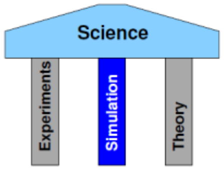
Introduction

What is Scientific Computing?



- Algorithms and modeling and simulation
- Computer and information science
- The computing infrastructure

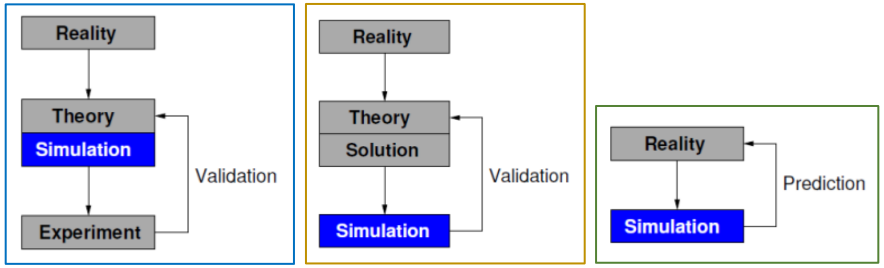
Gaining Scientific Knowledge

The classical scientific process

- Characterization of the real world
  - Observation
  - Quantification/measurement
- Hypothesis
  - Theory
  - model
- Prediction
  - Consequences/logical deducation from hypothesis/model
- Experiment
  - Verification/falsification
  - Discrepancies might lead to improved model

When is a Simulation required?

- Replacing analytical solvers
- Replacing Experiments
- Replacing analytical solvers and experiments



High Performance Computing (HPC)

Parallel processing for running advanced application programs efficiently, reliably and quickly.

Introduction - Population Models

Population Models and ODE

Population models describe the interaction between $k \in \mathbb{N}$ different species $y_1, \dots, y_k$ in an ecological or social system.

They are often described as an initial value problem based on a set of Ordinary Differential Equations (ODE) of first order.

$$\frac{d}{dt} y = f\big(t, y(t)\big)$$

Where

$$y(t) = \begin{pmatrix} y_1(t) \\ \vdots \\ y_n(t) \end{pmatrix}, \qquad f\big(t, y(t)\big) = \begin{pmatrix} f_1\big(t, y(t)\big) \\ \vdots \\ f_n\big(t, y(t)\big) \end{pmatrix}, \qquad y(t = t_0) = y^{(0)} = \begin{pmatrix} y_1(t_0) \\ \vdots \\ y_n(t_0) \end{pmatrix}$$

Such ODEs' can be solved numerically, e.g. using the *Runge-Kutta* method.

Such models often depend on plausability considerations rather than natural laws. Therefore, it is important to compare the outcome of such numerical simulations with real data.

Population Model

Let us consider the species $y(t)$ as a function of time t without any interaction with its environment:

- $y(t)$: head count at time t
- $b > 0$: birth rate
- $m > 0$: mortality rate
- $b - m$: growth rate

We can describe the development of $y(t)$ through an ODE of first order:

$$\frac{dy}{dt} = b \cdot y - m \cdot y = (b - m) \cdot y$$

Preditor-Prey Model

In its original form, it describes a theory of competition between two species. Applied to an interaction between a predator and its prey, we can reduce the model to the following assumptions:

Two populations $y_1(t)$ = prey and $y_2(t)$ = predator

- $y_1(t)$ increases with the specific net rate g_1
- $y_2(t)$ dies with the specific net rate g_2

The prey is eaten by the predator, which results in an increase of predators and a corresponding decrease of prey by $g_3 \cdot y_1(t) \cdot y_2(t)$.

Mathematically, the model is described by

- $\frac{dy_1}{dt} = g_1 y_1 - g_3 y_1 y_2$
- $\frac{dy_2}{dt} = g_2 y_2 + g_3 y_1 y_2$

Or as a two dimensional vectors

$$\begin{pmatrix} \frac{dy_1}{dt} \\ \frac{dy_2}{dt} \end{pmatrix} = \begin{pmatrix} g_1 y_1 - g_3 y_1 y_2 \\ g_2 y_2 + g_3 y_1 y_2 \end{pmatrix}$$

Of special interest is the so-called fixed point, where both derivatives vanish:

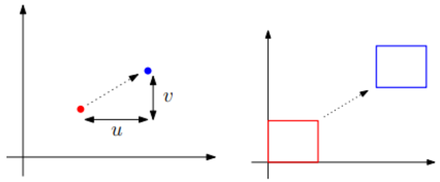
- $\frac{dy_1}{dt} = g_1 y_1 - g_3 y_1 y_2 = 0 \to \tilde{y}_2 = \frac{g_1}{g_3}$
- $\frac{dy_2}{dt} = g_2 y_2 + g_3 y_1 y_2 = 0 \to \tilde{y}_1 = \frac{g_2}{g_3}$

Basic Transformations

Representing translations as matrix multiplications

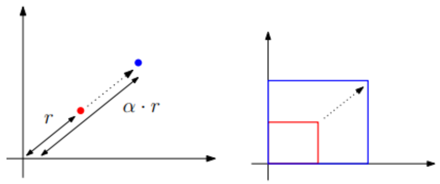
Translation by $(u; v)$

$$\begin{pmatrix} x_{new} \\ y_{new} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & u \\ 0 & 1 & v \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_{old} \\ y_{old} \\ 1 \end{pmatrix}$$



Scaling by a factor α

$$\begin{pmatrix} x_{new} \\ y_{new} \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} x_{old} \\ y_{old} \\ 1 \end{pmatrix}$$



Rotation around a point $(a; b)$

$$\begin{pmatrix} x_{new} \\ y_{new} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -a \\ 0 & 1 & -b \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_{old} \\ y_{old} \\ 1 \end{pmatrix}$$

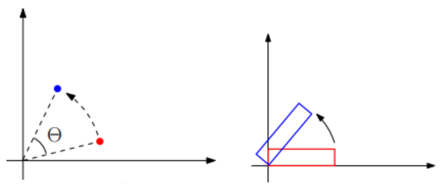


Image Processing using Filters

Goal Restore and or Modify Images

Moving average

Replace each pixel value with the weighted average of its neighborhood.

$$Im = Image, \qquad F = Kernel\ Filter, \qquad len(F) = 2N + 1$$

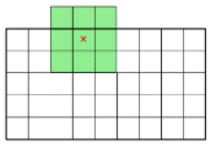
$$J(x, y) = \sum_{k=-N}^{N} \sum_{I=-N}^{N} Im(x + k, y + I) \cdot F(N + k, N + I) \qquad \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Key Properties

- Shift invariance $\quad F\big(shift(I)\big) = shift\big(F(I)\big)$
- Linearity* $\quad F(I_1 + I_2) = F(I_1) + F(I_2)$

Problem

Not specified for pixels close to the edge. For example, if the neighborhood of the marked pixel is outside of the boundary.



Solutions

- Treat pixels outside as 0
- Wrap around pixels from the opposite edge
- Treat like nearest pixel

Image Processing using Filters

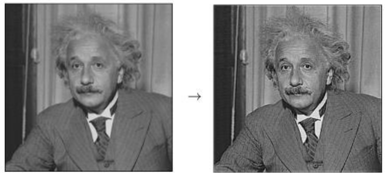
Average Filter (Blur)

$$\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

Sharpening Filter (Sharpening)

- 2 * original image – blurred image

$$\begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} + \begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix}$$



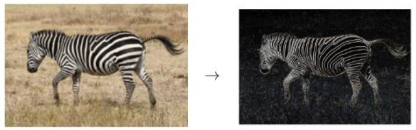
Gradient Filter

- Gradient $grad(f)$ Measures how the function is changing
- Magnitude $|grad(f)|$ Measures how quickly the function is changing

$$grad(f) = \begin{pmatrix}\frac{\delta f}{\delta x}\\ \frac{\delta f}{\delta x}\end{pmatrix}, \qquad |grad(f)| = \sqrt{\left(\frac{\delta f}{\delta x}\right)^2 + \left(\frac{\delta f}{\delta y}\right)^2}$$

$$\frac{\delta f}{\delta x} = \begin{bmatrix}-1&0&1\\-1&0&1\\-1&0&1\end{bmatrix}$$

$$\frac{\delta f}{\delta y} = \begin{bmatrix}-1&-1&-1\\0&0&0\\1&1&1\end{bmatrix}$$

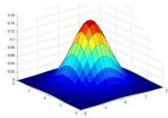


Gaussian Filter (Systematic Blur)

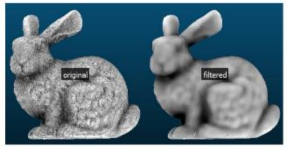
Choose the weights of the neighborhood pixels such that their contribution decreases with growing distance from the center.

Density function of Gaussian distribution

$$G_\sigma = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}, \qquad 2\cdot\sigma \approx 0.5\cdot len(F)$$



0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003



Median Filter

Median filter operates over a window by selecting the median intensity in the window.

10	15	20
23	90	27
33	31	30

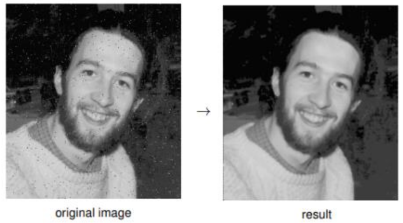
Sort

Median value

10 15 20 23 $\boxed{27}$ 30 31 33 90

Replace

10	15	20
23	27	27
33	31	30



Naive Bayes Approach

Bayes Filter

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Formal Description

- Events are described by a feature vector $X = (X_1, \ldots, X_n)$
- Variables X_i have to be independent
- Prediction Variable $Y \in \{0,1\}$

Calculated estimations

- Step 1.1 $P(X|Y=0)\ \prod_{i=1}^{n} P(X_i|Y=0)$
- Step 1.2 $P(X|Y=1)\ \prod_{i=1}^{n} P(X_i|Y=1)$
- Step 2.1 $P(Y=0|X)\ \frac{P(X|Y) \cdot P(Y=0)}{P(X)}$
- Step 2.2 $P(Y=1|X)\ \frac{P(X|Y) \cdot P(Y=1)}{P(X)}$
- Step 3.1 $P(Y=0|X) > P(Y=1|X) \rightarrow predict\ Y = 0$
- Step 3.2 $P(Y=0|X) < P(Y=1|X) \rightarrow predict\ Y = 1$

Day	Outlook	Temperature	Humidity	Wind	Play Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Prediction - Toy example

$$X := \left(\underbrace{Sunny}_{:=X_1}, \underbrace{Cool}_{:=X_2}, \underbrace{High}_{:=X_3}, \underbrace{Strong}_{:=X_4}\right)$$

Step 1

$$P(Play = Yes) = \frac{9}{14}, \qquad P(Play = No) = \frac{5}{14}$$

$$P(X_i|Play = Yes) = \prod_{i=1}^{n} P(X_i|Y = Yes) = \frac{2}{9} \cdot \frac{3}{9} \cdot \frac{3}{9} \cdot \frac{3}{9} = 0.00823, \qquad for\ i = 1, \ldots, 4$$

$$P(X_i|Play = No) = \prod_{i=1}^{n} P(X_i|Y = No) = \frac{3}{5} \cdot \frac{1}{5} \cdot \frac{4}{5} \cdot \frac{3}{5} = 0.0576, \qquad for\ i = 1, \ldots, 4$$

Step 2

$$P(X_i \wedge Play = Yes) = 0.00823 \cdot \frac{9}{14} = 0.0053$$

$$P(X_i \wedge Play = No) = 0.0576 \cdot \frac{5}{14} = 0.0206$$

$$P(Play = Yes|X_i) = \frac{P(X|Y) \cdot P(Y = Yes)}{P(X_i)} = \frac{P(X_i \wedge Play = Yes)}{P(X_i)} = \frac{0.0053}{0.02186} = 0.242$$

$$P(Play = No|X_i) = \frac{P(X|Y) \cdot P(Y = No)}{P(X_i)} = \frac{P(X_i \wedge Play = No)}{P(X_i)} = \frac{0.0206}{0.02186} = 0.942$$

Step 3 Compare the two results of Step 2 and choose the more likely event.

$$\max\big(P(Play = Yes|X_i), P(Play = No|X_i)\big) = \max(0.242, 0.942) = P(Play = No|X_i)$$

Predict $(Play\ = No)$ *for* X_i

Clustering (Unsupervised Learning)

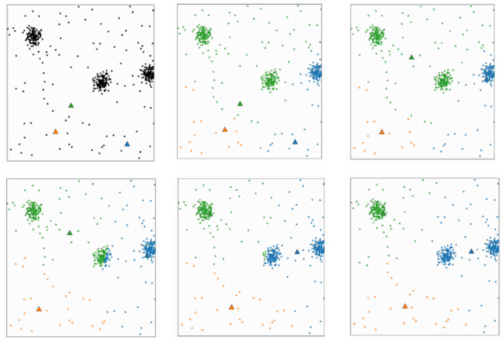
K-Means Clustering

Step 1: Choose k objects as initial cluster centers.

Step 2: Assign each data point to the cluster which has the closest mean point (centroid) under chosen distance metric.

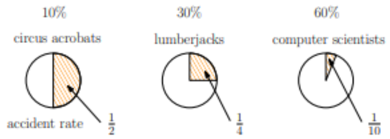
Step 3: When all data points have been assigned, recalculate the positions of k centroids (mean points).

Step 4: Repeat steps 2 and 3 until the centroids do not change anymore.



Maximum Likelihood Estimation

Given the following example



Problem 1: What is the *most likely profession* of a person *who had an accident*?

- $p(acrobat \wedge accident) = 0.1 \cdot 0.5 = 0.05$
- $p(lumberjack \wedge accident) = 0.3 \cdot 0.25 = \mathbf{0.075}$
- $p(computer\ scientist \wedge accident) = 0.6 \cdot 0.1 = 0.06$

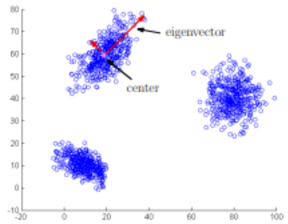
Problem 2: What is the *probability* that a random person has an *accident*?

- $p(accident) = 0.1 \cdot 0.5 + 0.3 \cdot 0.25 + 0.6 \cdot 0.1 = 0.185$
- $p(no\ accident) = 0.1 \cdot 0.5 + 0.3 \cdot 0.75 + 0.6 \cdot 0.9 = 0.815$

Problem 3: What is the probability of this constellation
$(5 \times no\ accident, 2 \times accident)$?

$$p = 0.815^5 \cdot 0.185^2 = 0.01$$

Gaussian Mixture Model



Input: number k of clusters

Parameters of the distribution:

- A priori probability p_i
- A «center» c_i
- A 2×2 covariance matrix S_i

Properties of the covariance matrix:

- Eigenvectors denote the main directions of the «spread of the data»
- Eigenvalues express the length of the corresponding eigenvectors

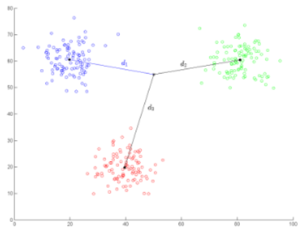
Silhouette value

The *silhouette value* is one of many measures to determine how good a clustering is. The larger the value, the better the point fits in the cluster.

$$s(p) = \frac{b(p) - a(p)}{\max(a(p), b(p))}$$

- $a(p)$ average *dist* to other points in cluster
- $b(p)$ minimum average *dist* to points in a different cluster

Nearest Mean



1. Determine the mean of each cluster
2. For each new point p:
   Find the cluster whose mean has the shortest distance to p and assign p to this cluster

Pros and Cons

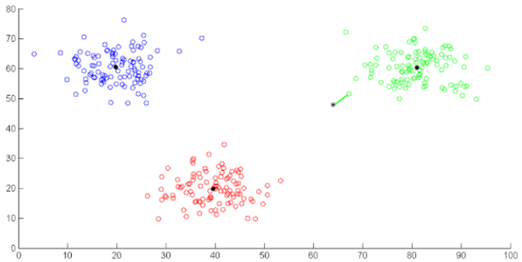
- Efficiently computable
- No further knowledge about the structure of the data is needed
- influenced by outliers
- mean is not always representative

(K-) Nearest Neighbor Classifier

- For each new point p:
  Determine the category point p is nearest to assign p to its cluster

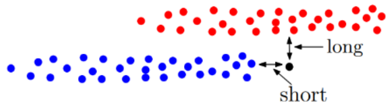
Pros and Cons

- More robust towards outliers
- No further knowledge about the structure of the data is needed
- computationally expensive to find the nearest neighbour
- failure in case of different "spread of data" for different directions



Bayes Classifier based on the Gaussian Mixture Model

Use a distance measure with an appropriate scaling (with respect to the corresponding eigenvalues).



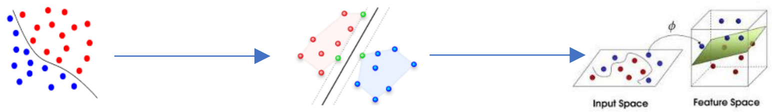
Implementation

- Model each class by a multivariate Gaussian distribution.
- Assign each point p according to the *maximum likelihood principle*
  - For each class i determine the *probability density* $pd_i(p)$ of p according to the corresponding distribution
  - Assign p to the class i for which $pd_i(p)$ is maximized

Support Vector Machines

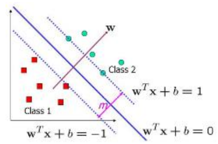
Basic Idea

- Finding the *best separating line* between two classes of data
- *Maximize the margin* between the line and the data
- Not linearly separable → Transform into *higher dimensional space*



Linearly separable problem

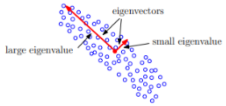
- Representation of a 2D line: $ax + b$ or $(a \quad -1) \cdot \binom{x}{y} + b = 0$
- Representation of a hyperplane $\omega^T x + b = 0$
- Goal: maximize m of the margin.
- It can be shown: $m = \frac{2}{||w||}$



Compression and Detection: Technical Aspects and Linear Algebra

Goal Reduce the number of dimensions without reducing the "information-content" too much.

Covariance Matrix

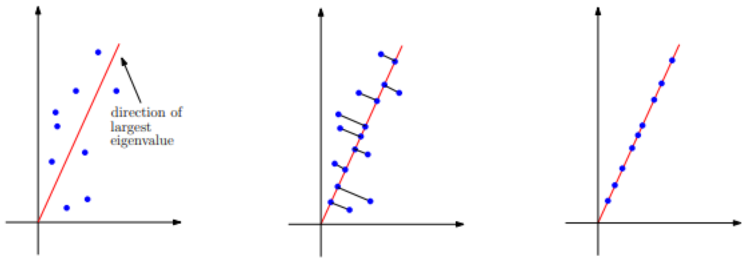


The *covariance matrix* C contains characteristic information

- Its *eigenvectors* express the main directions of the «spread of data»
- A large *eigenvalue* indicates a large amount of spread

Generalization to higher dimesions

- Determine *eigenvectors* with largest eigenvalues
- Maintain only *components* corresponding to those eigenvectors



Principle Component Analysis (PCA) Summary

1. Represent images as vectors
2. Compute *mean* and *covariance matrix* of the corresponding data
3. Normalize data by subtracting the mean-vector from each input-vector
4. Compress the data by maintaining only the components corresponding to the *largest eigenvectors*
5. Transform the vectors back

Change of Basis

Remark: A square matrix M whose colums are orthonormal vectors has the property that $M^{-1} = M^T$.

Transformation from standard basis B to a new basis M

$$\begin{pmatrix} \tilde{r}_1 \\ \vdots \\ \tilde{r}_d \end{pmatrix} = M^{-1} \cdot \begin{pmatrix} r_1 \\ \vdots \\ r_d \end{pmatrix}, \qquad M = (v_1 | v_2 | \cdots | v_n)$$

Reverse transformation from basis M to B

$$\begin{pmatrix} r_1 \\ \vdots \\ r_d \end{pmatrix} = M \cdot \begin{pmatrix} \tilde{r}_1 \\ \vdots \\ \tilde{r}_d \end{pmatrix}$$

Using a subset of the basis vectors gives *lossy transformations*

<u>Example</u> with kept vectors = 3

- Transformation $\begin{pmatrix} \tilde{r}_1 \\ \vdots \\ \tilde{r}_3 \end{pmatrix} = (v_1 | v_2 | v_3)^T \cdot \begin{pmatrix} r_1 \\ \vdots \\ r_d \end{pmatrix}$
- Reverse transformation $\begin{pmatrix} r_1 \\ \vdots \\ r_d \end{pmatrix} = (v_1 | v_2 | v_3) \cdot \begin{pmatrix} \tilde{r}_1 \\ \vdots \\ \tilde{r}_3 \end{pmatrix}$

A measure of similarity of two images

Sum of squared differences (SSD) $\quad I = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}, \quad J = \begin{pmatrix} 1 & 3 & 5 \\ 7 & 9 & 11 \end{pmatrix}$

$$SSD(I, J) = \\ (1-1)^2 + (2-3)^2 + (3-5)^2 + (4-7)^2 + (5-9)^2 + (6-11)^2 = 55$$

Compression and Detection: PCA Algorithm

Step 1: Preprocessing

$$C := covariance\ matrix\ of\ \underbrace{\begin{pmatrix} \cdots \\ \cdots \\ \vdots \\ \cdots \end{pmatrix}}_{d} \begin{matrix} \leftarrow data\ 1 \\ \leftarrow data\ 2 \\ \vdots \\ \leftarrow data\ N \end{matrix}$$

$$V_1, V_2, \ldots, V_k := eigenvectors\ of\ C\ with\ the\ largest\ eigenvalues$$

Step 2: Adjusting data

- Calculate the mean-vector
- Adjust data by subtracting the mean-vector from each data point

Step 3: Tranforming a data point

- Represent the adjusted data point as column vector
- Transform data point p via lossy change of basis

$$\begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_k \end{pmatrix} = (v_1 | v_2 | \cdots | v_k)^T \cdot \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_d \end{pmatrix}$$

Step 4: Reprojecting back to the original coordinate system

- Reproject back via lossy change of basis

$$\begin{pmatrix} \tilde{p}_1 \\ \tilde{p}_2 \\ \vdots \\ \tilde{p}_d \end{pmatrix} = (v_1 | v_2 | \cdots | v_k) \cdot \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_k \end{pmatrix}$$

Step 5: Add back mean

Example

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

Step 1: Preprocessing

- For $k = 1$: $V_1 = \begin{pmatrix} 0.6779 \\ 0.7352 \end{pmatrix}$

Step 2: Adjusting data

- Mean vector $= (1.81, 1.91)$
- Result for the first three datapoints

data:

x	y
2.5	2.4
0.5	0.7
2.2	2.9

adj.:

x	y
0.6900	0.4900
-1.3100	-1.2100
0.3900	0.9900

…

Advantages

Often, only a few eigenvectors are necessary to get a good compression → allows to *efficiently store and compare* images.

Matching will typically work better because only main characteristics are preserved and irrelevant details are discarded.

Drawback

Differences caused by varying illumination can become more substantial than differences between faces.

An Application: Skin Cancer Detection

ABCD rule (methods generally use a combination of defined visual clues and indicators to assess the risk of a skin lesion)

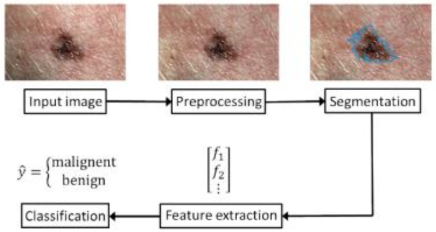
- A Asymmetry $[0-2]$ 0 = symmetric, 2 = asymmetric
- B Border $[0-8]$ presence of border irregulations in 8 regions
- C Color $[1-6]$ presence of one to six specific colors
- D Diameter or Differential Structure $[1-5]$ presence of one to five distinct structures or textures

Evaluation	TDS Score
Benign	<4.75
Suspicious	4.75 to 5.45
Malignant	>5.45

TDS-Algorithm $TDS = 1.3 \cdot A + 0.1 \cdot B + 0.5 \cdot C + 0.5 \cdot D$

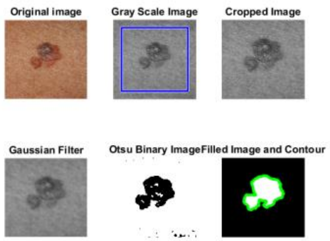
Worflow to develop a computer aided diagnosis system for malignant meloma

1. Input image
2. Preprocessing
3. Segmentation
4. Feature extraction $(f_1, f_2, f_3, \dots, f_n)^T$
5. Classification



Picture Segmentation

- Color → Gray Grey Scale Image
- Crop image Cropped Image
- Blur image Gaussian Filter
- Convert to binary image Otsu Binary
- Fill holes & find contours Filled Image



Feature Extraction

Calculate color score and variance

$$var = \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2, \qquad \mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Calculate the Asymmetry

- Calculate an array of radii $r_i = (r, \alpha) = (x, y)$ for each of 360°
- For each of the 360° radii r_i a score is calculated by comparing the lengths of pairs of radii that are symmetric across r_i.
- If a pair of radii have a difference of 0.1 or less, than a point is given. The sum of points is the SFA_i for r_i.
- The radius with the maximum SFA score is defined as the major axis of symmetry.
- The SFA of the major axis as well as the perpendicular are stored.

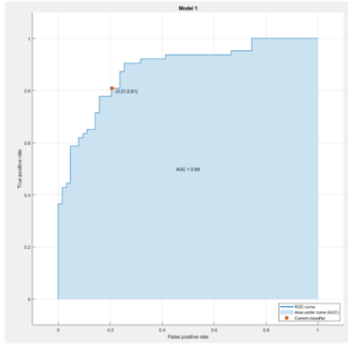
SFA Results	Description	Asym. Score
major axis $\geq$140 and minor axis $\geq$ 140	Symmetric across both axes	0
major axis $\geq$140 and minor axis < 140	Symmetric across one axis	1
major axis <140 and minor axis < 140	Asymmetric	2

Confusion Matrix: Rows *show* the *true class* and *colums* show the *predicted class*.

- Accuracy correct predictions / total predictions $\frac{TP+TN}{TP+TN+FP+FN}$
- Precision $\frac{TP}{TP+FP}$

Classification

- ROC **R**eceiver **O**perating **C**haracteristics Curve
- FPR / TPR **T**rue / **F**alse **P**ositive **R**ate
- AUC **A**rea **U**nder **C**urve: number is a measure of the overall quality of the classifier



Neurons

- Basic unit of a multilayer perceptron (MLP)
- Weighted sum of signals arriving at the input is subjected to a transfer function
- Several different transfer functions can be used. The one that is preferred in this chapter is the so-called sigmoid defined by the following formula where Σ is the weighted sum of inputs

$$f(\Sigma) = \frac{1}{1+e^{-\Sigma}}$$

Artificial Neural Network: Error

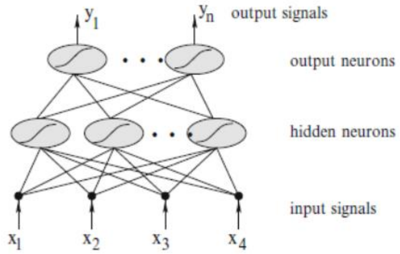
Mean square error (MSE): The mean square error is defined using differences between the elements of the output vector y and the target vector t:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(t_i - y_i)^2$$

Neural Networks

MLP Multilayer Perceptron

- There is no communication between neurons of the same layer, adjacent layers are fully interconnected
- Each link is associated with a weight
  - w_{ji}: weight of the link form the j-th hidden neuron to the i-th output neuron
  - w_{kj}: weight of the link from the k-th attribute to the j-th hidden neuron



Forward Propagation

- $x = (x_1, x_2, \dots, x_n)$: input attribute vector (e.g. x_k is one of the features calculate for one image)
- The values x_k are multiplied by the weights associated with the links
- The *j-th* hidden neuron then receives as input the weighted sum

$$\Sigma_k w_{kj}^{(2)} x_k$$

  And subjects this sum to the sigmoid

$$f\left(\Sigma_k w_{kj}^{(2)} x_k\right)$$

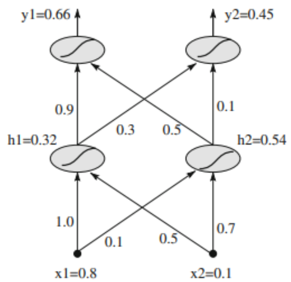
- The *i-th* output neuron then receives the weighted sum of the values coming from the hidden neurons and, agani subjects it to the transfer function.

$$y_i = f\left(\sum_j w_{ji}^{(1)} \cdot f\left(\sum_k w_{kj}^{(2)} x_k\right)\right)$$

Theorem: The so-called *universality theorem* is that the multilayer perceptron can in principle be used to address just about any classification problem.

Example Neural Network - Multilayer Perceptron

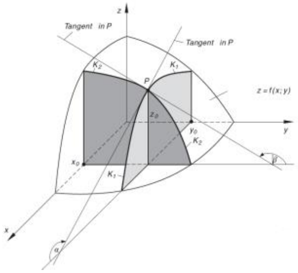
- Inputs of hidden-layer neurons $z_1^{(2)} = 0.8 \cdot (-0.1) + 0.1 \cdot 0.5 = -0.75$ $z_2^{(2)} = 0.8 \cdot 0.1 + 0.1 \cdot 0.7 = 0.15$
- Outputs of hidden layer neurons $h_1 = f\left(z_1^{(2)}\right) = \frac{1}{1+e^{-(-0.75)}} = 0.32$ $h_2 = f\left(z_2^{(2)}\right) = \frac{1}{1+e^{-0.15}} = 0.54$
- Inputs of output-layer neurons $z_1^{(1)} = 0.54 \cdot 0.9 + 0.32 \cdot 0.5 = 0.65$ $z_2^{(1)} = 0.54 \cdot -0.3 + 0.32 \cdot -0.1 = -0.19$
- Outputs of output-layer neurons $y_1 = f\left(z_1^{(1)}\right) = \frac{1}{1+e^{-0.65}} = 0.66$ $y_1 = f\left(z_1^{(1)}\right) = \frac{1}{1+e^{-(-0.19)}} = 0.45$



Partial Differential Equations - PDE

Partial derivatives of first order

Consider a scalar-valued function $y = f(x,y)$. f_x and f_y are the partial derivatives of f with respect to x and y.



- $\frac{\partial f}{\partial x}(x,y) = f_x = \lim_{\Delta x \to 0} \frac{f(x+\Delta x,y)-f(x,y)}{\Delta x}$
- $\frac{\partial f}{\partial y}(x,y) = f_y = \lim_{\Delta y \to 0} \frac{f(x,y+\Delta y)-f(x,y)}{\Delta y}$

The *gradient* of the function f is defined as

- $grad(f) = \nabla f = \left(\frac{\partial f}{\partial x} \quad \frac{\partial f}{\partial y}\right)^T$

where ∇ is called *Nabla-Operator.*

Partial derivatives of higher order

$$z = f(x,y), \qquad f_x = \frac{\partial f}{\partial x}, \qquad f_y = \frac{\partial f}{\partial y}$$

Partial derivatives of second order

$$f_{xx} = \frac{\partial f}{\partial x}\left(\frac{\partial f}{\partial x}\right) = \frac{\partial^2 f}{\partial x^2}, \qquad f_{yy} = \frac{\partial f}{\partial y}\left(\frac{\partial f}{\partial y}\right) = \frac{\partial^2 f}{\partial y^2}$$

$$f_{xy} = \frac{\partial f}{\partial y}\left(\frac{\partial f}{\partial x}\right) = \frac{\partial^2 f}{\partial x \partial y}, \qquad f_{yx} = \frac{\partial f}{\partial x}\left(\frac{\partial f}{\partial y}\right) = \frac{\partial^2 f}{\partial y \partial x}$$

Schwartz' Theorem

Consider the function $f: \mathbb{R}^n \to \mathbb{R}$. If the mixed partial derivatives exist and are continuous at a point $x_0 \in \mathbb{R}^n$, then they are equal at x_0 regardless of the order in which they are taken.

Laplacian

Consider a scalar-valued function $f: \mathbb{R}^n \to \mathbb{R}, y = f(x_1, \dots, x_n)$

The *Laplacian* operator Δ is defined as

$$\Delta = \nabla \cdot \nabla = \sum_{i=1}^{n} \frac{\partial^2}{\partial x_i^2} = \left(\frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \dots + \frac{\partial^2}{\partial x_n^2}\right)$$

i.e.

$$\Delta f = f x_1 x_1 + f x_2 x_2 + \dots + f x_n x_n = \left(\frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \dots + \frac{\partial^2}{\partial x_n^2}\right)$$

Fourier Series

Let $f: \mathbb{R} \to \mathbb{R}$ be a periodic continuous function with angular frequency ω_0 and period $T = \frac{2\pi}{\omega_0}$. The Fourier series of $f(x)$ is defined as

$$f(x) = \frac{A_0}{2} + \sum_{k=1}^{\infty}[A_k \cdot \cos(k \cdot \omega_0 \cdot x) + B_k \cdot \sin(k \cdot \omega_0 \cdot x)]$$

- $\omega_0 = \frac{2\pi}{T}$ angular frequency of the first harmonic oscillation
- $k \cdot \omega_0$ angular frequency of the $k - th$ harmonic oscillation

The Fourier coefficients of f can be calculated according to

- $A_0 = \frac{2}{T}\int_{(T)} f(x) \cdot dx,$
- $A_k = \frac{2}{T}\int_{(T)} f(x) \cdot \cos(k \cdot \omega_0 \cdot x) \cdot dx$
- $B_k = \frac{2}{T}\int_{(T)} f(x) \cdot \sin(k \cdot \omega_0 \cdot x) \cdot dx$

Partial Differential Equations - PDE

Ordinary Differential Equation (ODE)

$$\frac{d^2\Theta}{dt^2} + \frac{g}{L} \cdot \sin(\Theta) = 0$$

Partial Differential Equation (PDE)

$$\Delta\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0$$

Partial Differential Equation (PDE) - Definition

A partial differential equation PDE relates the partial derivatives of a function of two or more independent variables together. The order of the highest partial derivative is the order of the equation. We say a function is a solution to a PDE if it satisfies the equation and any side conditions given.

Linear, homogenous PDE of first order

If the coefficients $a = a(x, y), b = b(x, y)$ are continuously differentiable functions $a: D \subset \mathbb{R}^2 \to \mathbb{R}$ and $b: D \subset \mathbb{R}^2 \to \mathbb{R}$, then the PDE for $u = u(x, y)$

$$a(x, y) \cdot u_x + b(x, y) \cdot u_y = 0$$

Is a linear homogenous PDE of first order. Homogenous means that the right-hand side of the PDE vanishes.

Linear PDE of second order and their classification

Consider the continuously differentiable functions $a, b, c, d, e, f, g: D \subset \mathbb{R}^2 \to \mathbb{R}$. The PDE for $u = u(x, y)$

$$a(x, y) \cdot u_{xx} + 2b(x, y) \cdot u_{xy} + c(x, y) \cdot u_{yy} =$$

$$d(x, y) \cdot u_x + e(x, y) \cdot u_y + f(x, y) \cdot u + g(x, y)$$

Is a linear (non-homogenous) PDE of second order.

The linear PDE is said to be

- Parabolic if $b^2 - ac = 0$ e.g. heat flow and diffusion-type
- Hyperbolic if $b^2 - ac > 0$ e.g. vibrating and wave motion
- Elliptic if $b^2 - ac < 0$ e.g. steady-state potential-type

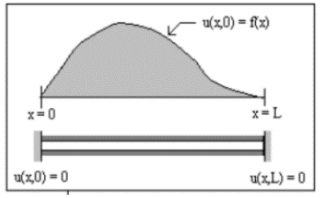
<u>Example</u>: **Heat Transfer Equation**

The heat transfer equation is a parabolic PDE that describes the temperature variation u as a function of time and special coordinates (k is a constant describing thermal diffusivity).

Heat Equation in 1d

We search for a twice continuously differentiable function $u = u(x, t)$ which solves the heat transfer equation

$$u_t = ku_{xx}$$



Subject to the initial condition

$$u(x, 0) = f(x)$$

And the constant-value boundary conditions

$$u(0, t) = 0, \quad u(L, t) = 0, \quad x \in [0, L], \quad t \geq 0$$

Computational Fluid Dynamics

C*omputational* ***F****luid* ***D****ynamics* (CFD) is the analysis of systems involving fluid flow, heat transfer and associated phenomena such as chemical reactions by means of computer-based simulation.

The aim of CFD simulations is to determine…

- Velocity field $\boldsymbol{u} = \boldsymbol{u}(x, y, z, t)$
- Pressure distribution $p = p(x, y, z, t)$
- Density distribution $\rho = \rho(x, y, z, t)$
- Temperature distribution $T = T(x, y, z, t)$

…of a fluid flow at any given point (x, y, z) and at any given time t.

Conversation laws of physics

1. The mass of a fluid is conserved.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0$$

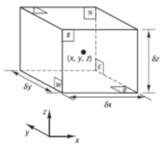
2. The rate of change of momentum equals the sum of the forces on a fluid particle.

$$\dot{\boldsymbol{p}} = m\dot{\boldsymbol{v}} = \sum_{i=1}^{n} \boldsymbol{F}_i$$

3. The rate of change of energy is equal to the sum of the rate of heat addition to the rate of work done on a fluid particle.

To derive the governing PDE for each conservation law, consider a small element of fluid with sides $\delta x, \delta y$ and δz.

- Volume $V = \delta x \delta y \delta z$
- Mass $m = \rho V = \rho \delta x \delta y \delta z$



General Structure of Governing Equations

- Accumulation: Temporal rate of change of a given quantity within V
- Convection: Transport of the quantity due to any existing velocity field
- Diffusion: Transport of the quantity due to the presence of and gradients of that quantity.

$$\underbrace{\frac{\partial \rho \phi}{\partial t}}_{Accumulation} + \underbrace{\nabla \cdot (\rho \boldsymbol{u} \phi)}_{Convection} = \underbrace{\nabla \cdot (\Gamma \nabla \phi)}_{Diffusion} + \underbrace{S_\phi}_{Source}$$

Rate of increase of ϕ of fluid element	+ Net rate of flow of ϕ out of fluid element	= Rate of increase of ϕ due to diffusion	+ Rate of increase of ϕ due to sources

Full Set of Governing Equations

- Conservation of mass $\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0$
- Conservation of momentum (Navier – Stokes)
  - $\frac{\partial \rho u}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} u) = \nabla \cdot (\mu \nabla u) - \frac{\partial p}{\partial x} + S_{Mx}$
  - $\frac{\partial \rho v}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} v) = \nabla \cdot (\mu \nabla v) - \frac{\partial p}{\partial x} + S_{My}$
  - $\frac{\partial \rho w}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} w) = \nabla \cdot (\mu \nabla w) - \frac{\partial p}{\partial z} + S_{Mz}$
- Conservation of energy $\frac{\partial \rho i}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} i) = \nabla \cdot (k \nabla T) - p \nabla \boldsymbol{u} + S_{Dis}$
- Equations of state
  - $p = p(\rho, T)$
  - $i = i(\rho, T)$

OpenFoam

General Commands

Change to *RUN* directory

cd $FOAM_RUN

Copy tutorial

cp -r $FOAM_TUTORIALS/name .

Change to newly created directory

cd name

Create mesh of case (= name)

case/blockMesh

Check mesh case (= name)

case/checkMesh

View mesh

paraFoam

Run commands in Allrun

./Allrun

Multiple Cores

Create decomposePar file

cp -r $FOAM_UTILITIES/parallelProcessing/decomposePar .

Create decomposeParDict (copy from pitzDaily/system or damBreak/system)

...

Run DecomposePar

decomposePar

Run the case in parallel using the solver XXX

mpirun -np 4 XXX -parallel | tee log

reconstruct the result from the single processors

reconstructPar

Have a look at the result with paraFoam

paraFoam &

XLaunch