UiO : Department of Informatics
University of Oslo

INF 5860 Machine learning for image classification

Lecture 1b : Introduction to classification

Anne Solberg

Practical information

- Lecturers:
  - Sigmund Rolfsjord: sigmunjr@ifi.uio.no
    - Office 44
  - Anne Schistad Solberg: anne@ifi.uio.no
    - Office 4458, Phone 22862435
  - Group lectures: led by one of us, room Modula
    - First time: Monday 23.1. 10.15-12: Python/Numpy for image analysis

Mandatory exercises:

- Required: use Python and Tensorflow!
- Exercise 1: Implement basic neural nets and backpropagation in Python.
- Exercise 2: Implement Convolutional neural nets in Tensorflow.

Web page

- http://www.uio.no/studier/emner/matnat/ifi/inf5860/2017/index.html
- Lectures with links to lecture foils, curriculum, weekly exercises, and mandatory exercises.
- Messages
- Urgent messages sent to studenter.inf5860@ifi.uio.no, studenter.inf9860@ifi.uio.no, which links to your Ifi/uio official email – read this!

Who is this course for?

- Master and PhD students working in image analysis problems using deep learning

Curriculum

- Lecture notes, weekly exercises and mandatory exercises defines the curriculum.
- The field is so new and rapidly changing that no printed book covers it well.
- Parts of the lecture are based on:
  - Deep learning, Ian Goodfellow, Yoshua Bengio, Aaron Courville, available at http://www.deeplearningbook.org/
- Each lecture will give links to relevant literature
- Very good material in the Stanford courses CS 229 and CS 231n. Links will be given.

Required background

- Good knowledge in Python programming
- Experience in image analysis programming
  - Sliding windows, convolution, edge detection, basic pixel-based classifcation.
- Good knowledge in mathematics
  - Builds on MAT 1110, MAT 1120
  - Linear algebra
  - Gradient-based optimization
  - Gradients and derivation

Linear algebra *required* knowledge

- Vectors
- Dot products
- Matrix multiplication
- Matrix transpose and inversion

Example formula

$$\boxed{p(\mathbf{x} \mid \omega_s) =} \boxed{\frac{1}{(2\pi)^{d/2} \left|\boldsymbol{\Sigma}_s\right|^{1/2}}} \exp\left[-\frac{1}{2} \boxed{(\mathbf{x} - \boldsymbol{\mu}_s)^t} \boxed{\boldsymbol{\Sigma}_s^{-1}} \boxed{(\mathbf{x} - \boldsymbol{\mu}_s)}\right]$$

Scalar probability

Scalar

1xn vector transposed

nx1 vector transposed

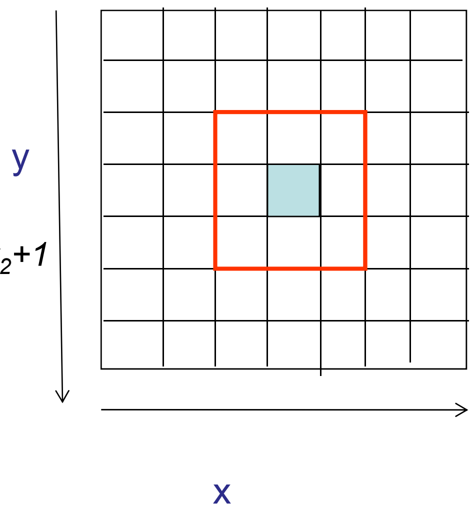
nxn matrix
Inverse of covariance matrix

2D convolution: required knowledge

- Input image f(i,j).Output image g(i,j):

$$g(x,y) = \sum_{j=-w_1}^{w_1} \sum_{k=-w_2}^{w_2} h(j,k) f(x-j, y-k)$$

$$= \sum_{j=x-w_1}^{x+w_1} \sum_{k=y-w_2}^{y+w_2} h(x-j, y-k) f(j,k)$$



- h is $m \times n$ filter with size $m=2w_1+1$, $n=2w_2+1$
- *Implement this is this week's exercises*

Required: image gradients

- The gradient in a point has a direction and a magnitude.
- The direction is the direction with maximum slope, and the magnitude is the slope in that direction.

Required: image gradients

- Gradient of *F* along *r* in direction θ

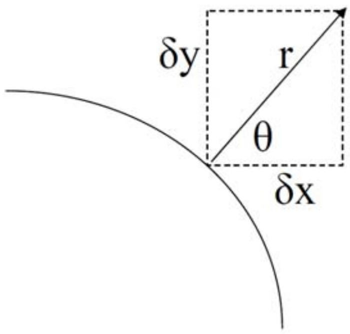
$$\frac{\partial F}{\partial r} = \frac{\partial F}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial F}{\partial y}\frac{\partial y}{\partial r}$$

$$\frac{\partial F}{\partial r} = \frac{\partial F}{\partial x}\cos\theta + \frac{\partial F}{\partial y}\sin\theta$$

- Largest gradient when $\frac{\partial}{\partial \theta}\left(\frac{\partial F}{\partial r}\right) = 0$
- So the angle θ_g where

$$-\frac{\partial F}{\partial x}\sin\theta_g + \frac{\partial F}{\partial y}\cos\theta_g = 0 \Leftrightarrow \frac{\partial F}{\partial y}\cos\theta_g = \frac{\partial F}{\partial x}\sin\theta_g$$

-



Required: Gradient magnitude and direction

- Gradient direction:

$$\boxed{\theta_g = \tan^{-1}\left(\frac{g_y}{g_x}\right)}$$

- Gradient magnitude:

$$\boxed{\left(\frac{\partial F}{\partial r}\right)_{\max} = \left[g_x^2 + g_y^2\right]^{1/2}}$$

Gradient-operators

- Prewitt-operatoren

$$H_x(i,j) = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} H_y(i,j) = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

- Sobel-operatoren

$$H_x(i,j) = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} H_y(i,j) = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Required: Gradient descent optimization

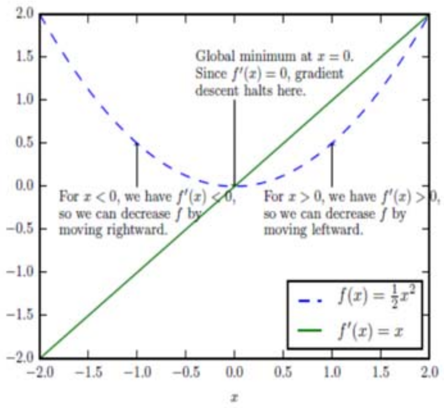
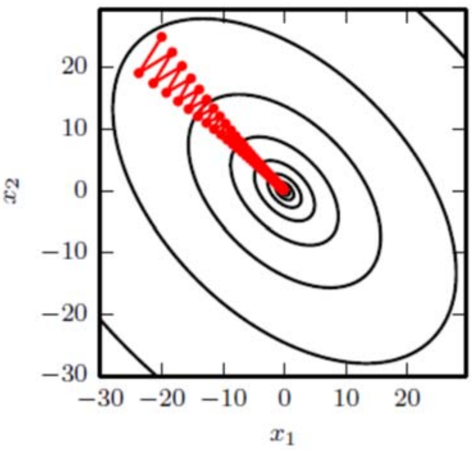
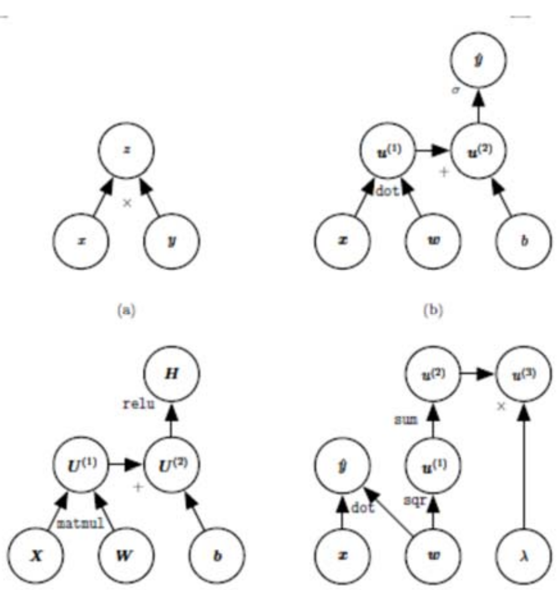


Figure 4.1



Computational graphs and derivation



Derivation using the chain rule to understand backpropagation

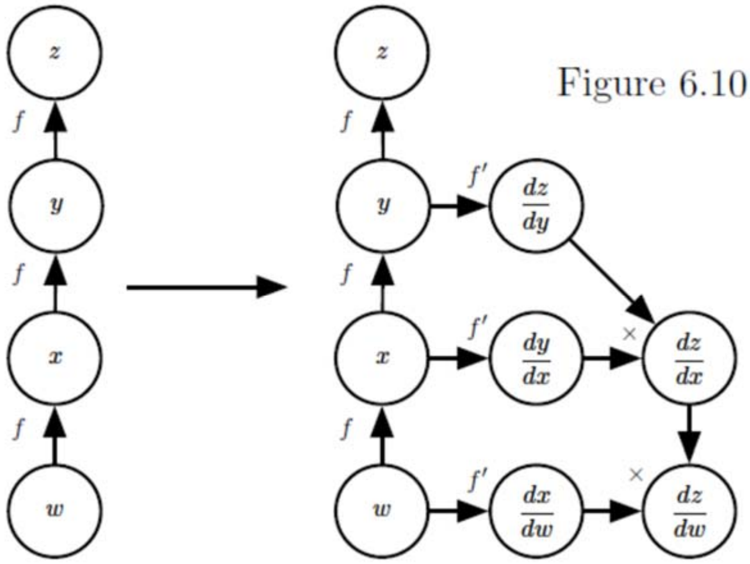


Figure 6.10

Image classification - introduction

→ DOG

Task: use the entire image to classify the image into one of a set of known classes

Which object does the image contain

Find ONE object in the image, no precise localization on WHERE in the image

Input to the classifier/net is ALL pixels in ALL bands/channels (RGB in this case)

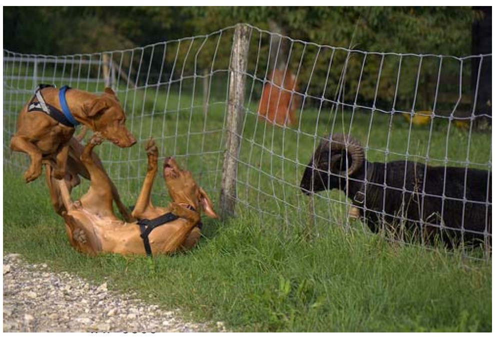
Challenges - Illumination

Challenges - occlusion

Challenges – background clutter

UiO : Department of Informatics
University of Oslo

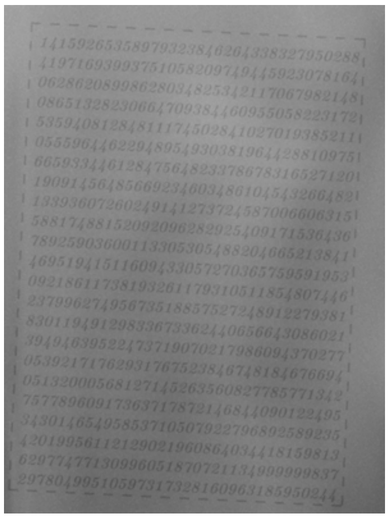
Challenges - deformation



Challenges – intraclass variation

Conventional approach to image classification

- Goal: get the series of digits, e.g. 14159265358979323846……

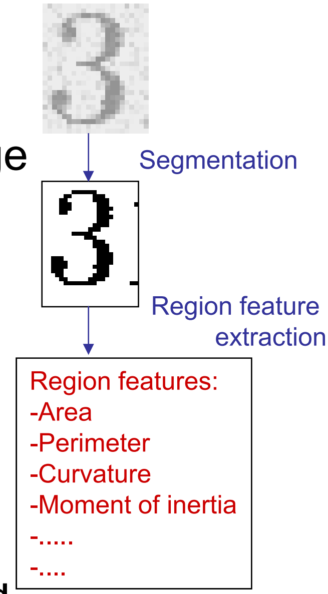


Steps in the program:

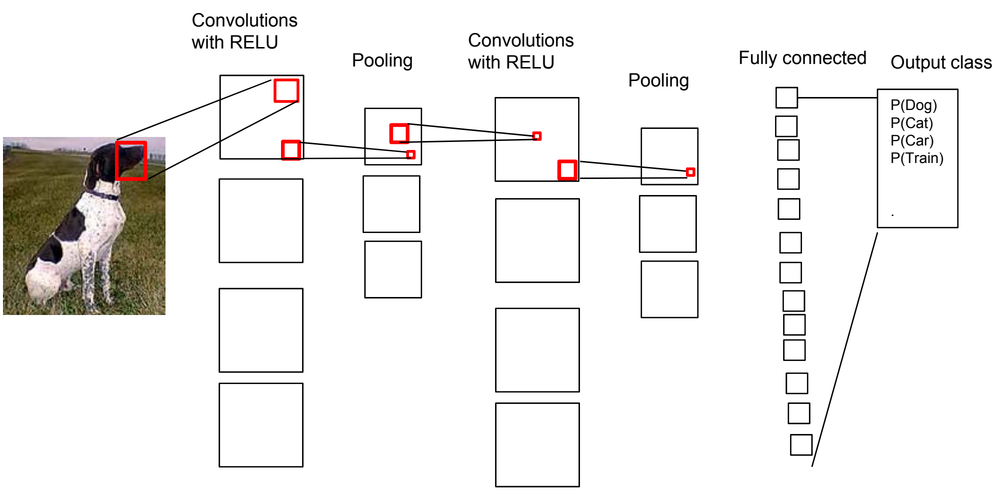
1. **Segment** the image to find digit pixels.
2. Find angle of rotation and rotate back.
3. Create region objects – **one object pr. digit** or connected component.
4. **Compute features** describing shape of objects
5. Train a classifier on many objects of each digit.
6. Assign a class label to each new object, i.e., the class with the highest probability.

Conventional feature-based classifcation

- Input to the classifier is a set of features derived from the image data, not the image data itself.
- We would either:
  - Segment the image to identify each object, then extract features from the objects pixels, and classify the object..
  - OR: Compute feature for each pixel in a sliding window around each pixel (e.g. texture features)
  - Classification would be done pixel-by-pixel.
  - **Deep learning can do the feature extraction for us, and classify the entire image.**
  - **The simplest task is to have images containg ONE object, and classfy that object.**



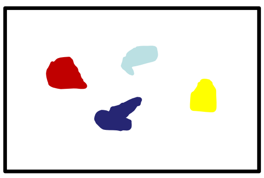
Classification using a convolutional neural net



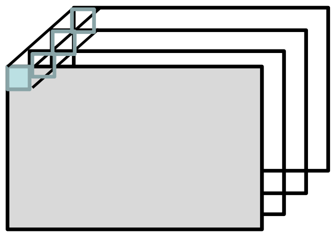
Repetition of pixel-based classification (from INF 4300)

Repetition of pixel-based classification (from INF 4300)

- x_i – feature vector for pixel i
- ω_{i}- The class label for pixel i
- K – the number of classes given in the training data



Mask with training pixels



Multiband image with
n spectral channels or features

$$p(\mathbf{x} \mid \omega_s) = \frac{1}{(2\pi)^{n/2} \left|\boldsymbol{\Sigma}_s\right|^{1/2}} \exp\left[-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_s)^t \boldsymbol{\Sigma}_s^{-1} (\mathbf{x} - \boldsymbol{\mu}_s) \right]$$

Concepts in classification

- Supervised classification: a dataset with known class labels is used to train the classifier.
  - Training set: used to estimate parameters of the classifier
  - Validation set: used to estimate hyperparameters
  - Test set: ONLY used at last to estimate the final classification accuracy
  - Classifier accuracy: computed by comparing estimated and true labels.
  - Computing the probability that an object belongs to a class.
    - Let each class be represented by a probability density function. Assign a pixel to the class with the highest probability (or smallest distance)
  - Bayes rule

Bayes rule for a classification problem

- Suppose we have J, j=1,...J classes. ω is the class label for a pixel, and *x* is the observed gray level (or feature vector).
- We can use Bayes rule to find an expression for the class with the highest probability:

$$P(\omega_j \mid x) = \frac{p(x \mid \omega_j) P(\omega_j)}{p(x)}$$

$$\text{posterior probability} = \frac{\textit{likelihood} \times \text{prior probability}}{\text{normalizing factor}}$$

- If we don't have special knowledge that one of the classes occur more frequent than other classes, we set them equal for all classes. ($P(\omega_j)$=1/J, j=1.,,,J).
- Small p means a probability distribution
- Capital P means a probability (scalar value between 0 and 1)

Bayes rule explained

$$P(\omega_j \mid x) = \frac{p(x \mid \omega_j) P(\omega_j)}{p(x)}$$

- $p(x|\omega_j)$ is the probability density function that models the likelihood for observing gray level x if the pixel belongs to class $\omega_{j.}$
  - Typically we assume a type of distribution, e.g. Gaussian, and the mean and covariance of that distribution is fitted to some data that we know belong to that class.
- $P(\omega_j|x)$ is the posterior probability that the pixel actually belongs to class $\omega_{j.}$ We will soon se that the the classifier that achieves the minimum error is a classifier that assigns each pixel to the class ω_j that has the highest posterior probability.
- p(x) is just a scaling factor that assures that the probabilities sum to 1.

Probability of error

- If we have 2 classes, we make an error either if we decide ω_1 if the true class is ω_2 if we decide ω_2 if the true class is $\omega_{1.}$
- If $P(\omega_1|x) > P(\omega_2|x)$ we have more belief that x belongs to ω_1, and we decide $\omega_{1.}$
- The probability of error is then:

$$P(error \mid x) = \begin{cases} P(\omega_1 \mid x) \text{ if we decide } \omega_2 \\ P(\omega_2 \mid x) \text{ if we decide } \omega_1 \end{cases}$$

Bayes decision rule

- In the 2 class case, our goal of minimizing the error implies a decision rule:

  Decide ω_1 if $P(\omega_1|x)>P(\omega_2|x)$; otherwise ω_2
- For *J* classes, the rule analogusly extends to choose the class with *maximum a posteriori* probability
- The *decision boundary* is the"border" between classes *i* and *j*, simply where $P(\omega_i|x)=P(\omega_j|x)$

The Gaussian density - univariate case (a single feature)

- To use a classifier we need to select a probability density function $p(x|\omega_i)$.
- The most commonly used probability density is the normal (Gaussian) distribution:

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

with expected value (or mean) $\mu = \mathrm{E}[x] = \int_{-\infty}^{\infty} xp(x)dx$

and variance $\sigma^2 = E\left[(x-\mu)^2\right] = \int_{-\infty}^{\infty} (x-\mu)^2 p(x)dx$

Training a univariate Gaussian classifier

- To be able to compute the value of the discriminant function, we need to have an estimate of μ_j and σ_j^2 for each class j.
- Assume that we know the true class labels for some pixels and that this is given in a mask image. The mask has N_k pixels for each class.
- Training the classifier then consists of computing μ_j and σ_j^2 for all pixels with class label j in the mask file.
- They are computed from training data as:
- For all pixels x_i with label k in the training mask, compute

$$\mu_k = \frac{1}{N_k}\sum_{i=i}^{N_k} x_i$$

$$\sigma_k^2 = \frac{1}{N_k}\sum_{i=i}^{N_k} (x_i - \mu_k)^2$$

Training

```
for i=1:N
  for j=i:M
    if mask(i,j)>==K
      increment nof. Samples in class K
      store the feature vector f(i,j) in a vector of training samples from class K
   end
end
end
```

```
For class k=1:K
    compute mean(k) and sigma(k)
```

$$\mu_k = \frac{1}{N_k}\sum_{i=i}^{N_k} x_i$$

$$\sigma_k^2 = \frac{1}{N_k}\sum_{i=i}^{N_k} (x_i - \mu_k)^2$$

How do to classification with a univariate Gaussian (1 feature)

- Decide on values for the prior probabilities, $P(\omega_j)$. If we have no prior information, assume that all classes are equally probable and $P(\omega_j)=1/J$.
- Estimate μ_j and σ_j^2 based on training data based on the formulae on the previous slide.
- For each pixel:

  For class j=1,….J, compute the discriminant function

$$P(\omega_j \mid x) = p(x \mid \omega_j)P(\omega_j) = \frac{1}{\sqrt{2\pi}\sigma_j}\exp\left[-\frac{1}{2}\left(\frac{x-\mu_j}{\sigma_j}\right)^2\right]P(\omega_j)$$

  Assign pixel x to the class C with the highest value of $P(\omega_j|x)$ by setting label_image(x,y)= C

The result after classification is an image with class labels corresponding to the most probable class for each pixel.

We compute the classification error rate from an independent test mask.

Estimating classification error

- A simple measure of classification accuracy can be to count the percentage of correctly classified pixels overall (averaged for all classes), or per. class. If a pixel has true class label k, it is correctly classified if ω_j=k.
- Normally we use different pixels to train and test a classifier, so we have a **disjoint training mask and test mask.**
- Estimate the classification error by classifying all pixels in the test set and count the percentage of wrongly classified pixels.

Validating classifier performance

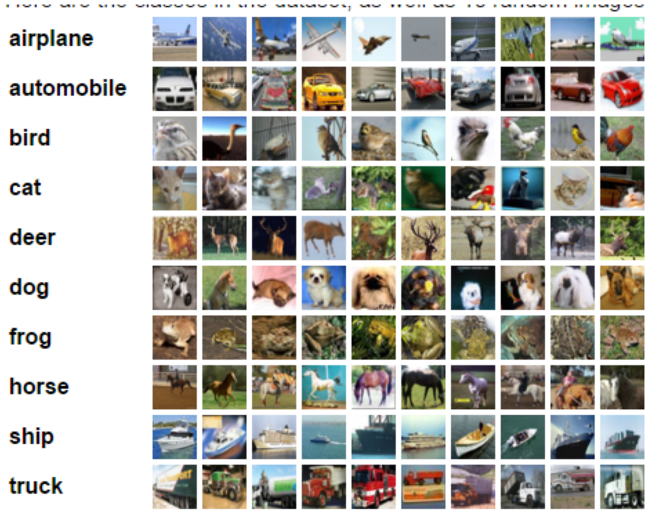
- Classification performance is evaluated on a different set of samples with known class - the test set.
- The training set and the test set must be independent!
- To set hyperparameters we also use a third dataset, the validation set
- The test set is ONLY used to estimate the final accuracy.

End of repetition

- Now: introduction to classification based on ALL PIXELS in the image
- Introducing the CIFAR-10 data set

The CIFAR image data set

10 classes
50 000 training images, size 32x32
10 000 test images



https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

Measuring similarity between two images

$$\text{L1} - \text{distance} : d_1(i,j) = \sum_j \sum_j \left| I_1(i,j) - I_2(i,j) \right|$$

$$\text{L2} - \text{distance} : d_1(i,j) = \sum_j \sum_j \sqrt{\left(I_1(i,j) - I_2(i,j) \right)^2}$$

L2 is called Euclidean distance

Nearest neighbour image classifier

- For a new image, compute the L1 or L2-distance to all the images in the training data set to find the single most similar image.
- Assign the image to the same class as the most similar image.
- Note: all training images must be stored.

k-Nearest-Neighbor classification

- Classification of a new sample x_i is done as follows:
  - Out of N training image, identify the *k* nearest neighbors (measured by L1 or L2) in the training set, irrespectively of the class label.
  - Out of these *k* samples, identify the number of images k_i that belong to class ω_i , *i:1,2,....M* (if we have *M* classes)
  - Assign x_i to the class ω_i with the maximum number of k_i samples.
- *k* should be odd, and must be selected a priori.
- The distance measure and k are hyperparameters.

About kNN-classification

- If *k=1* (1NN-classification), each sample is assigned to the same class as the closest sample in the training data set.
- If the number of training samples is very high, this can be a good rule.
- If k->∞, this is theoretically a very good classifier.
- This classifier involves no "training time", but the time needed to classify one pattern x_i will depend on the number of training samples, as the distance to all points in the training set must be computed.
- "Practical" values for *k*: *3<=k<=9*
- *Classification performance should **always** be computed on the test data set.*

Next week:

- Linear regression
- Introduction to loss functions and gradient descent.

- Classification as a regression problem with a learning rule
- Softmax-classification.