Machine Learning for Computer Vision

Dr. Rudolph Triebel

Class Schedule

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

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

1. Introduction to Learning and Probabilistic Reasoning

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 Exam
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.

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

Using Probabilities!

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: $\mathcal{S} = \mathbb{R}^+$

**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:

\[
\sum_x p(X = x) = 1 \quad \text{Discrete case} \\
\int p(X = x) dx = 1 \quad \text{Continuous case}
\]

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:

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

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**:

\[
p(X = x) = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{(x-\mu)^2}{2\sigma^2}}
\]

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:**
- $p(X = x \leftrightarrow p(x)$
- $p(X = x \text{ and } Y = y) 
  \leftrightarrow p(x,y)$

**Definition 1.3:** The **conditional probability** of $X$ given $Y$'s 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 $X \text{ and } Y$ we have:

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

Furthermore, it holds:

- **“Sum Rule”** $p(x) = \sum_y p(x,y)$
- **“Product Rule”** $p(x,y) = p(y \mid x)p(x)$
Law of Total Probability

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

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

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:

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

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

“Normalizer”

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

Computing the Normalizer

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

Bayes rule

Total probability

Expectation and Covariance

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

$E[X] = \sum_x x p(x)$ (discrete case)

$E[X] = \int x p(x)dx$ (continuous case)

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


Causal vs. Diagnostic Reasoning

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

$$p(open \mid z) = \frac{p(z \mid open)p(open)}{p(z)} = \frac{p(z \mid open)p(open)}{p(z \mid open)p(open) + p(z \mid \neg open)p(\neg open)}$$
Example with Numbers
Assume we have this sensor model:
\[ p(z \mid \text{open}) = 0.6 \quad p(z \mid \neg\text{open}) = 0.3 \]

and:
\[ p(\text{open}) = p(\neg\text{open}) = 0.5 \quad \text{“Prior prob.”} \]

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”

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.

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 \( x \).

\[
p(x \mid z_1, \ldots, z_n) = \frac{p(z_n \mid x)p(x \mid z_1, \ldots, z_{n-1})}{p(z_n \mid z_1, \ldots, z_{n-1})} = \prod_{i=1}^{n} p(z_i \mid x)p(x)
\]

Recursion

Example: Sensing and Acting

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

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:

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

If the state space is continuous:

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

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

The Overall Bayes Filter

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

Bayes Filter Variants

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 formulation 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

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 \( p(y_k | x) \) for each class

Generative Model

• Est. the likelihoods \( p(x | y_k) \) and use Bayes rule for the post.

Categories of 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

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 \( (x_1, t_1), (x_2, t_2), \ldots \)
• Rule: Each new data point is assigned to the class of its nearest neighbor in feature space

1. Training instances in feature space

\[ \phi(x_1), \phi(x_1), \ldots \]

2. Map new data point into feature space

\[ \phi(x) \]

3. Compute the distances to the neighbors

4. Assign the label of the nearest training instance

\[ t := \arg \max_k p(y = k \mid x) \]

Generative Model: Example

Nearest-neighbor classification:

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

\[ K_k: \text{Number of points from class } k \text{ inside sphere} \]

\[ N_k: \text{Number of all points from class } k \]

Given: data points \( (x_1, t_1), (x_2, t_2), \ldots \)
• Rule: Each new data point is assigned to the class of its nearest neighbor in feature space

With this we can estimate:

\[ p(x \mid y = k) = \frac{K_k}{N_k V} \]

\[ p(x) = \frac{K}{N V} \]

and likewise:

\[ p(y = k \mid x) = \frac{p(x \mid y = k)p(y = k)}{p(x)} = \frac{K_k}{K} \]

Generative Model: Example

Nearest-neighbor classification:

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

• To classify the new data point \( x \) we compute the posterior for each class \( k = 1,2,\ldots \) and assign the label that maximizes the posterior.

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

- **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 $p(y_i|x)$ for each class

**Generative Model**

- Estimates the likelihoods $p(x|y_i)$ and uses Bayes rule for the posterior

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 $\phi$ 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**: $y(x, w) = w_0 + t_x^Tw$

Polynomial Regression

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

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

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 $\phi: \mathcal{X} \rightarrow \mathcal{Y}$ 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

Linear Regression

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

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

“Sum of Squared Errors”

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

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

$$ \frac{\partial}{\partial x_i} y(x_i, w) = w^T x_i $$

Using vector notation: $x_i := (1, x_i)^T$

$$ \nabla E(w) = \sum_{i=1}^{N} w^T x_i x_i^T - \sum_{i=1}^{N} t_i x_i^T = (0, 0) \Rightarrow w^T \left( \sum_{i=1}^{N} x_i x_i^T - \sum_{i=1}^{N} t_i x_i^T \right) = 0 $$

$$ y(x, w) = w_0 + \sum_{i=1}^{N} w_i x_i $$
Polynomial Regression

We define: \( \phi(x) := (1, \phi_1(x), \ldots, \phi_{M-1}(x))^T \)
And obtain:
\[
y(x, w) = w^T \phi(x) \\
E(w) = \frac{1}{2} \sum_{i=1}^{N} (w^T \phi(x_i) - t_i)^2 \\
\nabla E(w) = w^T \left( \sum_{i=1}^{N} \phi(x_i) \phi(x_i)^T \right) - \sum_{i=1}^{N} t_i \phi(x_i)^T \\
\]

Polynomial Regression

We define: \( \phi(x) := (1, \phi_1(x), \ldots, \phi_{M-1}(x))^T \)
And obtain:
\[
y(x, w) = w^T \phi(x) \\
E(w) = \frac{1}{2} \sum_{i=1}^{N} (w^T \phi(x_i) - t_i)^2 \\
\nabla E(w) = w^T \left( \sum_{i=1}^{N} \phi(x_i) \phi(x_i)^T \right) - \sum_{i=1}^{N} t_i \phi(x_i)^T \\
\]

Computing the Pseudoinverse

Mathematically, a pseudoinverse \( \Phi^+ \) exists for every matrix \( \Phi \).
However: If \( \Phi \) is (close to) singular the direct solution of \( \Phi \) 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

The Resulting Model Parameters

\( N = 10 \)  \( M = 10 \)
\( N = 20 \)  \( M = 10 \)
\( N = 40 \)  \( M = 10 \)
\( N = 100 \)  \( M = 10 \)

1. Machine Learning for Computer Vision
2. Dr. Rudolph Triebel
3. Computer Vision Group
Other Basis Functions

Other basis functions are possible:

- Gaussian basis function:
  
  \[ \phi_j(x) := \exp\left( -\frac{(x - \mu_j)^2}{2s^2} \right) \]
  
  where \( \mu_j \triangleq \text{mean val} \)
  
  \( s \triangleq \text{scale} \)

- Sigmoidal basis function:
  
  \[ \phi_j(x) := \sigma\left( \frac{x - \mu_j}{s} \right) \]
  
  where \( \sigma(a) = \frac{1}{1 + \exp(-a)} \)

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

Gaussian Basis Functions

- \( N = 10 \)
  - \( M = 1 \)
- \( N = 10 \)
  - \( M = 3 \)
- \( N = 10 \)
  - \( M = 5 \)
- \( N = 10 \)
  - \( M = 10 \)

Sigoidal 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:

\[ E(w) = \frac{1}{2} \sum_{i=1}^{N} (w^T \phi(x) - t_i)^2 + \frac{\lambda}{2} \|w\|^2 \]

where \( \lambda \) rules the influence of the regularization.

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:

\[ t = y(x, w) + \epsilon \quad \text{where} \quad \epsilon \sim \mathcal{N}(\cdot, 0, \sigma^2) \]

Thus, we have

\[ p(t \mid x, w, \sigma) = \mathcal{N}(t; y(x, w), \sigma^2) \]

...
### Maximum Likelihood Estimation

**Aim:** we want to find the \( w \) that maximizes \( p \).

\[ p(t \mid x, w, \sigma) \] is the **likelihood** of the measured data given a model. Intuitively:

Find parameters \( w \) that maximize the probability of measuring the already measured data \( t \).

**“Maximum Likelihood Estimation”**

We can think of this as fitting a model \( w \) to the data \( t \).

Note: \( \sigma \) is also part of the model and can be estimated. For now, we assume \( \sigma \) is known.

\[
egin{align*}
\ln p(t \mid x, w, \sigma) &= \sum_{i=1}^{N} \ln p(t_i \mid x, w, \sigma) \\
&= \frac{1}{2} \sum_{i=1}^{N} -\ln(\sigma^2) - (2\pi) - \frac{1}{\sigma^2} (w^T \phi(x_i) - t_i)^2 \\
&= -\frac{N}{2} (\ln(\sigma^2) + (2\pi)) - \frac{1}{2} \sum_{i=1}^{N} (w^T \phi(x_i) - t_i)^2 \\
\text{Constant for all } w &\text{ Is equal to } E(w)
\end{align*}
\]

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

### Maximum A-Posteriori Estimation

So far, we searched for parameters \( w \), that maximize the data likelihood. Now, we assume a Gaussian prior:

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

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

\[
\begin{align*}
\mathcal{N}(w; \hat{w}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(w - \hat{w})^2}
\end{align*}
\]

**“Maximum A-Posteriori Estimation (MAP)”**

\[
\ln p(w \mid x, t, \sigma_1, \sigma_2) \propto \ln p(t \mid x, w, \sigma_1) + \ln p(w \mid \sigma_2)
\]

\[
\begin{align*}
\text{const.} - \frac{1}{\sigma_1^2} \sum_{i=1}^{N} (w^T \phi(x_i) - t_i)^2 &\quad \text{const.} - \frac{1}{2\sigma_2^2} w^T w \\
\propto -\frac{1}{\sigma_1^2} \left( \sum_{i=1}^{N} (w^T \phi(x_i) - t_i)^2 + \frac{\sigma_2^2}{\sigma_1^2} w^T w \right)
\end{align*}
\]

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 probabilistic 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

Some Useful Formulas Before

If we are given this:

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

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

Then it follows (properties of Gaussians):

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

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

where

\[
\Sigma = (\Sigma_1^{-1} + A^T\Sigma_2^{-1}A)^{-1}
\]

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)

\[
= \eta p(z_t \mid x_t) p(x_t \mid u_1, z_1, \ldots, u_t)
\]

(Tot. prob.)

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

(Markov)

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

(Markov)

\[
= \eta \int p(x_t \mid x_{t-1}) p(x_{t-1} \mid u_1, z_1, \ldots, z_{t-1}) dx_{t-1}
\]

(A Simple Example)

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

1. Assumption of the model:
   - Basis functions are equal to identity \( \phi(x) = x \)
   - Prior mean is zero, prior covariance \( \sigma_2^2 = 0.5 \), noise variance is \( \sigma_1^2 = 0.2^2 \)
   - Ground truth is \( f(x, a) = a_0 + a_1 x \)
   - Noise variance is \( \sigma_0^2 = 0.3^2 \)

2. 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

Bayesian Linear Regression

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

\[
p(t \mid x, t, x)
\]

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.)

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

(Markov)

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

(Markov)

\[
= \eta \int p(x_t \mid x_{t-1}) p(x_{t-1} \mid u_1, z_1, \ldots, z_{t-1}) dx_{t-1}
\]

Note: Different Notation!
Bayesian Line Fitting

One data point observed

Prior

Data Space

"Hough Space"

Line examples drawn from the prior

Bayesian Line Fitting

Two data points observed

Prior

Data Space

"Hough Space"

Line examples drawn from the prior

The Predictive Distribution

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

\[ p(t \mid x, t, x) = \int p(t \mid x, w)p(w \mid x, t)dw \]

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.

\[ p(w \mid x, t) = \mathcal{N}(w \mid m_N, S_N) \]

where

\[ S_N^{-1} = S_N^{-1} + \sigma^{-2} \Phi^T \Phi \]

\[ w_N = S_N^{-1}(S_N^{-1} m_N + \sigma^{-2} \Phi^T t) \]

Predictive Distribution (3)

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

Predictive Distribution (4)

• Example: Sinusoidal data, 9 Gaussian basis functions, 4 data points
Predictive Distribution (5)

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

From: C.M. Bishop

Some samples from the posterior

Summary
- Regression can be expressed as a least-squares problem
- To avoid overfitting, we need to introduce a regularisation term with an additional parameter $\lambda$
- 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

The Bayes Filter (Rep.)

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

(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.)

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

(Markov)

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

(Markov)

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

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 Acyclic Graphs (DAG).

Definition

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

Simple Example

Given: 3 random variables $a$, $b$, and $c$

Joint prob: $p(a, b, c) = p(c \mid a, b)p(a, b) = p(c \mid a, b)p(b \mid a)p(a)$

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

Simple Example

In general: $K$ random variables $x_1, x_2, \ldots, x_K$

Joint prob:

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

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:

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

$\vdots$
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 \( x_5 \).

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

\[
p(x) = \prod_{k=1}^{K} p(x_k|p_{a_k})
\]

where \( p_{a_k} \) are ancestors of \( p_k \) and

\[ p(x) = p(x_1, \ldots, x_k) \]

Elements of Graphical Models

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

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

Elements of Graphical Models (2)

We distinguish between input variables and explicit hyper-parameters:

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

Elements of Graphical Models (3)

We distinguish between observed variables and hidden variables:

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

(deterministic parameters omitted)

Regression as a Graphical Model

Regression: Prediction of a new target value \( i \):

\[
p(i|\hat{x}, t, x, \alpha, \sigma^2) = \int p(i|\hat{x}, w) p(w|\alpha) p(\hat{x}|t, x, \sigma^2) dw
\]

Here: conditioning on all deterministic parameters

Using this, we can obtain the predictive distribution:

\[
p(i|\hat{x}, t, x, \alpha, \sigma^2) \propto \int p(i|\hat{x}, w) p(w|\alpha) p(\hat{x}|t, x, \sigma^2) dw
\]

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
    \[ p(x | \mu) = \prod_{k=1}^{K} \mu_k^{x_k} \sum_{j=1}^{K} \mu_j \]
  - All random variables are Gaussian
    \[ x_i \sim \mathcal{N}(\mu_i, \sigma_i^2) \]

Discrete Variables: Example

- Two dependent variables: \( K^2 - 1 \) parameters
  - \( x_1 \) and \( x_2 \)
  - \( K = 2 \)
  - \( x_1 | x_2 \)
  - \( x_2 | x_1 \)
  - \( x_3 \)
  - \( K - 1 + K(K - 1) = K^2 - 1 \)

- Independent joint distribution: \( 2(K-1) \) parameters
  - \( x_1 \) and \( x_3 \)
  - \( K - 1 + K - 1 = 2(K-1) \)
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:

![Graph Image]

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:

$$p(x_i \mid pa_i) = \mathcal{N}(x_i; \sum_{j \in pa_i} w_{ij}x_j + b_i, v_i)$$

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:** $x \perp y \mid \emptyset$

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 conditionally 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:

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

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

**Conditional Independence: Example 1**

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

$$p(a, b \mid c) = \frac{p(a, b, c)}{p(c)} = \frac{p(a)c p(b|c)}{p(c)}$$

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

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

**Conditional Independence: Example 2**

Again, we marginalize over $c$:

$$p(a, b) = p(a) \sum_{c} p(c|a)p(b|c) = p(a) \sum_{c} p(c|a)p(b, c, a)$$

$$= p(a) \sum_{c} p(c, a)p(b|c, a) = p(a)p(b|a)$$

And we obtain: $a \not\perp b \mid \emptyset$
Conditional Independence: Example 2

As before, now we condition on $c$:

\[
p(a, b | c) = \frac{p(a, b, c)}{p(c)} = \frac{p(a)p(c|a)p(b|c)}{p(c)} = p(a|c)p(b|c)
\]

And we obtain: $a \perp b | c$

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:

\[
\sum_c p(a, b, c) = p(a)p(b) \sum_c p(c | a, b) = p(a)p(b)
\]

And the result is: $a \perp b | \emptyset$

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 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-Separtation

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-head 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:** $\text{dsep}(A, B | C)$

D-Separtation: Example

- $\neg \text{dsep}(a, b | c)$
  - We condition on a descendant of $e$, i.e. it does not block the path from $a$ to $b$.

- $\text{dsep}(a, b | f)$
  - 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 $p$:

\[
\forall A, B, C : \text{dsep}(A, B | C) \Rightarrow A \perp B | C
\]

**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 $G$:

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

**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 $p$.

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

A perfect map uniquely defines a probability distribution.

---

**The Markov Blanket**

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

$$p(x_i | x_{\neq i}) = \frac{p(x_1, \ldots, x_M)}{\prod_{k} p(x_k | p_{x_k})} \cdot \prod_{i} p(x_i | x_{M_i})$$

**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.

---

**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.

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

**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**

**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 $p(\neg c | \neg b) = 0.81$ and obtain $p(\neg b | \neg c) \approx 0.257$ similarly: $p(\neg b | \neg c, \neg a) \approx 0.111$

"a explains c away"
### 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 the “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**

### 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:

$$p(x_i, x_j | x_{\{i,j\}}) = p(x_i | x_{\{i,j\}})p(x_j | x_{\{i,j\}})$$

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.

### Converting Directed to Undirected Graphs (1)

In this case: $Z=1$

\[-dsep(a, b|c)\]
\[dsep(a, b|f)\]

### Converting Directed to Undirected Graphs (2)

**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)

\[ p(x) = p(x_1)p(x_2)p(x_3) | x_1, x_2, x_3 \]

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.

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

\[ p(x) = \frac{1}{Z} \prod_{i=1}^{N} \psi_i(x_1, x_2, x_3) \]

The marginal at \( x_3 \) is

\[ p(x_3) = \sum_{x_1} \sum_{x_2} \sum_{x_4} p(x) \]

In the general case with \( N \) nodes we have

\[ p(x) = \frac{1}{Z} \prod_{i=1}^{N} \psi_i(x_1, x_2, x_3) \]

and

\[ p(x_n) = \sum_{x_1} \cdots \sum_{x_{n-1}} \sum_{x_{n+1}} \cdots \sum_{x_N} p(x) \]

Inference on a Chain

The messages \( \mu_\alpha \) and \( \mu_\beta \) can be computed recursively:

\[ \mu_\alpha(x_n) = \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \mu_\alpha(x_{n-1}) \]

\[ \mu_\beta(x_n) = \sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \mu_\beta(x_{n+1}) \]

Computation of \( \mu_\alpha \) starts at the first node and computation of \( \mu_\beta \) starts at the last node.
Inference on a Chain

\[\mu_{\alpha}(x_2) = \sum_{\alpha_1, \alpha_2} \phi_{\alpha\alpha_1}(x_1, x_2) \quad \mu_{\beta}(x_{N-1}) = \sum_{x_N} \phi_{N-1,N}(x_{N-1}, x_N)\]

• The first values of \(\mu_{\alpha}\) and \(\mu_{\beta}\) are:

- \(\mu_{\alpha}(x_2) = \sum_{x_1} \phi_{\alpha\alpha_1}(x_1, x_2)\)
- \(\mu_{\beta}(x_{N-1}) = \sum_{x_N} \phi_{N-1,N}(x_{N-1}, x_N)\)

• 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 \(p(x_n)\)

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

Inference on a Chain (Rep.)

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

Undirected

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

for all variables required.

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

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

Computer Vision Group

Prof. Daniel Cremers

4a. Inference in Graphical Models

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

More General Graphs

Conversion from a directed to an undirected tree is no problem, because no links are inserted
The message-passing algorithm can be extended to more general graphs:

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

Thus, we have

Key insight: Sum and product can be exchanged!

"Messages from factors to node x"
The Sum-Product Algorithm

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

$$G_M(x_m, X_m) = \prod_{l \in \text{ne}(x_m)} F_l(x_m, X_m)$$

This results in the exact same situation as above!

We can now recursively apply the derived rules:

$$\mu_{x \rightarrow f}(x_m) = \prod_{l \in \text{ne}(x_m)} \sum_{x_m \in X_m} F_l(x_m, x_m)$$

$$= \prod_{l \in \text{ne}(x_m)} \mu_{f \rightarrow x_m}(x_m)$$

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 max instead of sum as above and maximize

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

Solution to the first:

Transform everything into log-space and use sums

Message passing can be used as above!

The Max-Sum Algorithm

To find the maximum value of $p(x)$, 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 $x$ can maximize $p(x)$, leading to wrong assignments of the overall maximum

Solution to the first:

Transform everything into log-space and use sums

The Max-Sum Algorithm

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

$$\max(ab, ac) = a \max(b, c)$$

if $a \geq 0$

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

Chain example:

$$\max_x p(x) = \frac{1}{Z} \max_{x_1} \ldots \max_{x_N} \left[ \psi(x_1, x_2) \ldots \psi_{N-1,N}(x_{N-1}, x_N) \right]$$

$$= \frac{1}{Z} \max_{x_1} \left[ \psi(x_1, x_2) \ldots \max \psi_{N-1,N}(x_{N-1}, x_N) \right]$$

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.

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 $y$ and features as continuous random variables $x$.
- A CRF represents the conditional probability
  \[ p(y | x, w) = \frac{\prod_C \phi_C(x_C, y_C; w)}{\sum_y \prod_C \phi_C(x_C, y_C; w)} \]
  where $w$ are parameters learned from training data.
- CRFs are **discriminative** and MRFs are **generative**.

Derivation of the formula for CRFs:

In the training phase, we compute parameters $w$ that maximize the posterior:

\[ w^* = \arg \max_w p(w | x^*, y^*) \times p(x^* | x^*, w)p(w) \]

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

\[ \arg \max_y p(y | x, w^*) \]

Potential Functions

- The only requirement for the potential functions is that they are positive. We achieve that with:
  \[ \phi_C(x_C, y_C, w) := \exp(f(x_C, y_C)) \]
  where $f$ is a compatibility function that is large if the labels $y_C$ fit well to the features $x_C$.
- This is called the **log-linear model**.
- The function $f$ can be, e.g. a local classifier.

CRF Training

We minimize the negative log-posterior:

\[ w^* = \arg \min_w \{- \log p(w | x^*, y^*) = \arg \min_w \{- \log p(y^* | x^*, w) - \log p(w)\} \]

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

\[ p(y^* | x^*, w) \approx \prod_C p(y^*_C | M(y^*_C), x^*, w) \]

where

\[ p(y^*_C | M(y^*_C), x^*, w) = \frac{\prod_C \phi_C(x^*_C, y^*_C; y^*_C; w)}{\sum_{y_C} \prod_C \phi_C(x^*_C, y^*_C; y^*_C; w)} \]

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

Pseudo Likelihood

Pseudo Likelihood

CRF Training and Inference

Training:
- Using pseudo-likelihood, training is efficient. We have to minimize:
  \[ L(w) = -\log p(y^* | x^*, w) + \frac{1}{2\sigma^2} w^T w \]

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

Inference:
- Only approximately, 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:

![Dynamic Bayes Network Diagram]

- This incorporates the following Markov assumptions:
  \[ p(z_t | x_{0:t}, u_{1:t}, z_{1:t}) = p(z_t | x_t) \] (measurement)
  \[ p(x_t | x_{0:t-1}, u_{1:t}, z_{1:t}) = p(x_t | x_{t-1}, u_t) \] (state)

Example

“Occasionally dishonest casino”:
- observations: faces of a die \( z_t \in \{1, 2, \ldots, 6\} \)
- hidden states: two different dice, one fair, one loaded

![Dice Diagram]

Formulation as HMM

1. Discrete random variables
   - Observation variables: \( \{z_i\}, n = 1..N \)
   - State variables (unobservable): \( \{x_i\}, n = 1..N \)
   - Number of states: \( K: x_i \in \{1..K\} \)

2. Transition model \( p(x_i | x_{i-1}) \)
   - 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 \( \pi_1, \pi_2, \pi_3 \)

3. Observation model \( p(z_i | x_i) \) with parameters \( \phi \)
   - Observation only depends on the current state
   - Example: output of a “local” place classifier

Application Example (1)

- Given an observation sequence \( z_1, z_2, z_3, \ldots \)
- Assume that the model parameters \( \theta = (A, \pi, \phi) \) are known
- What is the probability that the given observation sequence is actually observed under this model, i.e., \( p(Z; \theta) \)?
- 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 \( \theta = (A, \pi, \phi) \) 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 \( \theta = (A, \pi, \phi) \)?
- 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)

The Forward Algorithm

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

\[
p(x_t = j \mid z_{1:t-1}) = \sum_i p(x_t = j \mid x_{t-1} = i)p(x_{t-1} = i \mid z_{1:t-1})
\]

- Then, we do the update using Bayes rule:

\[
\alpha_t(j) := p(x_t = j \mid z_{1:t}) = p(x_t = j \mid z_t, z_{1:t-1}) = \frac{1}{Z_t} p(z_t \mid x_t = j, z_{1:t-1})p(x_t = j \mid z_{1:t-1})
\]

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

The Forward-Backward Algorithm

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

The Viterbi Algorithm

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

- This can be recursively computed (backwards):

\[
\beta_t^{-1}(i) = p(z_{t:T} \mid x_{t-1} = i) = \sum_j p(x_t = j, z_t, z_{t+1:T} \mid x_{t-1} = i)
\]

\[
= \sum_j p(z_{t+1:T} \mid x_t = j) p(z_t \mid x_t = j, z_{t-1} = i) p(x_t = j \mid x_{t-1} = i) = \sum_j \beta_t(j) p(z_t \mid x_t = j) p(x_t = j \mid x_{t-1} = i)
\]

- This is exactly the same as the message-passing algorithm (sum-product):

  - forward messages \( \alpha_t \) (vector of length \( K \))
  - backward messages \( \beta_t \) (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 \( p(X, Z | \theta) \)

- Define \( \delta_t(j) := \max_{x_1, \ldots, x_{t-1}} p(x_1 \ldots x_{t-1}, x_t = j | Z) \)

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

\[
\begin{align*}
\delta_0(j) &= \psi(j) \\
\delta_t(j) &= \max_{x_1, \ldots, x_{t-1}} p(x_1 \ldots x_{t-1}, x_t = j | Z) \\
&= \max_{x_1, \ldots, x_{t-1}} [\alpha_{x_1} \beta_{x_1} (j)]
\end{align*}
\]

3. Learning the Model Parameters

- Objective: Find the model parameters knowing the observations: \( \pi, A, \phi \)

- 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:
    \[
    \arg \max_{\pi, A, \phi} \log \left( \sum_{Z} P(Z) \right)
    \]

The Viterbi algorithm

- Initialize:
  - \( \delta_0 = p(x_0) \cdot p(z_0) \)
  - Compute recursively for \( n = 1 \ldots N \):
    \[
    \delta_n(j) = p(x_n, z_n) \max_i [\delta_{n-1}(i) p(z_n | x_n)]
    \]

- On termination:
  - \( p(Z, X | \theta) = \max \delta_n(n) \)
  - \( x_n^* = \arg \max_i \delta_n(i) \)
  - Backtracking:
    \[
    x_{n-1}^* = a(x_n^*)
    \]

The Baum-Welsh algorithm

- E-Step (assuming we know \( \pi, A, \phi \), i.e. \( \theta_{old} \))

- Define the posterior probability of being in state \( i \) at step \( k \):
  - \( \gamma(x_n) = p(x_n | Z) \)
  - It follows that \( \gamma(x_n) = \alpha(x_n) \beta(x_n) / p(Z) \)

- Define \( \xi(x_{n-1}, x_n) = p(x_{n-1} \mid x_n, Z) \)

- We need to compute:
  \[
  Q(\theta, \theta_{old}) = \sum_{Z} \log p(Z) \theta_{old} \log p(Z, X | \theta)
  \]
The Baum-Welsh algorithm

- Maximizing Q also maximizes the likelihood: 
  \[ p(Z|\theta) \geq p(Z|\theta^{\text{old}}) \]
- M-Step:
  \[
  \pi_k = \frac{\sum_{j=1}^{N} \gamma_j(x) x_{1k}}{\sum_{j=1}^{N} \sum_{x} \gamma_j(x) x_{1j}}
  \]
  here, we need forward and backward step!
- \[
  A_{jk} = \frac{\sum_{t=2}^{T} \xi(x_{t-1,j},x_{tk})}{\sum_{j=1}^{N} \sum_{t=2}^{T} \xi(x_{t-1,j},x_{tk})}
  \]
  With these new values, Q is recomputed
- This is done until the likelihood does not increase anymore (convergence)

The Scaling problem

- Probability of sequences 
  \[
  \prod_{i} p(x_i | ...) \ll 1
  \]
- 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.: 
  \[
  \hat{\alpha}(x) = \alpha(x) / p(z_1,z_2,...,z_n)
  \]

5. Boosting

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

Repetition: Regression

We start with a set of basis functions
\[
\phi(x) = (\phi_0(x), \phi_1(x), \ldots, \phi_{M-1}(x)) \quad x \in \mathbb{R}^d
\]
The goal is to fit a model into the data 
\[
y(x, w) = w^T \phi(x)
\]
To do this, we need to find an error function, e.g.:
\[
E(w) = \frac{1}{2} \sum_{i=1}^{N} (w^T \phi(x_i) - t_i)^2
\]
To find the optimal parameters, we derived \( E \) with respect to \( w \) 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 AdaBoost Algorithm

1. For \( i = 1, \ldots, N \) \( v_i \leftarrow \frac{1}{N} \)
2. For \( m = 1, \ldots, M \)
   - Fit a classifier (“basis function”) \( \phi_m \) that minimizes
     \[
     \sum_{i=1}^{N} v_i I(t_i \neq \phi_m(x_i))
     \]
   - Compute \( \text{err}_m = \frac{\sum_{i=1}^{N} v_i I(t_i \neq \phi_m(x_i))}{\sum_{i=1}^{N} v_i} \) and \( \alpha_m = \log \frac{1 - \text{err}_m}{\text{err}_m} \)
   - Update the weights: \( v_i \leftarrow v_i \exp(\alpha_m I(t_i \neq \phi_m(x_i))) \)
3. Use the resulting classifier:
   \[ y(x) = \text{sgn} \sum_{m=1}^{M} \alpha_m \phi_m(x) \]

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

Minimizing the Exponential Loss

Aim: find \( w_m \) and \( \phi_m \) so that
\[
(w_m, \phi_m) = \arg \min_{w, \phi} \sum_{i=1}^{N} L(t_i, y_m-1(x_i) + w\phi(x_i))
\]
where \( L(t, y) = \exp(-ty) \)

Solution:
\[
\phi_m = \arg \min_{\phi} \sum_{i=1}^{N} v_{i,m} I(t_i \neq \phi(x_i))
\]
\[
w_m = \frac{1}{2} \log \frac{1 - \text{err}_m}{\text{err}_m}
\]

Sequential Fitting of Basis Functions

Idea: We start with a basis function \( \phi_0(x) \):
\[
y_0(x, w_0) = w_0\phi_0(x) \quad w_0 = 1
\]
Then, at iteration \( m \), we add a new basis function \( \phi_m(x) \) to the model:
\[
y_m(x, w_0, \ldots, w_m) = y_{m-1}(x, w_0, \ldots, w_{m-1}) + w_m\phi_m(x)
\]

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 \( \phi_m \) so that
\[
(w_m, \phi_m) = \arg \min_{w, \phi} \sum_{i=1}^{N} L(t_i, y_m-1(x_i) + w\phi(x_i))
\]
where \( L(t, y) = \exp(-ty) \)

Solution:
\[
\phi_m = \arg \min_{\phi} \sum_{i=1}^{N} v_{i,m} I(t_i \neq \phi(x_i))
\]
\[
w_m = \frac{1}{2} \log \frac{1 - \text{err}_m}{\text{err}_m}
\]

Minimizing the Exponential Loss

Aim: find \( w_m \) and \( \phi_m \) so that
\[
(w_m, \phi_m) = \arg \min_{w, \phi} \sum_{i=1}^{N} L(t_i, y_m-1(x_i) + w\phi(x_i))
\]
where \( L(t, y) = \exp(-ty) \)

Solution:
\[
\phi_m = \arg \min_{\phi} \sum_{i=1}^{N} v_{i,m} I(t_i \neq \phi(x_i))
\]
\[
w_m = \frac{1}{2} \log \frac{1 - \text{err}_m}{\text{err}_m}
\]
\[
v_{i,m+1} = v_{i,m} \exp(2w_m I(t_i \neq \phi_m(x_i)))
\]
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 classification 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

- **m = 2**
- **m = 3**
- **m = 6**
- **m = 10**
- **m = 150**

### Decision Trees

- A more general version of decision stumps are decision **trees**:
  - At every node, a decision is made.
  - Can 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. \( p(y = 1 | x) = 1/2 \) if we have a red ellipse

**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?

**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

**Using the Log-Loss**

- The log-loss is defined as:
  \[ L(t, y(x)) = \log_2(1 + \exp(-2ty(x))) \]
- It penalizes misclassifications only linearly

**Growing the Log-Loss**

- The log-loss is defined as:
  \[ L(t, y(x)) = \log_2(1 + \exp(-2ty(x))) \]
- It penalizes misclassifications only linearly

**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

**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!

**The LogitBoost Algorithm**

1. For \( i = 1, \ldots, N \) : \( v_i \leftarrow 1/N \) \( \pi_i \leftarrow 1/2 \)
2. For \( m = 1, \ldots, M \)
   - Compute the working response \( z_i = \frac{t_i - \pi_i}{\pi_i(1 - \pi_i)} \)
   - Compute the weights \( v_i = \pi_i(1 - \pi_i) \)
   - Find \( \phi_m \) that minimizes \( \sum_{i=1}^{N} v_i(z_i - \phi(x_i))^2 \)
   - Update \( y(x) \leftarrow y(x) + \frac{1}{2} \phi_m(x) \) and \( \pi_i \leftarrow \frac{1}{1 + \exp(-2y(x_i))} \)
3. Use the resulting classifier:
   \[ y(x) = \text{sgn} \sum_{m=1}^{M} \phi_m(x) \]
**Weighted Least-Squares Regression**

- Instead of a weak classifier, LogitBoost uses “weighted least-squares regression”
- This is very similar to standard least-squares regression:
  \[ E(w) = \frac{1}{2} \sum_{i=1}^{N} v_i (w^T \phi(x_i) - t_i)^2 \]
- This results in a matrix \( \hat{\phi} = V^{1/2} \phi \) where \( V^{1/2} = \text{diag}(\sqrt{v_1}, \ldots, \sqrt{v_N}) \)
- The solution is \( w = (\hat{\phi}^T \hat{\phi})^{-1} \hat{\phi}^T t \)

**Generalization: Gradient Boost**

- Initialize \( f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(t_i, \phi(x_i, \gamma)) \)
- for \( m = 1, \ldots, M \)
  - Compute the gradient residual \( r_{im} = -\frac{\partial L(t_i, f_m(x_i))}{\partial f(x_i)} f_m(x_i) = f_{m-1}(x_i) \)
  - Use the weak learner to compute that minimizes \( \sum_{i=1}^{N} (r_{im} - \phi(x_i; \gamma_m))^2 \)
  - Update \( f_m(x) = f_{m-1}(x) + \nu \phi(x, \gamma) \)
  - Return \( f(x) = f_M(x) \)

**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.
  \( \int_{x_{min}}^{x_{max}} \int_{y_{min}}^{y_{max}} I(x, y) \text{d}x \text{d}y - \int_{x_{min}}^{x_{max}} \int_{y_{min}}^{y_{max}} I(x, y) \text{d}x \text{d}y \)
  - One feature defined as:
    - Feature type: A,B,C or D
    - Feature position and size

**Weak Classifiers Used**

- A weak classifier has 3 attributes:
  - A feature \( f_j \) (type, size and position)
  - A threshold \( \theta_j \)
  - A comparison operator op\( \_j = '<' \) or '>'
- The resulting weak classifier is:
  \( h_j(x) = f_j(x) \text{op}_j \theta_j \)
- \( x \) is a 24x24 pixels window in the image

**Gentle Boost**

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

**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!

**The integral image**

- Defined as:
  \[ I_{\text{int}}(X,Y) = \int_{x=X} \int_{y=Y} I(x,y) \text{d}x \text{d}y \]
- Integral on rectangle D can be computed in 4 access to \( I_{\text{int}} \):
  \[ \int_{D} I(x,y) = I_{\text{nc}}(4) + I_{\text{nc}}(1) - I_{\text{nc}}(2) - I_{\text{nc}}(3) \]
- Very efficient way to compute features

**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

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

6. Kernel Methods

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

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

if we write this in vector form, we get

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

and the solution is

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

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

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

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

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

“Dual Variables”

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

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

This is called the dual formulation.

Note: $a \in \mathbb{R}^N \quad w \in \mathbb{R}^D$
**Dual Representation**

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

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

\[
J(a) = \frac{1}{2} a^T K a - a^T K t + \frac{1}{2} t^T t + \frac{\lambda}{2} a^T K a
\]

\[
a = (K + \lambda I_N)^{-1} t
\]

This we can use to make predictions:

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

(now \(x\) is unknown and \(a\) is given from training)

**Constructing Kernels**

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

\[
k(x_1, x_2) = \phi(x_1)^T \phi(x_2)
\]

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

1. **Symmetry**: \(k(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle = \langle \phi(x_2), \phi(x_1) \rangle\)
2. **Linearity**: \(\langle a \phi(x_1) + z, \phi(x_2) \rangle = a \langle \phi(x_1), \phi(x_2) \rangle + \langle z, \phi(x_2) \rangle\)
3. **Positive definite**: \(\langle \phi(x), \phi(x) \rangle \geq 0\), equal if \(\phi(x) = 0\)

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

**Examples of Valid Kernels**

- **Polynomial Kernel**:
  \[
k(x_1, x_2) = (x_1^T x_2 + c)^d\quad c > 0\quad d \in \mathbb{N}
  \]

- **Gaussian Kernel**:
  \[
k(x_1, x_2) = \exp(-||x_1 - x_2||^2/2\sigma^2)
  \]

- **Kernel for sets**:
  \[
k(A_1, A_2) = 2|A_1 \cap A_2|
  \]

- **Matern kernel**:
  \[
k(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left( \frac{\sqrt{2\nu} r}{l} \right)^\nu K_\nu \left( \frac{\sqrt{2\nu} r}{l} \right)\quad r = ||x_1 - x_2||, \nu > 0, l > 0
  \]

**A Simple Example**

Define a kernel function as

\[
k(x, x') = (x^T x')^2\quad x, x' \in \mathbb{R}^2
\]

This can be written as:

\[
(x_1 x_1' + x_2 x_2')^2 = x_1^2 x_1'^2 + 2x_1 x_1' x_2 x_2' + x_2^2 x_2'^2\]

\[
= (x_1^2, x_2^2, \sqrt{2}x_1 x_2, x_2^2, \sqrt{2}x_1 x_2')^T
\]

\[
= \phi(x)^T \phi(x')
\]

It can be shown that this holds in general for

\[
k(x, x_j) = (x_j^T x_j)^d
\]

**Idea**: Don’t use features at all but simply define a similarity function expressed as the kernel!
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
  \[ \|x_i, x_j\|^2 = x_i^T x_i + x_j^T x_j + 2x_i^T x_j \]
- We can now replace the dot products by a valid Mercer kernel and we obtain:
  \[ d(x_i, x_j)^2 = k(x_i, x_j) + k(x_j, x_j) + 2k(x_i, x_j) \]
- This is a kernelized nearest-neighbor classifier
- We do not explicitly compute feature vectors!

Example: Principal Component Analysis

- Given: data set \( \{x_n\}_{n=1}^{N} \) \( x_n \in \mathbb{R}^D \)
- 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 \( u_1 \), i.e.: \( u_1^T u_1 = 1 \)
- Each data point is projected onto the subspace using the dot product: \( u_1^T x_n \)

Principal Component Analysis

Visualization:

- Mean:
  \[ \mu = \frac{1}{N} \sum_{n=1}^{N} u_1^T x_n = \frac{1}{N} u_1^T \sum_{n=1}^{N} x_n = u_1^T \bar{x} \]
- Variance:
  \[ \sigma^2 = \frac{1}{N} \sum_{n=1}^{N} (u_1^T x_n - u_1^T \bar{x})^2 = \frac{1}{N} \sum_{n=1}^{N} (u_1^T (x_n - \bar{x}))^2 = u_1^T \sum_{n=1}^{N} (x_n - \bar{x})(x_n - \bar{x})^T u_1 \]

Using a Lagrange multiplier:

\[ u^* = \arg \max_{u_1} u_1^T S u_1 + \lambda_1 (1 - u_1^T u_1) \]

Setting the derivative wrt. \( u_1 \) to 0 we obtain:

\[ S u_1 = \lambda_1 u_1 \]

Thus: \( u_1 \) must be an eigenvector of \( S \).

Multiplying with \( u_1^T \) from left gives: \( u_1^T S u_1 = \lambda_1 \)
Thus: \( \sigma^2 \) is largest if \( u_1 \) 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 \( u_1 \).
If we do this \( M \) times we obtain:

\[ u_1, \ldots, u_M \] are the eigenvectors of the \( M \) largest eigenvalues of \( S: \lambda_1, \ldots, \lambda_M \)

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

\[
x^T = \begin{pmatrix} u_1^T \\ \vdots \\ u_M^T \end{pmatrix} (x - \bar{x})
\]

Reconstruction using PCA

Plugging in, we have:

\[
\tilde{x}_n = \sum_{i=1}^{M} (x_n^T u_i) u_i + \sum_{i=M+1}^{D} (x_n^T u_i) u_i
\]

\[
= \sum_{i=1}^{M} (x_n^T u_i) u_i - \sum_{i=M+1}^{D} (x_n^T u_i) u_i + \sum_{i=1}^{M} (x_n^T u_i) u_i
\]

\[
x + \sum_{i=1}^{M} (x_n - \bar{x})^T u_i u_i
\]

4. Add mean

1. Subtract mean
2. Project onto first \( M \) eigenvectors

Application of PCA: Face Recognition

Approach:

- Convert the image into a 1D 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.

Kernel PCA

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

\[ \sum_{i=1}^{N} x_i = 0 \]

Then, in standard PCA we have the eigenvalue problem:

\[ S u_i = \lambda_i u_i \]

\[ S = \frac{1}{N} \sum_{i=1}^{N} x_i x_i^T \]

Now, we use a non-linear transformation \( \phi(x_n) \)
and we assume \( \sum_{i=1}^{N} \phi(x_n) = 0 \). We define \( C \) as:

\[ C = \frac{1}{N} \sum_{i=1}^{N} \phi(x_n) \phi(x_n)^T \]

Goal: find eigenvalues without using features!
Kernel PCA
Plugging in:
\[
\frac{1}{N} \sum_{n=1}^{N} \phi(x_n)\phi(x_n)^T v_i = \lambda_i v_i
\]
This means, there are values \( \alpha_{in} \) so that
\[
v_i = \sum_{n=1}^{N} \alpha_{in} \phi(x_n).
\]
With this we have:
\[
\frac{1}{N} \sum_{n=1}^{N} \phi(x_n)\phi(x_n)^T \sum_{m=1}^{N} \alpha_{im} \phi(x_m) = \lambda_i \sum_{m=1}^{N} \alpha_{in} \phi(x_n)
\]
Multiplying both sides by \( \phi(x_l) \) gives:
\[
\frac{1}{N} \sum_{n=1}^{N} k(x_l, x_n) \sum_{m=1}^{N} \alpha_{im} k(x_m, x_n) = \lambda_i \sum_{m=1}^{N} \alpha_{in} k(x_l, x_n)
\]
where \( k(x_l, x_n) = \phi(x_l)^T \phi(x_n) \). This is our expression in terms of the kernel function!

Rescaling: We can choose \( \alpha \) so that
\[
41
\]

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”):
\[
y(x, w) = w^T \phi(x) - b
\]

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
• The distance of a point \( x_n \) to the decision hyperplane is
\[
|y(x_n)| = t_n w^T \phi(x_n) + b
\]
This distance is independent of the scale of \( w \) and \( b \)
\[
t_n (\alpha w^T \phi(x_n) + ab) = t_n (w^T \phi(x_n) + b)
\]
Maximum margin is found by
\[
\arg \max_{w,b} \left\{ \frac{1}{|w|} \min \{ t_n (w^T \phi(x_n) + b) \} \right\}
\]
Rescaling: We can choose \( \alpha \) so that
\[
t_n (\alpha w^T \phi(x_n) + ab) = 1
\]
Rescaling

Maximum Margin

For all data points we have the constraint
\[ t_n(w^T \phi(x_n) + b) \geq 1, \quad n = 1, \ldots, N \]
This means we have to maximize:
\[ \arg\max_{w,b} \left\{ \frac{1}{2}||w||^2 \right\} \quad \text{s.th.} \quad t_n(w^T \phi(x_n) + b) \geq 1, \quad n = 1, \ldots, N \]
which is equivalent to
\[ \arg\min_{w,b} \left\{ \frac{1}{2}||w||^2 \right\} \quad \text{s.th.} \quad t_n(w^T \phi(x_n) + b) \geq 1, \quad n = 1, \ldots, N \]

Dual Formulation

For the constrained minimization we can introduce Lagrange multipliers \(a_n\):
\[ \min L(w, b, a) = \frac{1}{2}||w||^2 - \sum_{n=1}^{N} a_n (t_n(w^T \phi(x_n) + b) - 1) \]
Setting the derivatives of this wrt. \(w\) and \(b\) to 0 yields:
\[
\begin{align*}
    w &= \sum_{n=1}^{N} a_n t_n \phi(x_n) \\
    0 &= \sum_{n=1}^{N} a_n t_n
\end{align*}
\]
If we plug these constraints back into \(L(w, b, a)\) :
\[ \max \tilde{L}(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m) \]

The Kernel Trick in SVMs

- Other kernels are possible, e.g. the polynomial:
\[ \phi(x) = (x_1^2, x_2^2, x_1, x_2, x_2 x_1) \quad x \in \mathbb{R}^2 \]
\[ k(x_n, x_m) = \phi(x_n^T) \phi(x_m) = (x^T x)^2 \]
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 \(\Phi\) we obtain non-linear decision boundaries.

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)

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 \(t_n y(x_n) = 1\). In the latter case, they are support vectors.
- For classifying a new feature vector \(x\) we evaluate:
\[ y(x) = \sum_{n=1}^{N} a_n t_n k(x, x_n) + b \]
We only need to compute that for the support vectors.
Extension: Non-separable problems

- The slack variable $\xi_n$ is defined as follows:
  - For all points on the correct side: $\xi_n = 0$
  - For all other points: $\xi_n = |t_n - y(x_n)|$
- 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:
  $$t_ny(x_n) \geq 1 - \xi_n, \quad n = 1, \ldots, N$$
  and $\xi_n \geq 0$

Slack Variables

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