1})$$

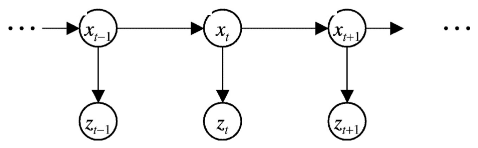
- Then, we do the update using Bayes rule:

$$\alpha_t(j) := p(x_t = j \mid \mathbf{z}_{1:t}) = p(x_t = j \mid z_t, \mathbf{z}_{1:t-1})$$
$$= \frac{1}{Z_t} p(\mathbf{z}_t \mid x_t = j, \cancel{\mathbf{z}_{1:t-1}}) p(x_t = j \mid \mathbf{z}_{1:t-1})$$

- This is exactly the same as the Bayes filter from the first lecture!

The Forward-Backward Algorithm

- As before we set $\alpha_t(j) := p(x_t = j \mid \mathbf{z}_{1:t})$
- We also define $\beta_t(j) := p(\mathbf{z}_{t+1:T} \mid x_t = j)$



$$\begin{tikzcd}
\cdots \arrow[r] & x_{t-1} \arrow[r] \arrow[d] & x_t \arrow[r] \arrow[d] & x_{t+1} \arrow[r] \arrow[d] & \cdots \\
 & z_{t-1} & z_t & z_{t+1} &
\end{tikzcd}$$

The Forward-Backward Algorithm

- As before we set $\alpha_t(j) := p(x_t = j \mid \mathbf{z}_{1:t})$
- We also define $\beta_t(j) := p(\mathbf{z}_{t+1:T} \mid x_t = j)$
- This can be recursively computed (backwards):

$$\beta_{t-1}(i) = p(\mathbf{z}_{t:T} \mid x_{t-1} = i)$$
$$= \sum_j p(x_t = j, z_t, \mathbf{z}_{t+1:T} \mid x_{t-1} = i)$$
$$= \sum_j p(\mathbf{z}_{t+1:T} \mid x_t = j, \cancel{x_{t-1} = i, z_t}) p(x_t = j, z_t \mid x_{t-1} = i)$$
$$= \sum_j p(\mathbf{z}_{t+1:T} \mid x_t = j) p(z_t \mid x_t = j, \cancel{x_{t-1} = i}) p(x_t = j \mid x_{t-1} = i)$$
$$= \sum_j \beta_t(j) p(z_t \mid x_t = j) p(x_t = j \mid x_{t-1} = i)$$

The Forward-Backward Algorithm

- As before we set $\alpha_t(j) := p(x_t = j \mid \mathbf{z}_{1:t})$
- We also define $\beta_t(j) := p(\mathbf{z}_{t+1:T} \mid x_t = j)$
- This can be recursively computed (backwards):

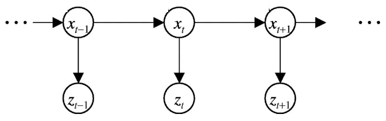
$$\beta_{t-1}(i) = p(\mathbf{z}_{t:T} \mid x_{t-1} = i)$$
$$= \sum_j \beta_t(j) p(z_t \mid x_t = j) p(x_t = j \mid x_{t-1} = i)$$

- This is exactly the same as the message-passing algorithm (sum-product)!
  - forward messages $\boldsymbol{\alpha}_t$ (vector of length K)
  - backward messages $\boldsymbol{\beta}_t$ (vector of length K)

2. Computing the Most Likely States

- Goal: find a state sequence $x_1, x_2, x_3 \ldots$ that maximizes the probability $\mathrm{p}(X,Z|\theta)$
- Define $\delta_t(j) := \max_{x_1,\ldots,x_{t-1}} p(\mathbf{x}_{1:t-1}, x_t = j \mid \mathbf{z}_{1:t})$

  This is the probability of state j by taking the most probable path.



2. Computing the Most Likely States

- Goal: find a state sequence $x_1, x_2, x_3 \ldots$ that maximizes the probability $\mathrm{p}(X,Z|\theta)$
- Define $\delta_t(j) := \max_{x_1,\ldots,x_{t-1}} p(\mathbf{x}_{1:t-1}, x_t = j \mid \mathbf{z}_{1:t})$

  This can be computed recursively:

$$\delta_t(j) := \max_i \delta_{t-1}(i) p(x_t \mid x_{t-1}) p(z_t \mid x_t)$$

  we also have to compute the argmax:

$$a_t(j) := \arg\max_i \delta_{t-1}(i) p(x_t \mid x_{t-1}) p(z_t \mid x_t)$$

The Viterbi algorithm

- Initialize:
  - $\delta(x_0) = \mathrm{p}(x_0)\, \mathrm{p}(z_0 \mid x_0)$
  - $\psi(x_0) = 0$
- Compute recursively for $n=1\ldots N$:
  - $\delta(x_n) = \mathrm{p}(z_n|x_n) \max_{x_{n-1}} [\delta(x_{n-1})\, \mathrm{p}(x_n|x_{n-1})]$
  - $a(x_n) = \operatorname{argmax}_{x_{n-1}} [\delta(x_{n-1})\, \mathrm{p}(x_n|x_{n-1})]$
- On termination:
  - $\mathrm{p}(Z,X|\theta) = \max_{x_N} \delta(x_N)$
  - $x^*_N = \operatorname{argmax}_{x_N} \delta(x_N)$
- Backtracking:
  - $x^*_n = a(x_{n+1})$

3. Learning the Model Parameters

- Given an observation sequence $z_1, z_2, z_3 \ldots$
- Find optimal model parameters θ
- We need to maximize the likelihood $\mathrm{p}(Z|\theta)$
- Can not be solved in closed form
- Iterative algorithm:
  Expectation Maximization (EM) or for the case of HMMs: Baum-Welch algorithm

Expectation Maximisation

- Objective: Find the model parameters knowing the observations: π,A,ϕ
- Result:
  - Train the HMM to recognize sequences of input
  - Train the HMM to generate sequences of input
- Technique: Expectation Maximisation
  - E: Find the best state sequence given the parameters
  - M: Find the parameters using the state sequence
  - Maximisation of the log-likelihood:

$$\arg\max_{pi,A,\phi} -\log\left(P\left(\left\{Z_i\right\}\ddot{\ }\pi, A\Omega\phi\right)\right)$$

The Baum-Welsh algorithm

- E-Step (assuming we know π,A,ϕ, i.e. θ^{old})
- Define the posterior probability of being in state i at step k:
- Define $\gamma(x_n) = \mathrm{p}(x_n|Z)$

The Baum-Welsh algorithm

- E-Step (assuming we know π,A,ϕ, i.e. θ^{old})
- Define the posterior probability of being in state i at step k:
- Define $\gamma(x_n) = \mathrm{p}(x_n|Z)$
- It follows that $\gamma(x_n) = \alpha(x_n)\, \beta(x_n) / \mathrm{p}(Z)$

The Baum-Welsh algorithm

- E-Step (assuming we know π,A,ϕ, i.e. θ^{old})
- Define the posterior probability of being in state i at step k:
- Define $\gamma(x_n) = \mathrm{p}(x_n|Z)$
- It follows that $\gamma(x_n) = \alpha(x_n)\, \beta(x_n) / \mathrm{p}(Z)$
- Define $\xi(x_{n-1}, x_n) = \mathrm{p}(x_{n-1}, x_n|Z)$
- It follows that
  $\xi(x_{n-1}, x_n) = \alpha(x_{n-1})\mathrm{p}(z_n|x_n)\mathrm{p}(x_n|x_{n-1})\beta(x_n) / \mathrm{p}(Z)$
- We need to compute:
  $Q(\theta, \theta^{\mathrm{old}}) = \sum_X \mathrm{p}(X|Z, \theta^{\mathrm{old}}) \log \mathrm{p}(Z,X|\theta)$

Expected complete data log-likelihood

The Baum-Welsh algorithm

- Maximizing Q also maximizes the likelihood: $p(Z|\theta) \geq p(Z|\theta^{old})$
- M-Step:
  - $$\pi_k = \frac{\sum_{\mathbf{x}} \gamma(\mathbf{x}) x_{1k}}{\sum_{j=1} \sum_{\mathbf{x}} \gamma(\mathbf{x}) x_{1j}}$$

  here, we need forward and backward step!
  - $$A_{jk} = \frac{\sum_{t=2}^{T} \xi(x_{t-1,j}, x_{tk})}{\sum_{l=1}^{K} \sum_{t=2}^{T} \xi(x_{t-1,j}, x_{tl})}$$
- With these new values, Q is recomputed
- This is done until the likelihood does not increase anymore (convergence)

The Baum-Welsh algorithm - summary

- Start with an initial estimate of $\theta=(\pi,A,\varphi)$ e.g. uniformly and k-means for φ
- Compute $Q(\theta,\theta^{old})$ (E-Step)
- Maximize Q (M-step)
- Iterate E and M until convergence
- In each iteration one full application of the forward-backward algorithm is performed
- Result gives a **local** optimum
- For other local optima, the algorithm needs to be started again with new initialization

The Scaling problem

- Probability of sequences

$$\prod_i \underbrace{p(x_i \mid ...)}_{<1} << 1$$

  - Probabilities are very small
  - The product of the terms soon is very small
- Usually: converting to log-space works
- But: we have sums of products!
- Solution: Rescale/Normalise the probability during the computation, e.g.:

$\hat{\alpha}(x_n) = \alpha(x_n) / p(z_1, z_2, \ldots, z_n)$

Summary

- HMMs are a way to model sequential data
- They assume discrete states
- Three possible operations can be performed with HMMs:
  - Data likelihood, given a model and an observation
  - Most likely state sequence, given a model and an observation
  - Optimal Model parameters, given an observation
- Appropriate scaling solves numerical problems
- HMMs are widely used, e.g. in speech recognition

5. Boosting

Repetition: Regression

We start with a set of **basis functions**

$$\phi(\mathbf{x}) = (\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \ldots, \phi_{M-1}(\mathbf{x})) \qquad \mathbf{x} \in \mathbb{R}^d$$

The goal is to fit a model into the data

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x})$$

To do this, we need to find an error function, e.g.:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} (\mathbf{w}^T \phi(\mathbf{x}_i) - t_i)^2$$

To find the optimal parameters, we derived E with respect to $\mathbf{w}$ and set the derivative to zero.

Some Questions

1. Can we do the same for classification? As a special case we consider two classes:
   $$t_i \in \{-1, 1\} \quad \forall i = 1, \ldots, N$$
2. Can we use a different (better?) error function?
3. Can we learn the basis functions together with the model parameters?
4. Can we do the learning sequentially, i.e. one basis function after another?

Answer to all questions: Yes, using Boosting!

The Loss Function

Definition: a real-valued function $L(t, y(\mathbf{x}))$, where t is a target value and y is a model, is called a **loss function**.

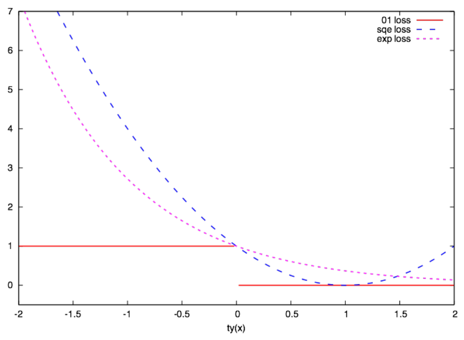
Examples:

01-loss: $$L_{01}(t, y(\mathbf{x})) = \begin{cases} 0 & \text{if } t = y(\mathbf{x}) \\ 1 & \text{else} \end{cases}$$

squared error loss: $$L_{sqe}(t, y(\mathbf{x})) = (t - y(\mathbf{x}))^2$$

exponential loss: $$L_{exp}(t, y(\mathbf{x})) = \exp(-ty(\mathbf{x}))$$

Loss Functions



- 01-loss is not differentiable
- squared error loss has only one optimum

Sequential Fitting of Basis Functions

Idea: We start with a basis function $\phi_0(\mathbf{x})$:

$$y_0(\mathbf{x}, w_0) = w_0\phi_0(\mathbf{x}) \qquad w_0 = 1$$

Then, at iteration m, we add a new basis function $\phi_m(\mathbf{x})$ to the model:

$$y_m(\mathbf{x}, w_0, \dots, w_m) = y_{m-1}(\mathbf{x}, w_0, \dots, w_{m-1}) + w_m\phi_m(\mathbf{x})$$

Two questions need to be answered:

1. How do we find a good new basis function?
2. How can we determine a good value for w_m?

Idea: Minimize the exponential loss function

Minimizing the Exponential Loss

Aim: find w_m and ϕ_m so that

$$(w_m, \phi_m) = \arg\min_{w,\phi} \sum_{i=1}^N L(t_i, y_{m-1}(\mathbf{x}_i) + w\phi(\mathbf{x}_i))$$

where $L(t, y) = \exp(-ty)$

Minimizing the Exponential Loss

Aim: find w_m and ϕ_m so that

$$(w_m, \phi_m) = \arg\min_{w,\phi} \sum_{i=1}^N L(t_i, y_{m-1}(\mathbf{x}_i) + w\phi(\mathbf{x}_i))$$

where $L(t, y) = \exp(-ty)$

Solution: $\phi_m = \arg\min_{\phi} \sum_{i=1}^N v_{i,m}\mathbb{I}(t_i \neq \phi(\mathbf{x}_i))$

Minimizing the Exponential Loss

Aim: find w_m and ϕ_m so that

$$(w_m, \phi_m) = \arg\min_{w,\phi} \sum_{i=1}^N L(t_i, y_{m-1}(\mathbf{x}_i) + w\phi(\mathbf{x}_i))$$

where $L(t, y) = \exp(-ty)$

Solution: $\phi_m = \arg\min_{\phi} \sum_{i=1}^N v_{i,m}\mathbb{I}(t_i \neq \phi(\mathbf{x}_i))$

$$w_m = \frac{1}{2}\log\frac{1 - \mathrm{err}_m}{\mathrm{err}_m}$$

Minimizing the Exponential Loss

Aim: find w_m and ϕ_m so that

$$(w_m, \phi_m) = \arg\min_{w,\phi} \sum_{i=1}^N L(t_i, y_{m-1}(\mathbf{x}_i) + w\phi(\mathbf{x}_i))$$

where $L(t, y) = \exp(-ty)$

Solution: $\phi_m = \arg\min_{\phi} \sum_{i=1}^N v_{i,m}\mathbb{I}(t_i \neq \phi(\mathbf{x}_i))$

$$w_m = \frac{1}{2}\log\frac{1 - \mathrm{err}_m}{\mathrm{err}_m} \qquad v_{i,m+1} = v_{i,m}\exp(2w_m\mathbb{I}(t_i \neq \phi_m(\mathbf{x}_i))$$

The AdaBoost Algorithm

1. For $i = 1, \dots, N$: $\quad v_i \leftarrow 1/N$
2. For $m = 1, \dots, M$

   Fit a classifier ("basis function") ϕ_m that minimizes

   $$\sum_{i=1}^N v_i\mathbb{I}(t_i \neq \phi_m(\mathbf{x}_i))$$

   Compute $\mathrm{err}_m = \frac{\sum_{i=1}^N v_i\mathbb{I}(t_i \neq \phi_m(\mathbf{x}_i))}{\sum_{i=1}^N v_i}$ and $\alpha_m = \log\frac{1 - \mathrm{err}_m}{\mathrm{err}_m}$

   Update the weights: $v_i \leftarrow v_i\exp(\alpha_m\mathbb{I}(t_i \neq \phi_m(\mathbf{x}_i)))$
3. Use the resulting classifier:

$$y(\mathbf{x}) = \mathrm{sgn}\sum_{m=1}^M \alpha_m\phi_m(\mathbf{x})$$

The "Basis Functions"

- Can be any classifier that can deal with weighted data
- Most importantly: if these "base classifiers" provide a training error that is at most as bad as a random classifier would give (i.e. it is a **weak** classifier), then AdaBoost can return an arbitrarily small training error (i.e. AdaBoost is a strong classifier)
- Many possibilities for weak classifiers exist, e.g.:
  - Decision stumps
  - Decision trees

Decision Stumps

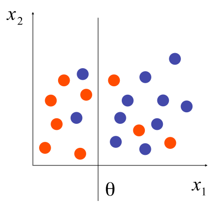
Decision Stumps are a kind of very simple weak classifiers.

Goal: Find an axis-aligned hyperplane that minimizes the class. error

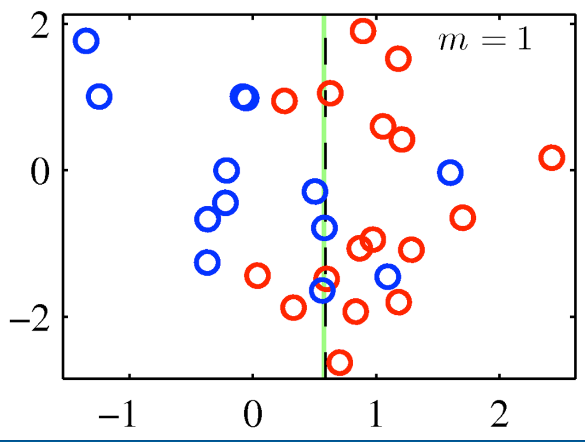
This can be done for each feature (i.e. for each dimension in feature space)

It can be shown that the classif. error is always better than 0.5 (random guessing)

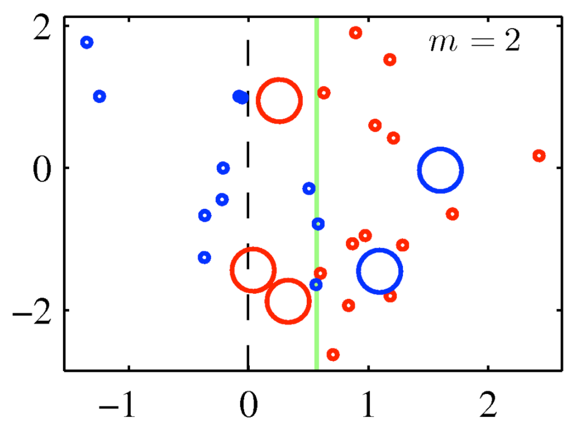
Idea: apply many weak classifiers, where each is trained on the misclassified examples of the previous.



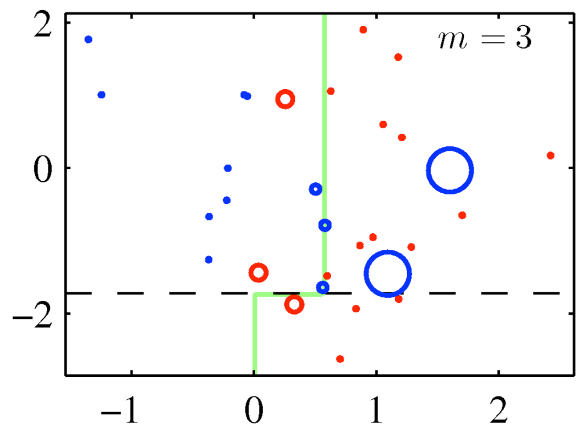
Classification Example



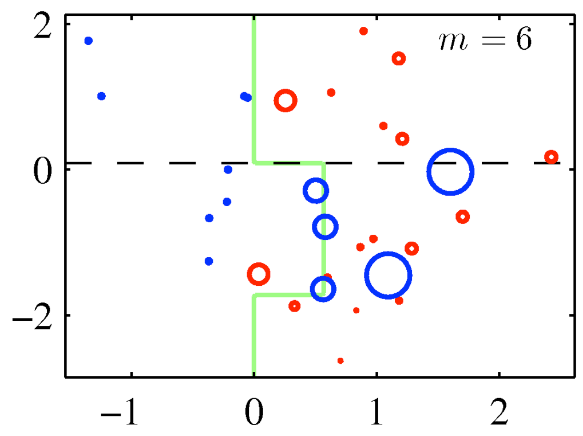
Classification Example



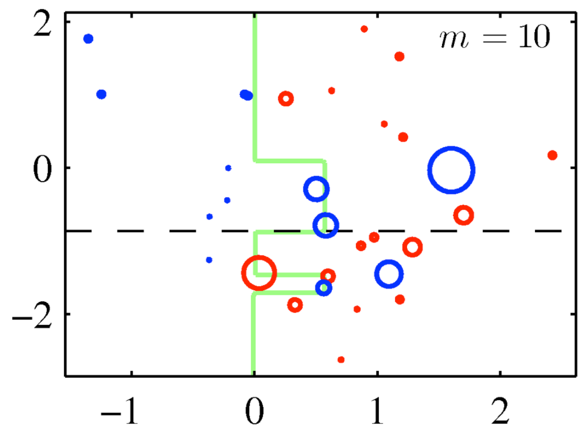
Classification Example



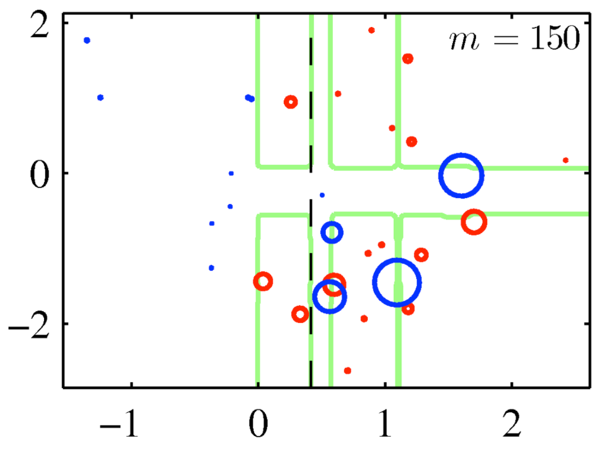
Classification Example



Classification Example

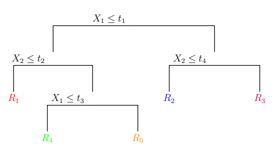


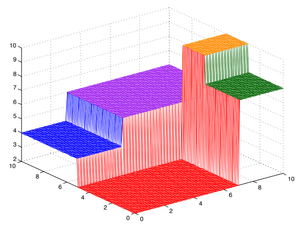
Classification Example



Decision Trees

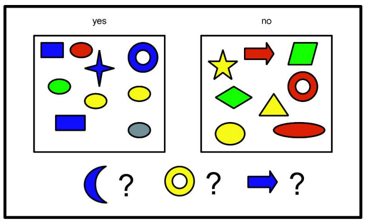
- A more general version of decision stumps are decision **trees:**

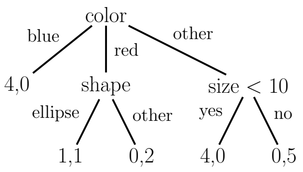




- At every node, a decision is made
- Dan be used for classification and for regression (Classification And Regression Trees CART)

Decision Trees for Classification





- Stores the distribution over class labels in each leaf (number of positives and negatives)
- With these, we can class label probabilities, e.g. $p(y = 1 \mid \mathbf{x}) = 1/2$ if we have a red ellipse

Growing a Decision Tree

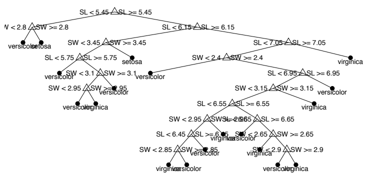
- Finding the optimal partition of the data is an NP-complete problem!
- Instead: use a greedy strategy:

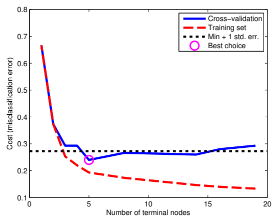
```
function fitTree(node, $\mathcal{D}$, depth):
1. node.prediction = class label distribution
2. $(j^*, t^*, \mathcal{D}_L, \mathcal{D}_R) = \text{split}(\mathcal{D})$
3. if not worth splitting then return node
4. node.test $\leftarrow x_{j^*} < t^*$
5. node.left = fitTree(node, $\mathcal{D}_L$, depth +1)
6. node.right = fitTree(node, $\mathcal{D}_R$, depth +1)
```

Growing a Decision Tree

- The Split-function finds an optimal feature and an optimal value for that feature
- For classification, it finds a split that minimizes some cost function, e.g. misclassification
- A decision stump is a decision tree with depth 1
- Stopping criteria for growing the tree are:
  - reduction of cost too small?
  - maximum depth reached?
  - is the distribution in the sub-trees homogenous?
  - is the number of samples in the sub-trees too small?

Tree Pruning





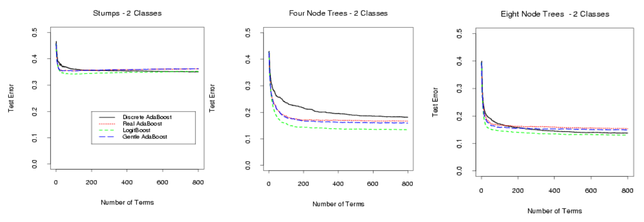
- If the tree grows too large, the algorithm overfits
- Simply stopping to grow can lead to situations where the tree is not expressive enough
- Idea: Build first full tree and then prune it
- Pruning can be done using cross-validation

Random Forests

- To reduce the variance of the classification estimate, we can train several trees on randomly sampled subsets of the data
- However, this can result in correlated classifiers, limiting the reduction in variance
- Idea: chose data subset and variable (feature) subset randomly
- The resulting algorithm is known as Random Forests
- Random Forests have very good accuracy and are widely used, e.g. body pose recognition

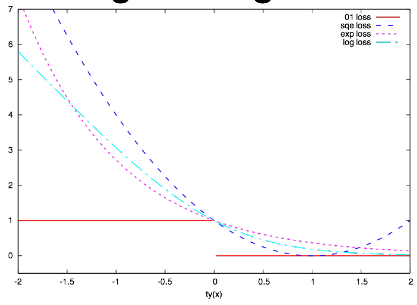
Back to Boosting

- AdaBoost has been shown to perform very well, especially when using decision trees as weak classifiers



- However: the exponential loss weighs misclassified examples very high!

Using the Log-Loss



- The log-loss is defined as:

$$L(t, y(\mathbf{x})) = \log_2(1 + \exp(-2ty(\mathbf{x})))$$

- It penalizes misclassifications only **linearly**

The LogitBoost Algorithm

1. For $i = 1, \dots, N$: $v_i \leftarrow 1/N \quad \pi_i \leftarrow 1/2$
2. For $m = 1, \dots, M$

   Compute the working response $z_i = \dfrac{t_i - \pi_i}{\pi_i(1 - \pi_i)}$

   Compute the weights $v_i = \pi_i(1 - \pi_i)$

   Find ϕ_m that minimizes

$$\sum_{i=1}^{N} v_i (z_i - \phi(\mathbf{x}_i))^2$$

   Update $y(\mathbf{x}) \leftarrow y(\mathbf{x}) + \frac{1}{2}\phi_m(\mathbf{x})$ and $\pi_i \leftarrow \dfrac{1}{1 + \exp(-2y(\mathbf{x}_i))}$

3. Use the resulting classifier:

$$y(\mathbf{x}) = \text{sgn} \sum_{m=1}^{M} \phi_m(\mathbf{x})$$

Weighted Least-Squares Regression

- Instead of a weak classifier, LogitBoost uses "weighted least-squares regression"
- This is very similar to standard least-squares regression:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N} v_i(\mathbf{w}^T\phi(\mathbf{x}_i) - t_i)^2$$

- This results in a matrix $\hat{\Phi} = V^{1/2}\Phi$ where

$$V^{1/2} = \mathrm{diag}(\sqrt{v_1}, \ldots, \sqrt{v_N})$$

- The solution is

$$\mathbf{w} = (\hat{\Phi}^T\hat{\Phi})^{-1}\hat{\Phi}^T\mathbf{t}$$

GentleBoost

```
Gentle AdaBoost
1. Start with weights $w_i = 1/N$, $i = 1, 2, \ldots, N$, $F(x) = 0$.
2. Repeat for $m = 1, 2, \ldots, M$:
    (a) Fit the regression function $f_m(x)$ by weighted least-squares of $y_i$ to $x_i$ with weights $w_i$.
    (b) Update $F(x) \leftarrow F(x) + f_m(x)$
    (c) Update $w_i \leftarrow w_i e^{-y_i f_m(x_i)}$ and renormalize.
3. Output the classifier $\mathrm{sign}[F(x)] = \mathrm{sign}[\sum_{m=1}^{M} f_m(x)]$
```

Algorithm 4: *A modified version of the Real AdaBoost algorithm, using Newton stepping rather than exact optimization at each step*

- Numerically more stable than LogitBoost
- Tends to perform better than AdaBoost and LogitBoost

Generalization: Gradient Boost

- Initialize

$$f_0(\mathbf{x}) = \arg\min_{\gamma}\sum_{i=1}^{N} L(t_i, \phi(\mathbf{x}_i, \gamma))$$

- for $m = 1, \ldots, M$
    - Compute the gradient residual

    $$r_{im} = -\left[\frac{\partial L(t_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\right]_{f(\mathbf{x}_i) = f_{m-1}(\mathbf{x}_i)}$$

    - Use the weak learner to compute that minimizes

    $$\sum_{i=1}^{N}(r_{im} - \phi(\mathbf{x}_i; \gamma_m))^2$$

    - Update $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \nu\phi(\mathbf{x}, \gamma)$
- Return $f(\mathbf{x}) = f_M(\mathbf{x})$

Application of AdaBoost: Face Detection

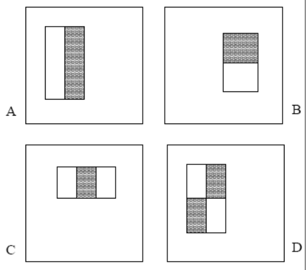
- The biggest impact of AdaBoost was made in face detection
- Idea: extract features ("Haar-like features") and train AdaBoost, use a cascade of classifiers
- Features can be computed very efficiently
- Weak classifiers can be decision stumps or decision trees
- As inference in AdaBoost is fast, the face detector can run in **real-time**!

Haar-like Features

- Defined as difference of rectangular integral area:
    - The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles.

$$\left(\iint_{White} I(x,y)dxdy\right) - \left(\iint_{Grey} I(x,y)dxdy\right)$$

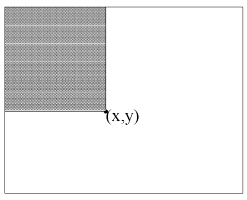
- One feature defined as:
    - Feature type: A,B,C or D
    - Feature position and size



The integral image

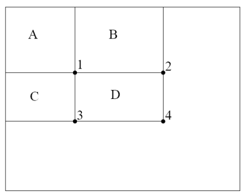
- Defined as :

$$I_{int}(X,Y) = \int_{x \le X}\int_{y \le Y} \mathrm{I(x,y)\ dy\ dx}$$



- Integral on rectangle D can be computed in 4 access to I_{int}:

$$\iint_D I(x,y) = I_{int}(4) + I_{int}(1) - I_{int}(2) - I_{int}(3)$$



- Very efficient way to compute features

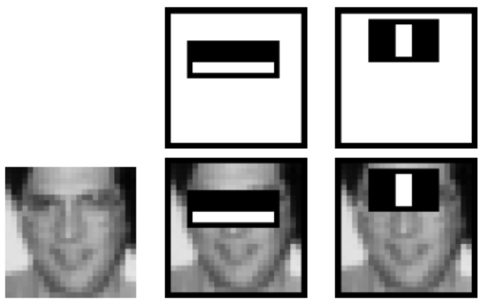
Weak Classifiers Used

- A weak classifier has 3 attributes:
    - A feature f_j (type, size and position)
    - A threshold θ_j
    - A comparison operator op_j = '<' or '>'
- The resulting weak classifier is:

$$h_j(x) = f_j(x) \quad op_j \quad \theta_j$$

- *x* is a 24x24 pixels window in the image

Two First Classifiers Selected by AdaBoost



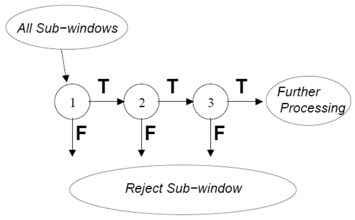
A classifier with only this two features can be trained to recognise 100% of the faces, with 40% of false positives

The Inference Algorithm

```
• scale = 24x24
• Do {
    • For each position in the image {
        • Try classifying the part of the image starting at this
          position, with the current scale, using the classifier
          selected by AdaBoost
      }
    • Scale = Scale x 1.5
  } until maximum scale
```

Another Improvement: the Cascade

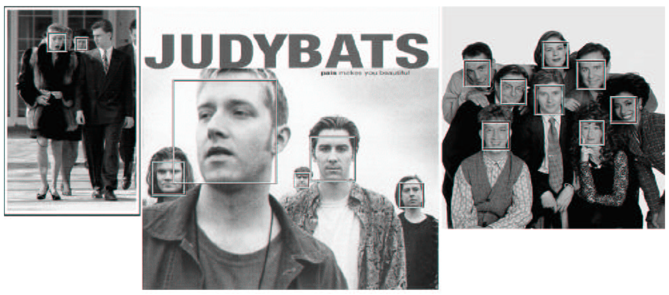
- Basic idea:
  - It is easy to detect that something is not a face
  - Tune(boost) classifier to be very reliable at saying NO (i.e. very low false negative)
  - Stop evaluating the cascade of classifier if one classifier says NO



Advantage of the Cascade

- Faster processing
  - Quick elimination of useless windows
- Each individual classifier is trained to deal only with the example that the previous ones could not process
  - Very specialised
- The deeper in the cascade, the more complex (the more features) in the classifiers.

Results (1)



Results (2)

Summary

- Boosting is a method to use a weak classifier and turn it into a strong one (arbitrarily small training error!)
- AdaBoost minimizes the exponential loss
- To be more robust against outliers, we can use LogitBoost or GentleBoost
- Weak learners can be decision stumps or decision trees
- Face detection can be solved with Boosting

6. Kernel Methods

Motivation

- Usually learning algorithms assume that some kind of feature function is given
- Reasoning is then done on a feature vector of a given (finite) length
- But: some objects are hard to represent with a fixed-size feature vector, e.g. text documents, molecular structures, evolutionary trees
- Idea: use a way of measuring similarity without the need of features, e.g. the edit distance for strings
- This we will call a **kernel function**

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^T\phi(\mathbf{x}_n) - t_n)^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \qquad \phi(\mathbf{x}_n) \in \mathbb{R}^D$$

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^T\phi(\mathbf{x}_n) - t_n)^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \qquad \phi(\mathbf{x}_n) \in \mathbb{R}^D$$

if we write this in vector form, we get

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}^T\Phi^T\mathbf{t} + \mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \qquad \mathbf{t} \in \mathbb{R}^N$$

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^T\phi(\mathbf{x}_n) - t_n)^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \qquad \phi(\mathbf{x}_n) \in \mathbb{R}^D$$

if we write this in vector form, we get

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}^T\Phi^T\mathbf{t} + \mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \qquad \mathbf{t} \in \mathbb{R}^N$$

and the solution is

$$\mathbf{w} = (\Phi^T\Phi + \lambda I_D)^{-1}\Phi^T\mathbf{t}$$

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}^T\Phi^T\mathbf{t} + \mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

$$\mathbf{w} = (\Phi^T\Phi + \lambda I_D)^{-1}\Phi^T\mathbf{t}$$

However, we can express this result in a different way using the **matrix inversion lemma:**

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}$$

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}^T\Phi^T\mathbf{t} + \mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

$$\mathbf{w} = (\Phi^T\Phi + \lambda I_D)^{-1}\Phi^T\mathbf{t}$$

However, we can express this result in a different way using the **matrix inversion lemma:**

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}$$

$$\mathbf{w} = \Phi^T(\Phi\Phi^T + \lambda I_N)^{-1}\mathbf{t}$$

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}^T\Phi^T\mathbf{t} + \mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

$$\mathbf{w} = (\Phi^T\Phi + \lambda I_D)^{-1}\Phi^T\mathbf{t}$$

$$\mathbf{w} = \Phi^T\underbrace{(\Phi\Phi^T + \lambda I_N)^{-1}\mathbf{t}}_{=:\,\mathbf{a}}$$

"Dual Variables"

Plugging $\mathbf{w} = \Phi^T\mathbf{a}$ into $J(\mathbf{w})$ gives:

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T\Phi\Phi^T\Phi\Phi^T\mathbf{a} - \mathbf{a}^T\Phi\Phi^T\mathbf{t} + \mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^T\Phi\Phi^T\mathbf{a}$$

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}^T\Phi^T\mathbf{t} + \mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^TKK\mathbf{a} - \mathbf{a}^TK\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^TK\mathbf{a} \qquad K = \Phi\Phi^T$$

This is called the **dual formulation**.

Note: $\mathbf{a} \in \mathbb{R}^N \qquad \mathbf{w} \in \mathbb{R}^D$

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}^T\Phi^T\mathbf{t} + \mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^TKK\mathbf{a} - \mathbf{a}^TK\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^TK\mathbf{a}$$

This is called the **dual formulation**.

The solution to the dual problem is:

$$\mathbf{a} = (K + \lambda I_N)^{-1}\mathbf{t}$$

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}^T\Phi^T\mathbf{t} + \mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T KK\mathbf{a} - \mathbf{a}^T K\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^T K\mathbf{a}$$

$$\mathbf{a} = (K + \lambda I_N)^{-1}\mathbf{t}$$

This we can use to make **predictions**:

$$y(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) = \mathbf{a}^T\Phi\phi(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T(K + \lambda I_N)^{-1}\mathbf{t}$$

(now $\mathbf{x}$ is unknown and $\mathbf{a}$ is given from training)

Dual Representation

$$y(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T(K + \lambda I_N)^{-1}\mathbf{t}$$

where:

$$\mathbf{k}(\mathbf{x}) = \begin{pmatrix} \phi(\mathbf{x}_1)^T\phi(\mathbf{x}) \\ \vdots \\ \phi(\mathbf{x}_N)^T\phi(\mathbf{x}) \end{pmatrix} \quad K = \begin{pmatrix} \phi(\mathbf{x}_1)^T\phi(\mathbf{x}_1) & \dots & \phi(\mathbf{x}_1)^T\phi(\mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ \phi(\mathbf{x}_N)^T\phi(\mathbf{x}_1) & \dots & \phi(\mathbf{x}_N)^T\phi(\mathbf{x}_N) \end{pmatrix}$$

Thus, y is expressed only in terms of **dot products** between different pairs of $\phi(\mathbf{x})$, or in terms of the **kernel function**

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$$

Representation using the Kernel

$$y(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T(K + \lambda I_N)^{-1}\mathbf{t}$$

Now we have to invert a matrix of size $N \times N$, before it was $M \times M$ where $M < N$, but:

By expressing everything with the kernel function, we can deal with very high-dimensional or even **infinite**-dimensional feature spaces!

Idea: Don't use features at all but simply define a similarity function expressed as the kernel!

Constructing Kernels

The straightforward way to define a kernel function is to first find a basis function $\phi(\mathbf{x})$ and to define:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$$

This means, k is an inner product in some space $\mathcal{H}$, i.e:

1. Symmetry: $k(\mathbf{x}_i, \mathbf{x}_j) = \langle\phi(\mathbf{x}_j), \phi(\mathbf{x}_i)\rangle = \langle\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle$
2. Linearity: $\langle a(\phi(\mathbf{x}_i) + \mathbf{z}), \phi(\mathbf{x}_j)\rangle = a\langle\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle + a\langle\mathbf{z}, \phi(\mathbf{x}_j)\rangle$
3. Positive definite: $\langle\phi(\mathbf{x}_i), \phi(\mathbf{x}_i)\rangle \geq 0$, equal if $\phi(\mathbf{x}_i) = \mathbf{0}$

Can we find conditions for k under which there is a (possibly infinite dimensional) basis function into $\mathcal{H}$, where k is an inner product?

Constructing Kernels

Theorem (Mercer): If k is

1. symmetric, i.e. $k(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_j, \mathbf{x}_i)$ and
2. positive definite, i.e.

$$K = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & \dots & k(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix}$$ "Gram Matrix"

is positive definite, then there exists a mapping $\phi(\mathbf{x})$ into a feature space $\mathcal{H}$ so that k can be expressed as an inner product in $\mathcal{H}$.

This means, we don't need to find $\phi(\mathbf{x})$ explicitly!

We can directly work with k "Kernel Trick"

Constructing Kernels

Finding valid kernels from scratch is hard, but:

A number of rules exist to create a new valid kernel k from given kernels k_1 and k_2. For example:

$$k(\mathbf{x}_1, \mathbf{x}_2) = ck_1(\mathbf{x}_1, \mathbf{x}_2), \quad c > 0$$

$$k(\mathbf{x}_1, \mathbf{x}_2) = f(\mathbf{x}_1)k_1(\mathbf{x}_1, \mathbf{x}_2)f(\mathbf{x}_2)$$

$$k(\mathbf{x}_1, \mathbf{x}_2) = \exp(k_1(\mathbf{x}_1, \mathbf{x}_2))$$

$$k(\mathbf{x}_1, \mathbf{x}_2) = k_1(\mathbf{x}_1, \mathbf{x}_2) + k_2(\mathbf{x}_1, \mathbf{x}_2)$$

$$k(\mathbf{x}_1, \mathbf{x}_2) = k_1(\mathbf{x}_1, \mathbf{x}_2)k_2(\mathbf{x}_1, \mathbf{x}_2)$$

$$k(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^T A\mathbf{x}_2$$ where A is positive semidefinite and symmetric

Examples of Valid Kernels

- Polynomial Kernel:

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T\mathbf{x}_j + c)^d \quad c > 0 \quad d \in \mathbb{N}$$

- Gaussian Kernel:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma^2)$$

- Kernel for sets:

$$k(A_1, A_2) = 2^{|A_1 \cap A_2|}$$

- Matern kernel:

$$k(r) = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}r}{l}\right)^\nu K_\nu\left(\frac{\sqrt{2\nu}r}{l}\right) \quad r = \|\mathbf{x}_i - \mathbf{x}_j\|, \nu > 0, l > 0$$

A Simple Example

Define a kernel function as

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}')^2 \qquad \mathbf{x}, \mathbf{x}' \in \mathbb{R}^2$$

This can be written as:

$$(x_1x_1' + x_2x_2')^2 = x_1^2x_1'^2 + 2x_1x_1'x_2x_2' + x_2^2x_2'^2$$

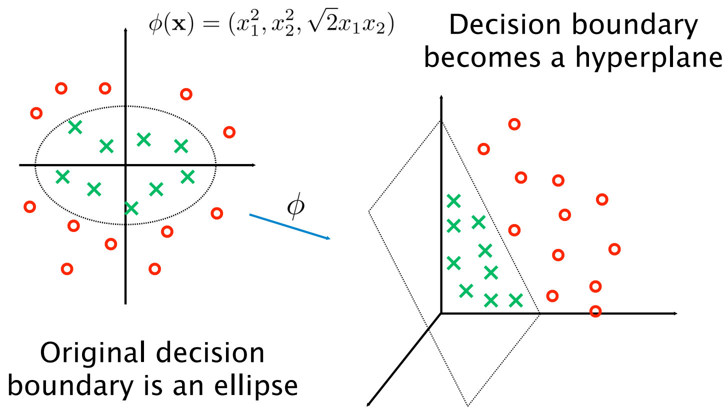
$$= (x_1^2, x_2^2, \sqrt{2}x_1x_2)(x_1'^2, x_2'^2, \sqrt{2}x_1'x_2')^T$$

$$= \phi(\mathbf{x})^T\phi(\mathbf{x}')$$

It can be shown that this holds in general for

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T\mathbf{x}_j)^d$$

Visualization of the Example



$\phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$
Decision boundary becomes a hyperplane
ϕ
Original decision boundary is an ellipse

Application Examples

Kernel Methods can be applied for many different problems, e.g.:

- Density estimation (unsupervised learning)
- Regression
- Principal Component Analysis (PCA)
- Classification

Most important Kernel Methods are

- Support Vector Machines
- Gaussian Processes

Kernelization

- Many existing algorithms can be converted into kernel methods
- This process is called “kernelization”

Idea:

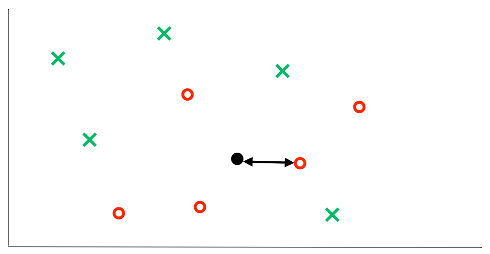
- express similarities of data points in terms of an inner product (dot product)
- replace all occurrences of that inner product by the kernel function

This is called the **kernel trick**

Example: Nearest Neighbor

- The NN classifier selects the label of the nearest neighbor in Euclidean distance

$$\|\mathbf{x}_i, \mathbf{x}_j\|^2 = \mathbf{x}_i^T\mathbf{x}_i + \mathbf{x}_j^T\mathbf{x}_j + 2\mathbf{x}_i^T\mathbf{x}_j$$



Example: Nearest Neighbor

- The NN classifier selects the label of the nearest neighbor in Euclidean distance

$$\|\mathbf{x}_i, \mathbf{x}_j\|^2 = \mathbf{x}_i^T\mathbf{x}_i + \mathbf{x}_j^T\mathbf{x}_j + 2\mathbf{x}_i^T\mathbf{x}_j$$

- We can now replace the dot products by a valid Mercer kernel and we obtain:

$$d(\mathbf{x}_i, \mathbf{x}_j)^2 = k(\mathbf{x}_i, \mathbf{x}_i) + k(\mathbf{x}_j, \mathbf{x}_j) + 2k(\mathbf{x}_i, \mathbf{x}_j)$$

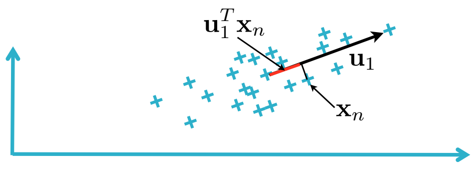
- This is a kernelized nearest-neighbor classifier
- We do not explicitly compute feature vectors!

Example: Principal Component Analysis

- Given: data set $\{\mathbf{x}_n\} \quad n = 1, \ldots, N \quad \mathbf{x}_n \in \mathbb{R}^D$
- Project data onto a subspace of dimension M so that the variance is maximized (“decorrelation”)
- For now: assume M is equal to 1
- Thus: the subspace can be described by a D-dimensional unit vector $\mathbf{u}_1$, i.e.: $\mathbf{u}_1^T\mathbf{u}_1 = 1$
- Each data point is projected onto the subspace using the dot product: $\mathbf{u}_1^T\mathbf{x}_n$

Principal Component Analysis

Visualization:



Mean:

$$\mu = \frac{1}{N}\sum_{n=1}^{N}\mathbf{u}_1^T\mathbf{x}_n = \frac{1}{N}\mathbf{u}_1^T\sum_{n=1}^{N}\mathbf{x}_n = \mathbf{u}_1^T\bar{\mathbf{x}}$$

Variance:

$$\sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{u}_1^T\mathbf{x}_n - \mathbf{u}_1^T\bar{\mathbf{x}})^2 = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{u}_1^T(\mathbf{x}_n - \bar{\mathbf{x}}))^2 = \mathbf{u}_1^T\underbrace{\frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T}_{S}\mathbf{u}_1$$

Principal Component Analysis

Goal: Maximize $\mathbf{u}_1^T S\mathbf{u}_1$ s.t. $\mathbf{u}_1^T\mathbf{u}_1 = 1$

Using a Lagrange multiplier:

S symmetric

$$\mathbf{u}^* = \arg\max_{\mathbf{u}_1} \mathbf{u}_1^T S\mathbf{u}_1 + \lambda_1(1 - \mathbf{u}_1^T\mathbf{u}_1)$$

Setting the derivative wrt. $\mathbf{u}_1$ to 0 we obtain:

$$S\mathbf{u}_1 = \lambda_1\mathbf{u}_1$$

Thus: $\mathbf{u}_1$ must be an eigenvector of S.
Multiplying with $\mathbf{u}_1^T$ from left gives: $\mathbf{u}_1^T S\mathbf{u}_1 = \lambda_1$

Thus: σ^2 is largest if $\mathbf{u}_1$ is the eigenvector of the largest eigenvalue of S

Principal Component Analysis

We can continue to find the best one-dimensional subspace that is orthogonal to $\mathbf{u}_1$

If we do this M times we obtain:

$\mathbf{u}_1, \dots, \mathbf{u}_M$ are the eigenvectors of the M largest eigenvalues of S: $\lambda_1, \dots, \lambda_M$

To project the data onto the M-dimensional subspace we use the dot-product:

$$\mathbf{x}^\perp = \begin{pmatrix} \mathbf{u}_1^T \\ \vdots \\ \mathbf{u}_M^T \end{pmatrix} (\mathbf{x} - \bar{\mathbf{x}})$$

Reconstruction using PCA

- We can interpret the vectors $\mathbf{u}_1, \dots, \mathbf{u}_M$ as a basis if $M = D$
- A reconstruction of a data point $\mathbf{x}$ into an M-dimensional subspace ($M<D$) can be written:

$$\tilde{\mathbf{x}}_n = \sum_{i=1}^{M} z_{ni}\mathbf{u}_i + \sum_{i=M+1}^{D} b_i \mathbf{u}_i$$

- Goal is to minimize the squared error:

$$J = \frac{1}{N} \sum_{n=1} \|\mathbf{x}_n - \tilde{\mathbf{x}}_n\|^2$$

- This results in:

$$z_{ni} = \mathbf{x}_n^T \mathbf{u}_i \qquad b_i = \bar{\mathbf{x}}^T \mathbf{u}_i$$

These are the coefficients of the eigenvectors

Reconstruction using PCA

Plugging in, we have:

$$\begin{aligned}
\tilde{\mathbf{x}}_n &= \sum_{i=1}^{M} (\mathbf{x}_n^T \mathbf{u}_i)\mathbf{u}_i + \sum_{i=M+1}^{D} (\bar{\mathbf{x}}^T \mathbf{u}_i)\mathbf{u}_i \\
&= \sum_{i=1}^{D} (\bar{\mathbf{x}}^T \mathbf{u}_i)\mathbf{u}_i - \sum_{i=1}^{M} (\bar{\mathbf{x}}^T \mathbf{u}_i)\mathbf{u}_i + \sum_{i=1}^{M} (\mathbf{x}_n^T \mathbf{u}_i)\mathbf{u}_i \\
&= \bar{\mathbf{x}} + \sum_{i=1}^{M} (\mathbf{x}_n^T \mathbf{u}_i - \bar{\mathbf{x}}^T \mathbf{u}_i)\mathbf{u}_i \\
&= \bar{\mathbf{x}} + \sum_{i=1}^{M} ((\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_i)\mathbf{u}_i
\end{aligned}$$

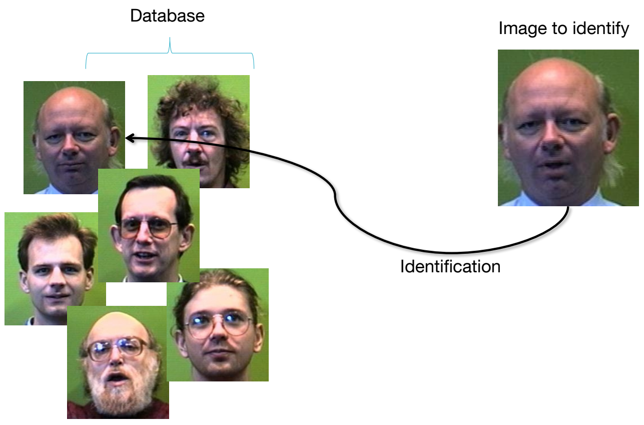
4. Add mean

3. Back-project

1. Substract mean

2. Project onto first M eigenvectors

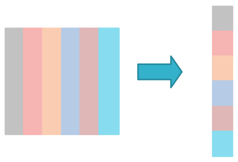
Application of PCA: Face Recognition



Application of PCA: Face Recognition

Approach:

- Convert the image into a nm vector by stacking the columns:

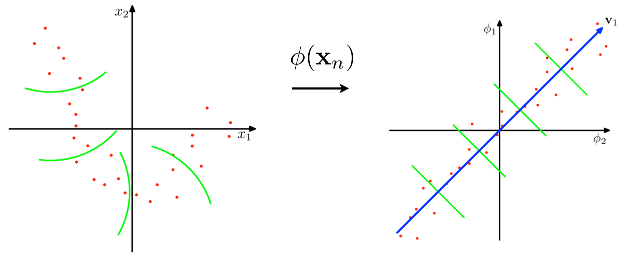


- A small image is 100x100 -> a 10000 element vector, i.e. a point in a 10000 dimension space
- Then compute covariance matrix and eigenvectors
- Select number of dimensions in subspace
- Find nearest neighbor in subspace for a new image

Results of Face Recognition

- 30% of faces used for testing, 70% for learning.

Can We Use Kernels in PCA?



- What if data is distributed along non-linear principal components?
- **Idea:** Use non-linear kernel to map into a space where PCA can be done

Kernel PCA

Here, assume that the mean of the data is zero:

$$\sum_{n=1}^{N} \mathbf{x}_n = \mathbf{0}$$

Then, in standard PCA we have the eigenvalue problem:

$$S\mathbf{u}_i = \lambda_i \mathbf{u}_i \qquad S = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^T$$

Now, we use a non-linear transformation $\phi(\mathbf{x}_n)$ and we assume $\sum_{n=1}^{N} \phi(\mathbf{x}_n) = \mathbf{0}$. We define C as

$$C = \frac{1}{N}\sum_{n=1}^{N} \phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T \text{ , with } C\mathbf{v}_i = \lambda_i \mathbf{v}_i$$

Goal: find eigenvalues without using features!

Kernel PCA

Plugging in: $$\frac{1}{N}\sum_{n=1}^{N}\phi(\mathbf{x}_n)\underbrace{\phi(\mathbf{x}_n)^T\mathbf{v}_i}_{\in\mathbb{R}} = \lambda_i\mathbf{v}_i$$

This means, there are values a_{in} so that $\mathbf{v}_i = \sum_{i=1}^{N} a_{in}\phi(\mathbf{x}_n)$.
With this we have:

$$\frac{1}{N}\sum_{n=1}^{N}\phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T\sum_{m=1}^{N}a_{im}\phi(\mathbf{x}_m) = \lambda_i\sum_{i=1}^{N}a_{in}\phi(\mathbf{x}_n)$$

Multiplying both sides by $\phi(\mathbf{x}_l)$ gives:

$$\frac{1}{N}\sum_{n=1}^{N}k(\mathbf{x}_l,\mathbf{x}_n)\sum_{m=1}^{N}a_{im}k(\mathbf{x}_n,\mathbf{x}_m) = \lambda_i\sum_{i=1}^{N}a_{in}k(\mathbf{x}_l,\mathbf{x}_n)$$

where $k(\mathbf{x}_l,\mathbf{x}_n) = \phi(\mathbf{x}_l)^T\phi(\mathbf{x}_n)$. This is our expression in terms of the kernel function!

Kernel PCA

The problem can be cast as finding eigenvectors of the kernel matrix K:

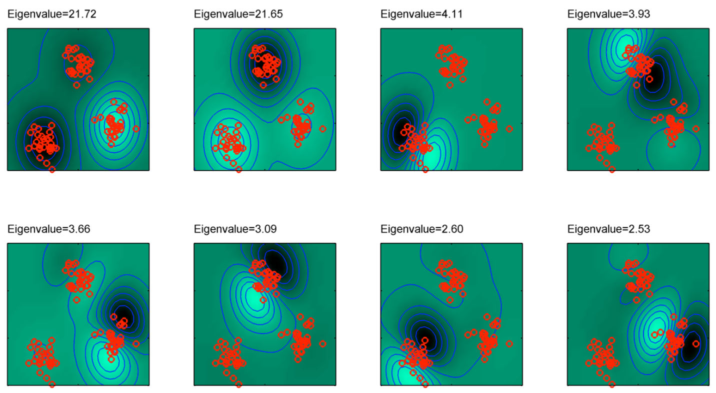
$$K\mathbf{a}_i = \lambda_i N\mathbf{a}_i$$

With this, we can find the projection of the image of $\mathbf{x}$ onto a given principal component as:

$$\phi(\mathbf{x})^T\mathbf{v}_i = \sum_{n=1}^{N}a_{in}\phi(\mathbf{x})^T\phi(\mathbf{x}_n) = \sum_{n=1}^{N}a_{in}k(\mathbf{x},\mathbf{x}_n)$$

Again, this is expressed in terms of the kernel function.

Kernel PCA: Example



Example: Classification

- We have seen kernel methods for density estimation, PCA and regression
- For classification there are two major kernel methods: Support Vector Machines (SVMs) and Gaussian Processes
- SVMs are probably the most used classification algorithm
- Main idea: use kernelisation to map into a high-dimensional feature space, where a linear separation between the classes can be found (“hyper-plane”)

Support Vector Machines

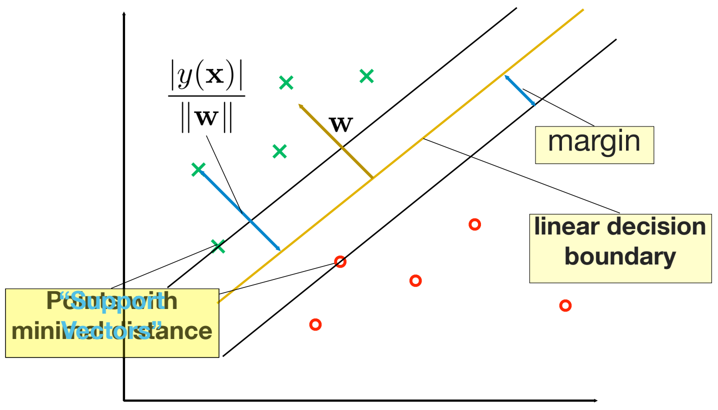
Support Vector Machines learn a linear discriminant function (“hyper-planes”):

$$y(\mathbf{x},\mathbf{w}) = \mathbf{w}^T\phi(\mathbf{x}) - b$$

parameters of the hyperplane (normal vector) — feature function — data point — Bias parameter

Assumptions for now: Data is linearly separable, Binary classification ($t_i \in \{-1;+1\}$).
“Maximum Margin”: find the decision boundary that maximizes the distance to the closest data point

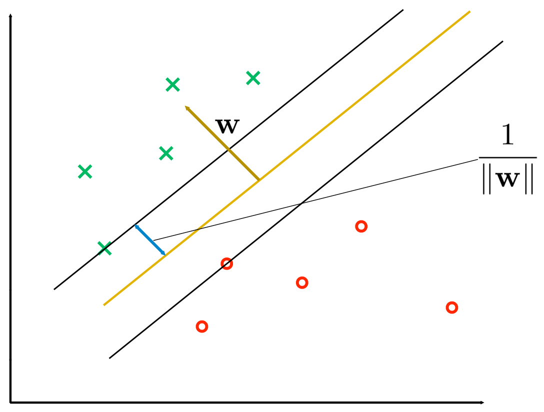
Maximum Margin


$\frac{|y(\mathbf{x})|}{\|\mathbf{w}\|}$
$\mathbf{w}$
margin
linear decision boundary
Point with minimal distance
“Support Vectors”

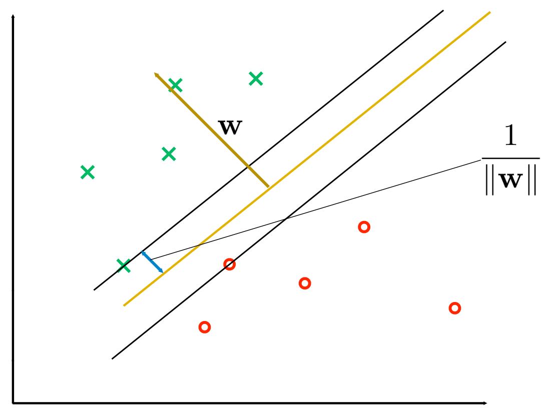
Maximum Margin

- The distance of a point $\mathbf{x}_n$ to the decision hyperplane is
$$\frac{|y(\mathbf{x}_n)|}{\|\mathbf{w}\|} = \frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|} = \frac{t_n(\mathbf{w}^T\phi(\mathbf{x}_n)+b)}{\|\mathbf{w}\|}$$
- This distance is independent of the scale of $\mathbf{w}$ and b
$$\frac{t_n(\alpha\mathbf{w}^T\phi(\mathbf{x}_n)+\alpha b)}{\|\alpha\mathbf{w}\|} = \frac{|t_n(\mathbf{w}^T\phi(\mathbf{x}_n)+b)|}{\|\mathbf{w}\|}$$
- Maximum margin is found by
$$\arg\max_{\mathbf{w},b}\left\{\frac{1}{\|\mathbf{w}\|}\min_n\{t_n(\mathbf{w}^T\phi(\mathbf{x}_n)+b)\}\right\}$$
- Rescaling: We can choose α so that
$$t_n(\alpha\mathbf{w}^T\phi(\mathbf{x}_n)+\alpha b) = 1$$

Rescaling



Rescaling



Maximum Margin

For all data points we have the constraint

$$t_n(\mathbf{w}^T\phi(\mathbf{x}_n)+b) \geq 1, \qquad n=1,\ldots,N$$

This means we have to maximize:

$$\arg\max_{\mathbf{w},b}\left\{\frac{1}{\|\mathbf{w}\|}\right\} \quad \text{s.th.} \quad t_n(\mathbf{w}^T\phi(\mathbf{x}_n)+b) \geq 1, \qquad n=1,\ldots,N$$

which is equivalent to

$$\arg\min_{\mathbf{w},b}\left\{\frac{1}{2}\|\mathbf{w}\|^2\right\} \quad \text{s.th.} \quad t_n(\mathbf{w}^T\phi(\mathbf{x}_n)+b) \geq 1, \qquad n=1,\ldots,N$$

Maximum Margin

$$\arg\min_{\mathbf{w},b}\left\{\frac{1}{2}\|\mathbf{w}\|^2\right\} \quad \text{s.th.} \quad t_n(\mathbf{w}^T\phi(\mathbf{x}_n)+b) \geq 1, \qquad n=1,\ldots,N$$

This is a constrained optimization problem.
It can be solved with a technique called quadratic programming.

Dual Formulation

For the constrained minimization we can introduce **Lagrange multipliers** a_n:

$$\min L(\mathbf{w},b,\mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n\left(t_n(\mathbf{w}^T\phi(\mathbf{x}_n)+b)-1\right)$$

Setting the derivatives of this wrt. $\mathbf{w}$ and b to 0 yields:

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n) \qquad 0 = \sum_{n=1}^{N} a_n t_n$$

If we plug these constraints back into $L(\mathbf{w},b,\mathbf{a})$:

$$\max \tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n,\mathbf{x}_m)$$

Dual Formulation

$$\max \tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n,\mathbf{x}_m)$$

subject to the constraints

$$a_n \geq 0, \qquad n=1,\ldots,N \qquad\qquad \sum_{n=1}^{N} a_n t_n = 0$$

This is called the **dual formulation** of the constrained optimization problem. The function k is again the **kernel function** and is defined as:

$$k(\mathbf{x}_n,\mathbf{x}_m) = \phi(\mathbf{x}_n^T)\phi(\mathbf{x}_m)$$

The simplest example of a kernel function is given for Φ= I. It is also known as the **linear kernel**.

$$k(\mathbf{x}_n,\mathbf{x}_m) = \mathbf{x}_n^T\mathbf{x}_m$$

The Kernel Trick in SVMs

- Other kernels are possible, e.g. the polynomial:

$$\phi(\mathbf{x}) = (x_1^2, x_2^2, x_1x_2, x_2x_1) \qquad \mathbf{x}\in\mathbb{R}^2$$
$$k(\mathbf{x}_n,\mathbf{x}_m) = \phi(\mathbf{x}_n^T)\phi(\mathbf{x}_m) = (\mathbf{x}^T\mathbf{x})^2$$

Kernel Trick for SVMs: If we find an optimal solution to the dual form of our constrained optimization problem, then we can replace the kernel by any other valid kernel and obtain again an optimal solution.

- Consequence: Using a non-linear feature transform Φ we obtain non-linear decision boundaries.

Observations and Remarks

- The kernel function is evaluated for each pair of training data points during training
- It can be shown that for every training data point it holds either $a_n = 0$ or $t_n y(\mathbf{x}_n) = 1$. In the latter case, they are support vectors.
- For classifying a new feature vector $\mathbf{x}$ we evaluate:

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x},\mathbf{x}_n) + b$$

We only need to compute that for the support vectors

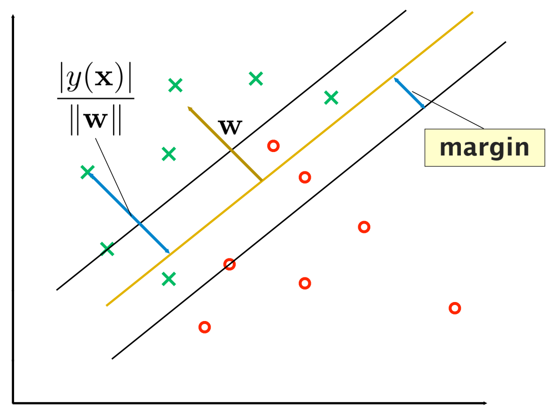
Multiple Classes

We can generalize the binary classification problem for the case of multiple classes.

This can be done with:

- one-to-many classification
- Defining a single objective function for all classes
- Organizing pairwise classifiers in a directed acyclic graph (DAGSVM)

Extension: Non-separable problems



Slack Variables

- The slack variable ξ_n is defined as follows:
- For all points on the correct side: $\xi_n = 0$
- For all other points: $\xi_n = |t_n - y(\mathbf{x}_n)|$
- This means that points with $0 < \xi_n \leq 1$ are correct classified, but inside the margin, points with $\xi_n > 1$ are misclassified.
- In the optimization, we modify the constraints:

$$t_n y(\mathbf{x}_n) \geq 1 - \xi_n, \qquad n = 1, \ldots, N$$

- and $\xi_n \geq 0$

Summary

- Kernel methods are used to solve problems by implicitly mapping the data into a (high-dimensional) feature space
- The feature function itself is not used, instead the algorithm is expressed in terms of the kernel
- Applications are manifold, including density estimation, regression, PCA and classification
- An important class of kernelized classification algorithms are Support Vector Machines
- They learn a linear discriminative function, which is called a hyper-plane
- Learning in SVMs can be done efficiently